VIVO 1.10.x Documentation

VIVO 1.10.x Documentation

Exported on 04/22/2018

Table of Contents

1 Introduction

1.1 What is VIVO?

VIVO [Pronunciation: /viːvəʊ/ or *vee-voh*] is member-supported, open source software and an ontology for representing scholarship. VIVO supports recording, editing, searching, browsing, and visualizing scholarly activity. VIVO encourages showcasing the scholarly record, research discovery, expert finding, network analysis, and assessment of research impact. VIVO is easily extended to support additional domains of scholarly activity.

When installed and populated with researcher interests, activities, and accomplishments by an institution, VIVO enables the discovery of research and scholarship across disciplines at that institution and beyond. VIVO supports browsing and a search function which returns faceted results for rapid retrieval of desired information. Content in a VIVO installation may be maintained manually, brought into VIVO in automated ways from local systems of record, such as HR, grants, course, and faculty activity databases, or from database providers such as publication aggregators and funding agencies.

1.2 Release Notes

1.2.1 Version 1.10.0

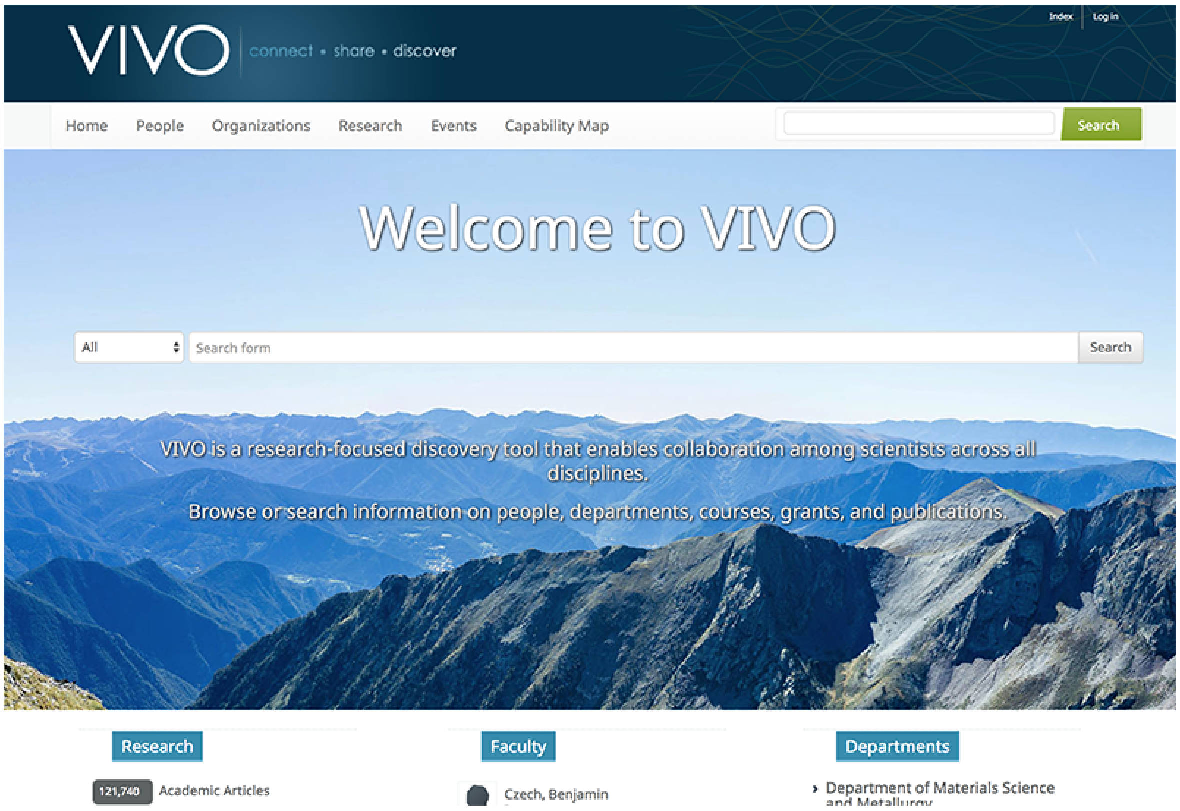
Please review the upgrade notes carefully

It is important that you reload the data in your triple store, as well as consider the impact that the dependency changes may have on your customisations and integrations.

There are no ontology changes, and data produced by 1.10.0 is compatible with 1.6 - 1.9 (and vice versa). It is not required to upgrade to this release prior to subsequent releases. Please see Upgrading VIVO (see page 40) for more details.

1.2.1.1 What's New

Bootstrap Theme - "Tenderfoot"


VIVO
connect • share • discover
Index
Log in
Home
People
Organizations
Research
Events
Capability Map
Search
Welcome to VIVO
All
Search form
Search
VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.
Browse or search information on people, departments, courses, grants, and publications.
Research
Faculty
Departments
121,740
Academic Articles
Czech, Benjamin
Department of Materials Science and Metallurgy

Tenderfoot is a new, responsive theme for VIVO. Based on the work by Symplectic, it uses Bootstrap 3 to provide a view that scales better to different sizes of devices.

Branding and Theming Improvements

It is no longer necessary to modify or create a new theme if all you want to do is apply some local CSS and/or message customisations to your installation. By applying local customisations on top of a theme, it allows for the possibility to swap compatible themes whilst retaining your site definitions, and easier upgrades in the future, where you don't need to merge changes to templates in a theme.

For theme developers, the remaining JSP pages now render their body and are wrapped by the Freemarker page structure, so you don't need to maintain a separate JSP page structure, and ensure that the structure matches that in Freemarker. This allows more flexibility in the theme structure, without adversely affecting the JSP pages.

Multi-Lingual Improvements

The message lookups have been extended, so that application wide VIVO messages are distinct from Theme messages (and distinct from Vitro messages). It also allows for an additional "local" messages bundle, which overrides the theme, VIVO and Vitro layers.

Language packs can now be added to VIVO through the dependency mechanism, although you will still need to edit your runtime.properties to enable the languages in your UI.

Linked Data Fragments

Linked Data Fragments presents a lightweight means of obtaining triples from a linked data application as a web service, with very low overhead, and high reliability (it only pattern matches for triples, there are no arbitrary complex queries, so individual requests can not have a high impact on the server).

ORCiD API v2

This release includes an updated ORCiD integration that can use the ORCiD v2 API. Note that ORCiD are planning to shut down the v1.x API endpoints.

Note that the configuration options have been changed, and you will need to update your runtime.properties.

The only options that are required now are:

```
orcid.clientId = 0000-0000-0000-000X
orcid.clientPassword = 00000000-0000-0000-0000-000000000000
orcid.webappBaseUrl = http://localhost:8080/vivo
orcid.externalIdCommonName = VIVO Cornell Identifier
orcid.apiVersion = 2.0
orcid.api = sandbox
```

orcid.apiVersion is simply the version value (e.g. 1.2, 2.0), and orcid.api is just "release" (for the production API), and "sandbox" for the sandbox.

Direct2Experts Endpoint

For any sites wishing to participate in the Direct2Experts federated site - http://direct2experts.org/ - VIVO now includes the necessary endpoints. Please see the Direct2Experts websites for more information on how to participate.

RDF 1.1 support

VIVO has now been updated to use Jena 3, which brings full RDF 1.1 support. As a result, internally all literals are treated as having a datatype - those with a language tag are rdf:langString, and any other are xsd:string, and the triple stores need to be reloaded to ensure that they have consistent internal representations. Only applications that directly open the triple stores using Jena libraries, or have SPARQL that explicitly references datatypes will be affected. Please see the upgrade notes for how to perform the migration and evaluate any impact.

In particular, if you are using VIVO Harvester, you will need to use a VIVO Harvester 2.x version.

Dependency Convergence and Vulnerability Elimination

Along with Jena 3 and Bootstrap, all the dependencies have been reevaluated and upgraded to ensure convergence.

In the back end, multiple conflicting versions of dependencies have been eliminated (e.g. httpclient, OSGi bundles), and all code now works with consistent versions of dependencies. Multiple JSON parsers have been removed, and all code now uses only Jackson.

Dependencies with known vulnerabilities, as determined by the Maven dependency-check plugin have been upgraded where a newer version exists.

In all, 38 dependencies have been removed, although 22 have been added due to unbundling the OSGi dependencies. Only 11 dependencies retrain the same version as VIVO 1.9.

In the front end, jQuery has been updated to support Bootstrap. D3.js has been upgraded to v4. All jQuery plugins have been updated to work with the updated jQuery.

One javascript library used to format the index page has been removed due to being GPL licensed, and replaced with an MIT licensed equivalent.

Beyond simple theming changes (e.g. logos, colours, text), local customisations may need to be upgraded for the dependency changes.

Vocabulary Services

AGROVOC has been updated to use a new API.

UMLS has been transitioned to the NIH service. In order to use this, you need to obtain an application key from NIH (free registration).

Reasoning Improvements

If SameAs reasoning is enabled (by default, this is disabled), the reasoner will now generate the correct vitro:mostSpecificType.

List View Query Improvements

List view configuration files can now include <precise-subquery></precisde-subquery> elements. Due to the way SDB works, OPTIONAL clauses are slow because they are evaluated independently, and then joined with the rest of the query restrictions. <precise-subquery> allows you to duplicate the external restrictions that the clause will be joined with. This results in much more efficient SQL queries.

By including the restrictons inside a <precise-subquery> element, we can eliminate the CONSTRUCT that would otherwise be used, making it easier to maintain the SELECT query.

For triple stores other than SDB, the <precise-subquery> element is filtered out, as it is often unnecessary to include these extra restrictions, and in some cases may hurt performance. However, the filtering can always be enabled or disabled globally via the runtime.properties.

Performance Improvements

Search results render faster if they contain Person results. Indexing time has also been improved.

Updating data via the UI is faster.

Graph URIs are now cached for triple stores using a Jena implementation - this is a significant difference for TDB triple stores.

Full handling of TDB type conversions, preventing isomorphic test failures that result in reloading filegraph on restart.

Servlet 3.0

VIVO already required the use of a Servlet 3.0 compatible version of Tomcat. The web.xml has now been upgraded to take advantage of the servlet 3.0 spec, which allows developers to use annotations for servlet configuration. If you have a customisation that adds a new servlet, you can enable it without modifying the web.xml file.

Java 9

The Maven projects and code have had updates to work with the latest Java JDK 9. Note that Java 9 is a very recent release, and has not been extensively tested with VIVO. Also, Java 9 removes support for endorsed dirs, so you need to use a compatible version of Tomcat.

Additional Error Checking During Builds

The Maven projects now integrate Google's Error Prone - http://errorprone.info/ - tool into the compilation to detect serious errors in the Java code. Any customisations and contributions will now automatically be checked, preventing many serious errors from entering the code base.

Testing Framework

The Selenium IDE tests have been updated to use specific named selectors, rather than positions. Additional attributes (domain and range for faux properties) have been added to the UI to allow for this.

This allows the tests to be run against both the old (wilma) and new (tenderfoot) themes, and will make the tests more robust in the event of future ontology changes.

Note that Selenium IDE no longer works with the current versions of Firefox. Whilst we can currently run the test suite using a Java project and WebDriver, we will need to consider how these tests can be maintained in the future.

1.2.1.2 Contributors

Sabih Ali, Digital Science

Jim Blake, Cornell

Mike Conlon, University of Florida

Kitio Fofack, Université du Québec à Montréal

Benjamin Gross, UNAVCO / Clarivate Analytics

Huda Khan, Cornell

Ted Lawless, Clarivate Analytics / Brown University

Jacob Levernier, University of Pennsylvania

Jose Luis Martin, UC3M

Christian Hauschke, TIB Hannover

Steve McKay, Plum Analytics

Simon Porter, Digital Science

Graham Triggs, Duraspace / TIB Hannover

Tatiana Walther, TIB Hannover

Stefan Wolff, TU Dresden

Rebecca Younes, Cornell

1.2.1.3 Resolved Issues

Bug

- [VIVO-855[1]] - RDF export not working correctly
- [VIVO-1060[2]] - Add DumpRestoreController to Vitro web.xml
- [VIVO-1310[3]] - pom.xml implies that VIVO develop branch can still run under Java 1.7
- [VIVO-1311[4]] - Remove or replace the UMLS concept source
- [VIVO-1324[5]] - Security vulnerabilties in 1.9.x release
- [VIVO-1342[6]] - Ampersands (and possibly other characters?) not escaped in GraphML export of collaborator network graph
- [VIVO-1394[7]] - AboxRecomputer does not generate correct mostSpecificType for equivalent classes
- [VIVO-1404[8]] - Some SPARQL queries trigger Chrome XSS Auditor
- [VIVO-1435[9]] - Compliance with ORCID style guidelines
- [VIVO-1461[10]] - Assertions are ascribed to Vcard, but terms are not in the Vcard ontology

Story

- [VIVO-1451[11]] - Making Capability Map i18n Compliant

New Feature

- [VIVO-1252[12]] - Incorporate Cornell's DataDistributor API into core Vitro.
- [VIVO-1312[13]] - Implement Linked Data Fragments
- [VIVO-1335[14]] - Create Bootstrap Theme

1 https://jira.duraspace.org/browse/VIVO-855
2 https://jira.duraspace.org/browse/VIVO-1060
3 https://jira.duraspace.org/browse/VIVO-1310
4 https://jira.duraspace.org/browse/VIVO-1311
5 https://jira.duraspace.org/browse/VIVO-1324
6 https://jira.duraspace.org/browse/VIVO-1342
7 https://jira.duraspace.org/browse/VIVO-1394
8 https://jira.duraspace.org/browse/VIVO-1404
9 https://jira.duraspace.org/browse/VIVO-1435
10 https://jira.duraspace.org/browse/VIVO-1461
11 https://jira.duraspace.org/browse/VIVO-1451
12 https://jira.duraspace.org/browse/VIVO-1252
13 https://jira.duraspace.org/browse/VIVO-1312
14 https://jira.duraspace.org/browse/VIVO-1335

Task

- [VIVO-812[15]] - Automate the process of adjusting documentation to the release
- [VIVO-1316[16]] - Place external lookup base architecture in Vitro layer

Improvement

- [VIVO-1063[17]] - Update included jQuery library
- [VIVO-1246[18]] - Improve the ConfigurationBeanLoader
- [VIVO-1247[19]] - Remove duplicate code, based on improvements in ConfigurationBeanLoader
- [VIVO-1248[20]] - Add functionality to the edu.cornell.mannlib.vitro.webapp.utils.sparql package
- [VIVO-1260[21]] - Make http.createCacheHeaders true by default
- [VIVO-1270[22]] - Update Jena to latest release
- [VIVO-1272[23]] - Inject JSP content to Freemarker, instead of having secondary layout
- [VIVO-1273[24]] - Update DOI URL Schema to follow recomendations from Crossref
- [VIVO-1290[25]] - Improve Multi-Lingual Support
- [VIVO-1294[26]] - Language values (all.properties) should not be part of theme
- [VIVO-1307[27]] - Remove dom4j from the project
- [VIVO-1309[28]] - Update DWR to more recent version
- [VIVO-1317[29]] - Reduce the number of JSON libraries in the dependencies
- [VIVO-1318[30]] - In the ORCID client code, use Jackson library to handle JSON data
- [VIVO-1319[31]] - Remove dependency on sourceforge.net[32] JSON parser.
- [VIVO-1367[33]] - AGROVOC external service not working
- [VIVO-1375[34]] - Upgrade to Servlet 3.0 spec and annotations
- [VIVO-1376[35]] - Add smoke test to ensure that there are no XSD:Strings in SDB
- [VIVO-1381[36]] - Upgrade Solr to 4.10.4
- [VIVO-1382[37]] - Update JFact dependency
- [VIVO-1383[38]] - Update pooling libraries
- [VIVO-1384[39]] - Use commons-lang3 throughout

15 https://jira.duraspace.org/browse/VIVO-812
16 https://jira.duraspace.org/browse/VIVO-1316
17 https://jira.duraspace.org/browse/VIVO-1063
18 https://jira.duraspace.org/browse/VIVO-1246
19 https://jira.duraspace.org/browse/VIVO-1247
20 https://jira.duraspace.org/browse/VIVO-1248
21 https://jira.duraspace.org/browse/VIVO-1260
22 https://jira.duraspace.org/browse/VIVO-1270
23 https://jira.duraspace.org/browse/VIVO-1272
24 https://jira.duraspace.org/browse/VIVO-1273
25 https://jira.duraspace.org/browse/VIVO-1290
26 https://jira.duraspace.org/browse/VIVO-1294
27 https://jira.duraspace.org/browse/VIVO-1307
28 https://jira.duraspace.org/browse/VIVO-1309
29 https://jira.duraspace.org/browse/VIVO-1317
30 https://jira.duraspace.org/browse/VIVO-1318
31 https://jira.duraspace.org/browse/VIVO-1319
32 http://sourceforge.net
33 https://jira.duraspace.org/browse/VIVO-1367
34 https://jira.duraspace.org/browse/VIVO-1375
35 https://jira.duraspace.org/browse/VIVO-1376
36 https://jira.duraspace.org/browse/VIVO-1381
37 https://jira.duraspace.org/browse/VIVO-1382
38 https://jira.duraspace.org/browse/VIVO-1383
39 https://jira.duraspace.org/browse/VIVO-1384

- [VIVO-1385[40]] - Replace unmaintained CSV parser with commons-csv
- [VIVO-1386[41]] - Make consistent use of HttpClient 4.5, remove conflicting dependencies
- [VIVO-1387[42]] - Update all dependencies with known vulnerabilities to latest versions
- [VIVO-1393[43]] - Replace isotope jQuery plugin
- [VIVO-1397[44]] - Improve performance and reliability of search indexing
- [VIVO-1400[45]] - Have optional "precise subquery" elements in list views for triple stores that perform better with more selective queries
- [VIVO-1403[46]] - Improve update performance
- [VIVO-1405[47]] - Defeat browser cacheing for new versions of JavaScript and CSS files.
- [VIVO-1406[48]] - Visualisations in a multi-lingual release
- [VIVO-1410[49]] - Release the next version of VIVO and Vitro
- [VIVO-1438[50]] - Some text on Forms are made of different strings associated following english syntax
- [VIVO-1447[51]] - Organize the ontology files. Produce a vivo.owl
- [VIVO-1448[52]] - Move password encryption from MD5 to a salted hash
- [VIVO-1458[53]] - Update to Jena 3.6
- [VIVO-1463[54]] - Update ontologies.owl
- [VIVO-1464[55]] - Identify and separate candidate ontologies for removal
- [VIVO-1470[56]] - Improve cross-platform support in build

Documentation

- [VIVO-31[57]] - Improve documentation of Google Analytics
- [VIVO-34[58]] - Make it easy to do "next", "previous" and "up" links in a Confluence page
- [VIVO-242[59]] - Establish Wiki versioning process
- [VIVO-813[60]] - Improve document details
- [VIVO-917[61]] - Create a confluence macro to show sections numbers for multi-page documents
- [VIVO-1274[62]] - Improve documentation process for next release
- [VIVO-1334[63]] - Create a recommendation for the use of string and langString
- [VIVO-1351[64]] - Write "TPF Endpoint" for Tech doc

40 https://jira.duraspace.org/browse/VIVO-1385
41 https://jira.duraspace.org/browse/VIVO-1386
42 https://jira.duraspace.org/browse/VIVO-1387
43 https://jira.duraspace.org/browse/VIVO-1393
44 https://jira.duraspace.org/browse/VIVO-1397
45 https://jira.duraspace.org/browse/VIVO-1400
46 https://jira.duraspace.org/browse/VIVO-1403
47 https://jira.duraspace.org/browse/VIVO-1405
48 https://jira.duraspace.org/browse/VIVO-1406
49 https://jira.duraspace.org/browse/VIVO-1410
50 https://jira.duraspace.org/browse/VIVO-1438
51 https://jira.duraspace.org/browse/VIVO-1447
52 https://jira.duraspace.org/browse/VIVO-1448
53 https://jira.duraspace.org/browse/VIVO-1458
54 https://jira.duraspace.org/browse/VIVO-1463
55 https://jira.duraspace.org/browse/VIVO-1464
56 https://jira.duraspace.org/browse/VIVO-1470
57 https://jira.duraspace.org/browse/VIVO-31
58 https://jira.duraspace.org/browse/VIVO-34
59 https://jira.duraspace.org/browse/VIVO-242
60 https://jira.duraspace.org/browse/VIVO-813
61 https://jira.duraspace.org/browse/VIVO-917
62 https://jira.duraspace.org/browse/VIVO-1274
63 https://jira.duraspace.org/browse/VIVO-1334
64 https://jira.duraspace.org/browse/VIVO-1351

1.3 Functional Overview

1.3.1 Online Access

VIVO provides an online portal to showcase the academics, their work, and their professional relationships.

1.3.1.1 Linked Open Data

All information within a VIVO system is represented natively in the RDF data model - everything is expressed as subject - predicate - object statements. These statements are written to a triple store, and are made available as RDF documents for each resource, in a number of serialisation formats.

1.3.1.2 Built-in Search

All content is indexed using Solr - a popular open source search platform built on Lucence.

1.3.1.3 Navigation

VIVO provides simple navigation through menus which lead to lists of various types of entities – people, organizations, research. An Index provides access to lists of all types of entities. The Capability Map provides a graphical method for finding people by concepts.

1.3.1.4 Optimisations for Google Indexing

VIVO embeds structured data - hcards and schema.org - into profile pages to better support Google indexing. It also creates a sitemap.xml for all of the profiles in the system, and includes a link to this sitemap in the robots.txt.

It is encouraged that you should register your VIVO instance in Google Webmaster Tools and submit the sitemap.xml for better visibility of how Google is indexing your content.

1.3.1.5 Support for Modern Browsers

VIVO creates standard HTML and CSS that can be used in all modern browsers. Visualisations are mostly built using D3 - a standard JavaScript library - which allows them to be viewed even on mobile devices.

1.3.2 Getting Data into VIVO

1.3.2.1 Manual Data Entry

All screens in VIVO can provide for data entry, for users logged in with sufficient access. There are numerous roles that VIVO provides - from administrators that can edit any of the data in the system, to self editor privileges for users so that they can edit their own profile and related information.

1.3.2.2 Automated Data Entry

It is possible to add data to VIVO using automated tools. VIVO provides a SPARQL update endpoint, which can be used by external tools to manipulate the data, or the VIVO Harvester provides a means to acquire and transform data, and load it directly into the triple store.

1.3.3 Access Control

VIVO has internal storage for user accounts, and can authenticate based on a password (stored as a hash), or via an external authentication mechanism, such as Shibboleth. For externally authenticated users, an internal user account is still required, and is matched based on the external ID.

1.4 System Requirements

1.4.1

1.4.2 Hardware Recommendations

You can install and run VIVO on most modern PC, laptop, or server hardware. Whilst the application layer needs a reasonable amount of memory, the majority of the workload is placed on the storage layers, which as a semantic web application means the triple store. As VIVO aims to be agnostic to the triple store, the precise requirements will depend on your choice of triple store. However, the default configuration is to use Jena SDB backed by MySQL - in this setup, it is recommended that you have very high IO bandwidth for the file system used by MySQL, and significant memory for caching layers of the database engine.

1.4.2.1 Minimum Specification

2 core x64 processor, 2GB RAM, 100GB HDD

1.4.2.2 Recommended Specification

4 core x64 processor, 16GB RAM, 500GB SSD

Note: I/O performance for MySQL is critical to the responsiveness of the application. The fastest SSD you can specify will help, as will having direct (e.g. not virtualised) access to it.

1.4.3 Software Requirements

1.4.3.1 Operating System

VIVO is largely agnostic to the OS that it is running on - as a Java application, it is dependent on having a Java Virtual Machine and a Tomcat servlet container. It should be possible to install and run VIVO on any OS where you are able to provide all of the other software requirements.

However, most sites will run their installations on a Linux server, and you may find that it is easier to follow the installation instructions on a Linux / UNIX variant. Notably, if you are running Windows, you may need to stop running processes (e.g. Tomcat) in order to complete some of the instructions, due to file locking semantics on Windows.

1.4.3.2 Java 8

The minimum requirement is Java 8. Both OpenJDK and Oracle JVMs are compatible. Other JVMs that meet the JDK 8 specification may work, but have not been tested.

Note that you need to have the full Java Development Kit installed in order for Tomcat to operate correctly - the runtime alone is not sufficient.

Warning

Java 9 has not been tested at time of writing.

1.4.3.3 Maven 3.0.3 or later

The installation mechanism uses Maven to package and deploy the VIVO application and other necessary files. Additionally, the development environment also uses Maven to compile the and package the code.

The minimum version of Maven required is 3.0.3, although it is better to use a more recent version of the 3.x releases where possible.

Maven can be downloaded from the following location: http://maven.apache.org/download.html, although you may use a version supplied by your operating system / package manager, providing it meets the minimum requirements.

Configuring a Proxy

You can configure a proxy to use for some or all of your HTTP requests in Maven. The username and password are only required if your proxy requires basic authentication (note that later releases may support storing your passwords in a secured keystore, in the mean time, please ensure your *settings.xml* file (usually *${user.home}/.m2/settings.xml*) is secured with permissions appropriate for your operating system).

Example:

```
<settings>
  .
  .
  <proxies>
   <proxy>
      <active>true</active>
      <protocol>http</protocol>
      <host>proxy.somewhere.com</host>
      <port>8080</port>
      <username>proxyuser</username>
      <password>somepassword</password>
      <nonProxyHosts>www.google.com|*.somewhere.com</nonProxyHosts>
    </proxy>
  </proxies>
  .
  .
</settings>
```

1.4.3.4 MySQL / MariaDB 5.5 or later (or any other supported by Jena SDB)

Jena SDB requires an SQL database to operate. By default, VIVO relies on MySQL - or the open source fork MariaDB, which is provided by most Linux distributions in place of MySQL.

Once installed, you only need to create a user and schema - see Installing VIVO (see page 32). VIVO will create the necessary tables and load the default data on startup.

Alternative databases: Jena SDB supports other databases - including PostgreSQL and Oracle. If you wish to use a different database, you will need to add the appropriate Java libraries to the application, and configure the VitroConnection.DataSource.* settings in runtime.properties so that Jena knows what database it is operating with.

1.4.3.5 Tomcat 7 or later

VIVO is a web application, which requires a servlet engine to host. It has been tested with Tomcat 7 and Tomcat 8. The applications make use of Tomcat context.xml configuration files - if you wish to use an alternative servlet engine, you will need to make the appropriate adjustments.

You may use Tomcat as supplied by your operating system / package manager providing is meets the minimum requirements, or you can download it from: http://tomcat.apache.org/download-80.cgi

Tomcat User: When running, Tomcat is usually launched under an unprivileged user account. As VIVO needs to be able to read and write to the home directory, you must ensure that permissions are set on the home directory correctly. This is most easily achieved by assigning ownership to the user that Tomcat is running as.

2 Installing VIVO

2.1 Installing from Distribution

2.1.1 Overview

Download the 1.9.x distribution release from the VIVO repository on [65]GitHub. The standard distribution consists of the projects required to create a home directory for VIVO, and to copy the web application and search index. All the compiled code and dependencies are resolved from the Maven central repository at the time you run Maven.

The standard distribution is laid out as follows:

```
vivo-2.0.0/
  pom.xml
  example-settings.xml
  home/
    pom.xml
    src
  solr/
    pom.xml
    src
  webapp/
    pom.xml
    src
```

2.1.2 Preparing the Installation Settings

In order to fully install VIVO, you need to create a settings file that provides some essential information:

app-name

vivo-dir

65 https://github.com/vivo-project/VIVO/releases/tag/rel-1.9.2

tomcat-dir

This file needs to be created following the Maven Settings Reference[66]. A template file already exists within the VIVO standard distribution, called "example-settings.xml". You may copy this file (it can be called anything you like), and edit the contents to fit your requirements / system configuration.

2.1.3 Installing VIVO

Once you have an appropriate settings file (these instructions will assume that you are using example-settings.xml - replace this with your actual file), you simply need to run Maven, specifying the install goal and your settings file.

```
$ cd VIVO
VIVO$ mvn install -s example-settings.xml
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] Vitro
[INFO] Vitro Dependencies
[INFO] Vitro API
[INFO] VIVO
[INFO] VIVO API
[INFO] Vitro Web App
[INFO] VIVO Web App
[INFO] Vitro Home
[INFO] VIVO Home
[INFO] Vitro Solr App
[INFO] VIVO Installer
[INFO] VIVO Prepare Home
[INFO] VIVO Prepare Solr App
[INFO] VIVO Prepare Web App
[INFO]
....
```

The VIVO home directory will now be created and the VIVO application installed to Tomcat.

In order to run VIVO, please read the section below "Completing the Installation (see page 36)".

2.2 Installing from GitHub

2.2.1 Preparing the Repositories

In order to install the development code from GitHub, you need to clone both the Vitro and VIVO repositories from the vivo-project organization. These clones should be in sibling directories called "Vitro" and "VIVO" respectively:

66 https://maven.apache.org/settings.html

```
$ git clone https://github.com/vivo-project/Vitro.git Vitro -b maint-rel-1.9
$ git clone https://github.com/vivo-project/VIVO.git VIVO -b maint-rel-1.9
$ ls -l
drwxr-xr-x  user  group  1 Dec 12:00  Vitro
drwxr-xr-x  user  group  1 Dec 12:00  VIVO
```

⚠ If you do not place the Vitro code in a sibling directory called "Vitro", then you will have to supply the "vitro-core" property to Maven - e.g. mvn package -Dvitro-core=~/Vitro

It is expected that the Maven project numbers are kept in sync between the Vitro / VIVO projects, however, depending on when you update / sync your repositories, you may need to adjust the project version numbers for the build to work.

2.2.2 Preparing the Installation Settings

In order to fully install VIVO, you need to create a settings file that provides some essential information:

app-name

vivo-dir

tomcat-dir

This file needs to be created following the Maven Settings Reference[67]. A template file already exists in the "installer" directory within the VIVO project, called "example-settings.xml". You may copy this file (it can be called anything you like), and edit the contents to fit your requirements / system configuration.

2.2.3 Installing VIVO

2.2.3.1 Default Installer

Once you have an appropriate settings file (these instructions will assume that you are using installer/example-settings.xml - replace this with your actual file), you simply need to run Maven, specifying the install goal and your settings file.

```
$ cd VIVO
VIVO$ mvn install -s installer/example-settings.xml
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] Vitro
[INFO] Vitro Dependencies
[INFO] Vitro API
[INFO] VIVO
[INFO] VIVO API
```

67 https://maven.apache.org/settings.html

```
[INFO] Vitro Web App
[INFO] VIVO Web App
[INFO] Vitro Home
[INFO] VIVO Home
[INFO] Vitro Solr App
[INFO] VIVO Installer
[INFO] VIVO Prepare Home
[INFO] VIVO Prepare Solr App
[INFO] VIVO Prepare Web App
[INFO]
....
```

The VIVO home directory will now be created and the VIVO application installed to Tomcat.

In order to run VIVO, please read the section below "Completing the Installation (see page 36)".

Custom Installer

If you want to use the source code / GitHub clone with your own customizations, you can exclude the supplied installer project, and use your own customized installer project instead. To do so, you need to supply the location of your custom installer project as the "vivo-installer-dir" property. This can be done on the command line or in the settings.xml. If you are supplying a relative path, it should be relative to the location of the VIVO/pom.xml.

```
$ cd VIVO
VIVO$ mvn install -s installer/example-settings.xml -Dvivo-installer-dir=../myedu-vivo
[INFO] Scanning for projects...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Build Order:
[INFO]
[INFO] Vitro
[INFO] Vitro Dependencies
[INFO] Vitro API
[INFO] VIVO
[INFO] VIVO API
[INFO] Vitro Web App
[INFO] VIVO Web App
[INFO] Vitro Home
[INFO] VIVO Home
[INFO] Vitro Solr App
[INFO] Custom VIVO Installer
[INFO] Custom VIVO Prepare Home
[INFO] Custom VIVO Prepare Solr App
[INFO] Custom VIVO Prepare Web App
[INFO]
....
```

The VIVO home directory will now be created and the VIVO application installed to Tomcat, including any customizations that are defined in your local installer project.

2.3 Completing the Installation

2.3.1 Configure the Database Schema

The default configuration of VIVO is to use MySQL as a backing store for Jena SDB. Whilst VIVO / Jena will create the necessary tables for the triple store, a database (schema) and authentication details need to have been created first. To do so, log in to MySQL as a superuser (e.g. root)

```
$ mysql -u root -p
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.9 MySQL Community Server (GPL)
Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE vitrodb CHARACTER SET utf8;
mysql> GRANT ALL ON vitrodb.* TO 'vitrodbUsername'@'localhost' IDENTIFIED BY 'vitrodbPassword';
```

2.3.2 Configure the Home Directory

There are two configuration files that are required to be in the home directory. By default, the installer does not create them so that they are not overwritten when you redeploy the application. Instead, example files are created in the home directory, which can be copied and used as the basis for your installation.

```
$ cd /usr/local/vivo/home
/usr/local/vivo/home$ cp config/example.runtime.properties runtime.properties
/usr/local/vivo/home$ cd config
/usr/local/vivo/home/config$ cp example.applicationSetup.n3 applicationSetup.n3
```

Minimum Configuration Required

In order for your installation to work, you will need to edit runtime.properties and ensure that the VitroConnection properties are correct for your database engine. They should look something like this.

```
VitroConnection.DataSource.url = jdbc:mysql://localhost/vitrodb
VitroConnection.DataSource.username = vitrodbUsername
VitroConnection.DataSource.password = vitrodbPassword
```

2.3.3 Configure and Start Tomcat

2.3.3.1 Set JVM parameters

VIVO copies small sections of your RDF database into memory in order to serve Web requests quickly (the in-memory copy and the underlying database are kept in synch as edits are performed).

VIVO may require more memory than allocated to Tomcat by default. With most installations of Tomcat, the setenv.sh or setenv.bat file in Tomcat's bin directory is a convenient place to set the memory parameters. *If this file does not exist in Tomcat's bin directory, you can create it.*

For example:

```
export CATALINA_OPTS="-Xms512m -Xmx512m -XX:MaxPermSize=128m"
```

This tells Tomcat to allocate an initial heap of 512 megabytes, a maximum heap of 512 megabytes, and a PermGen space of 128 megs. Larger values may be required, especially for production installations in large enterprises. In general, VIVO runs more quickly if given more memory.

If an OutOfMemoryError occurs during VIVO execution, increase the heap parameters and restart Tomcat.

2.3.3.2 **Set security limits**

VIVO is a multithreaded web application that may require more threads than are permitted under your operating system's installation's default configuration. Ensure that your installation can support the required number of threads for your application. For a Linux production environment you may wish to make the following edits to /etc/security/limits.conf, replacing apache and tomcat with the appropriate user or group name for your setup:

```
apache hard nproc 400

tomcat hard nproc 1500
```

2.3.3.3 **Set URI encoding**

In order for VIVO to correctly handle international characters, you must configure Tomcat to conform to the URI standard by accepting percent-encoded UTF-8.

Edit Tomcat's conf/server.xml and add the following attribute to each of the Connector elements: URIEncoding="UTF-8".

```
<Server ...>
  <Service ...>
    <Connector ... URIEncoding="UTF-8"/>
      ...
    </Connector>
  </Service>
</Server>
```

Some versions of Tomcat already include this attribute as the default.

2.3.3.4 **Take care when creating Context elements**

Each of the webapps in the VIVO distribution (VIVO and Solr) includes a "context fragment" file, containing some of the deployment information for that webapp.

Tomcat allows you to override these context fragments by adding Context elements to server.xml. If you decide to do this, be sure that your new Context element includes the necessary deployment parameters from the overridden context fragment.

2.3.3.5 Starting Tomcat

If everything has been completed successfully, then you should simply be able to start Tomcat at this point, and VIVO will be available. If you are using a Tomcat supplied by your operating system / package manager, then use your normal means for starting the application server.

Otherwise, start Tomcat by running the startup script - e.g.

```
$ /usr/local/tomcat/bin/startup.sh
```

2.4 Verify Your Installation

If you have completed the previous steps, you have good indications that the installation was successful.

- When you Start tomcat, you see that Tomcat recognizes the webapp, and that the webapp is able to present the initial page.
- The startup status will indicate if the basic configuration of the system was successful. If there were any serious errors, you will see the status screen and will not be allowed to continue with VIVO. If there are warnings, you will see the status screen when you first access VIVO, but after that you may use VIVO without hinderance. In this case, you can review the startup status from **siteAdmin -> Startup status**.
- Log in as root. Your root username is vivo_root@yourdomainname . The first time root password is rootPassword. You will be asked to change it.

Here is a simple test to see whether the ontology files were loaded:

- Click on the "Index" link on the upper right, below the logo. You should see a "locations" section, with links for "Country" and "Geographic Location." The index is built in a background thread, so on your first login

you may see an empty index instead. Refresh the page periodically to see whether the index will be populated. This may take some time: with VIVO installed on a modest laptop computer, loading the ontology files and building the index took more than 5 minutes from the time that Tomcat was started.
- Click on the "Country" link. You should see an alphabetical list of the countries of the world.

Here is a test to see whether your system is configured to serve linked data:

- Point your browser to the home page of your website and click the "Log in" link near the upper right corner. Log in with the rootUser.emailAddress you set in runtime.properties. If this is your first time logging in, you will be prompted to change the password.
- After you have successfully logged in, click "site admin" in the upper right corner. In the drop down under "Data Input" select "Faculty Member(core)" and click the "Add individual of this class" button.
- Enter the name "test individual" under the field "Individual Name," scroll to the bottom, and click "Create New Record." You will be taken to the "Individual Control Panel." Make note of the value of the field "URI" - it will be used in the next step.
- Open a new web browser or browser tab to the page http://lodview.it/. Enter the URI of the individual you created in the previous step and click "go."
- In the resulting page search for the URI of the "test individual." The page should display information for the individual, including an rdfs:label and rdf:type. This indicates that you are successfully serving linked RDF data. If the service returns an error you are not successfully serving linked data.

Finally, test the search index.

- Type the word "Australia" into the search box, and click on the Search button.You should see a page of results, with links to countries that border Australia, individuals that include Australia, and to Australia itself. To trigger a rebuild of the search index, you can log in as a site administrator and go to **Site Admin -> Rebuild search index**.

3 Upgrading VIVO

3.1 Upgrading from 1.9.x to 1.10.x

Due to the dependency updates in VIVO 1.10.x, it is important that you read the instructions here carefully, and plan your upgrade accordingly. You will need to consider:

1) The time it will take to upgrade the triple store (can not be performed against a running system)

2) SPARQL queries need to be checked for any explicit use of string datatypes

3) Any applications directly accessing the SDB triple store need to be upgraded to use Jena 3 libraries.

- This includes VIVO Harvester - if you use this, you must upgrade your Harvester to 2.x at the same time as VIVO. You must not use Harvester 1.x with VIVO 2.x, or Harvester 2.x with VIVO 1.x.

If you are planning an upgrade from anything prior to 1.9.x, please ensure that you read upgrade instructions relating to previous releases. If your version of VIVO is prior to 1.8.x, you will have some ontology changes to consider.

3.1.1 Java 8

With the upgrade to Jena 3.x, it is required that Java 8 is used. The Maven projects have been upgraded to state a dependency on version 8, and Maven will not run without it.

If you have previously been using Java 7, you will need to install Java 8. As usual, it needs to be a full JDK to support Tomcat.

Java 9 has now been released, but has not been tested with VIVO. However, it does build and appear to run on Java 9. Note that to do so, you need to have a compatible version of Tomcat, as endorsed directory support has been removed.

3.1.2 Jena 3.x

As the core framework used throughout VIVO, as well as the implementation for the default triple stores, this is the most significant part of the upgrade (although, if you have extensive Javascript customisations in your front end, that side may take more time).

3.1.2.1 Upgrading The Triple Store

This needs to be done by everybody upgrading from a previous version, using the (default) Jena triple stores. It is necessary due to the changes in handling untyped literals in RDF 1.1. If you fail to perform the upgrade, or you mix Jena libraries

If you are using an alternative triple store implementation (e.g. Virtuoso, Stardog), then you do not need to reload that triple store. But please remember that VIVO uses two triple stores - content and configuration - and if you are only using an alternative content triple store, you will still need to upgrade the configuration store.

Upgrading the triple store(s) (there are two - content and configuration) involves dumping the contents of your stores, and then reloading them. Whilst it is possible to do this using command line tools that are part of the Jena projects, VIVO has created and distributes tools that work with your VIVO configurations to make the process easier.

In order to upgrade your triple store, use the following steps (replace <your-settings.xml> and <vivo_home> with the appropriate values for your system:

1. Stop Tomcat - it is vital that Tomcat / VIVO, and any other applications that may access the triple stores, are not running during this process.

2. Run "mvn clean install -s <your-settings.xml>" in your VIVO 1.10.0 development area to update your web application and home directory
   This will install the tools into your <vivo_home>/bin directory. Alternatively, you can download jena2tools.jar[68] and jena3tools.jar[69] from this page.

3. Export your existing triple stores:

68 https://wiki.duraspace.org/download/attachments/96995727/jena2tools.jar?api=v2&modificationDate=1522787186384&version=1
69 https://wiki.duraspace.org/download/attachments/96995727/jena3tools.jar?api=v2&modificationDate=1522787186255&version=1

- ***To export successfully, you need to ensure no other programs are accessing your triple stores.***
- ***Your file system must have the space available and be capable of storing files large enough to contain your entire triple store serialisation.***

To export your triples store, use the jena2tools utility provided with VIVO 1.10.0, in <vivo home dir>/bin, specifying the export command, as shown below.

```
java -jar jena2tools.jar -d <vivo home dir> -e
```

Arguments:
    -d - the location of the Vitro/VIVO home directory
    -e - run in export mode
On execution, the program will read your configuration files, find your Vitro or VIVO configuration within the vivo/vitro home directory, and get the necessary information to connect to your configuration triple store (usually <vivo home dir>/tdbModels), and your content triple store (usually in SDB). If your triple store(s) are not SDB or TDB backed, then it will simply skip them.
jena2tools will then extract the contents of the available triple stores, and dump them to <vivo home dir>/dumps in TriG format.

4. Check that the export has completed - you should have a <vivo_home>/dumps directory, that contains the files "configuration.trig" and "content.trig".

5. Empty your triple stores ready for reloading:

   Content Triple Store:
   Drop all tables in your SDB database as named in your runtime.properties. You may drop your database and recreate it as empty, just as you would for creating a new VIVO install. jena3tools must find an empty database (no tables) as named in your runtime.properties and will recreate your SDB triple store as tables in the named database using the triples produced by jena2tools and stored in <vivo home dir>/dumps/content.trig

   In MySQL, DROP TABLE Nodes; DROP TABLE Quads; DROP TABLE Triples; DROP TABLE Prefixes;
   - OR - DROP DATABASE <vivodb>; CREATE DATABASE <vivodb> CHARACTER SET utf8mb4; GRANT ALL ON <vivodb>.* TO '<user>'@'localhost';
   - OR - if using a TDB content triple store: rm -rf <vivo_home>/tdbContentModels

   Configuration Triple Store:
   Delete all files in <vivo home dir>/tdbModels. Jena3tools will rebuild your configuration tdbModels based on the content created by jena2tools and stored in <vivo home dir>/dumps/configuration.trig
   rm -rf <vivo_home>/tdbModels

6. Reload the data
   :
   Having exported your Jena 2 triple stores, you can reload them using jena3tools, also available with VIVO 1.10.0, specifying the import command.

```
java -jar jena3tools.jar -d <vivo home dir> -i
```

Arguments:
    -d - the location of the Vitro/VIVO home directory

-i - run in import mode

On execution, the program will find your Vitro or VIVO configuration within the home directory, as well as the dumps that you have created with jena2tools. It will import them into the SDB and TDB triple stores, based on the configuration of your Vitro/VIVO instance.
jena3tools will be present in <vivo home dir>/bin when you install the 1.10.0 beta. Alternatively, it can be downloaded from GitHub[70].

- Note that this can take a while. A rough guide is to expect about 600 triples per second to reload. (Roughly 1 hour per 2 million triples).

7. Restart Tomcat

   - Note that there are a couple of spelling mistakes that have been corrected in the filegraphs. As a result, you should expect that VIVO will reinference and re-index the data when you start it up for the first time.

3.1.2.2 Upgrading Local Java Code using Jena

If you have local customisations or additional applications that make use of the Jena libraries, you will need to upgrade these to work with Jena 3. Mostly this is simply a case of renaming any packages in imports for Jena classes:

import com.hp.hpl.jena.*

becomes

import org.apache.jena.*

However, some classes have been moved, or removed, and some interfaces have additional methods. So in rare cases you may find that you need to make a few small changes beyond this.

3.1.2.3 A Note on Other Dependency Changes

To remove the possibility of incompatible classes being loaded, and to remove known vulnerabilities from the code base, most of the Java dependencies in VIVO have been removed, updated or replaced.

For the most part, this will have minimal impact on local customisations.

Some libraries - such as commons-lang3 - have new package names, but mostly compatible classes, and usually just require the imports to be adjusted.

The CSV libary was outdated and unmaintainted, and has been replaced with commons-csv.

There were originally 5 JSON libaries (Jackson, Gson, Glassfish, Sourceforge.net and Json.org parsers). All code is now using Jackson, and the other parsers have been removed.

38 dependencies have been removed from the standard distribution. If you have any customisations that make use of them, that should work, but you will need to add the dependencies to your own projects.

22 dependencies will appear to have been added, but these are mostly from unbundling the owlapi OSGi dependency, and these - along with other conflicting classes - had been packaged as part of the bundle.

Whilst every effort has been made to eliminate known vulnerabilities, the Maven dependency-check plugin still reports 13 vulnerabilities across 8 libraries - for which these are currently the latest releases.

70 https://github.com/vivo-project/jenatools

In total, 11 out of an original 113 dependencies are still at the same version as the previous release.

3.1.3 UI Changes

3.1.3.1 jQuery 1.12.4

Bootstrap based themes require a newer version of jQuery than was shipped with VIVO. In order for all of the functionality aross the UI to work, and to minimise the amount of duplication in the UI, it was necessary to upgrade all of the pages and plugins that used jQuery, so that the same upgraded version works across all of the pages and themes.

This release does require some migration of javascript that makes use of it. There is a script - jquery-migrate.js that helps with this, providing extra backwards compatibility, and logging warnings to the console whenever an old method is used.

All of the existing code that was relying on the migrate script has been updated, so that it is no longer needed.

To aid transition, VIVO still ships with the jquery-migrate.js script, allowing most existing code to work. You should monitor the Javascript console, and apply updates if you see any logged messages in your customisations - future versions of VIVO will remove this migration script in order to upgrade to later versions.

3.1.3.2 jQuery plugins

In order to support the upgrade to jQuery 1.12.4, and remove any backward compatibility messages from the Javascript console, the following plugins have been upgraded:

jQuery UI

jCrop

qTip

DataTable

In addition, the following plugins have been replaced with equivalents for compatibility and/or licensing issues:

Old	New	Notes
mb.FlipText	Jangle	Used for rotating text on the graphs (e.g. temporal graph)
isotope	wookmark	Used to render the three columns on the index page

3.1.3.3 D3 v4

This is another major upgrade that has architectural changes to improve modularity and make writing visualizations easier, as well as a selection of new features.

Where D3 was being included by VIVO (e.g. on the profile page, in co-authorship networks, etc.), these now include D3 v4, and the visualizations have been upgraded to use D3 v4.

The only visualization that has NOT been upgraded to D3 v4, is the Capability Map - as a full page visualization, that page is still including it's own D3 v3.

When upgraading, you have a few choices:

1) If you have a full page custom visualization, you can continue to specify whatever libraries and versions that you want to use, although if you are referencing the "shared" D3, you will need to adjust this to include your own D3 that is the version you require.

2) If you are embedding the visualization on a page that may have other visualizations, you should either:

a) Upgrade your custom visualization to use D3 v4.

b) Render your visualization into an iframe, so that you can specify your own javascript dependencies.

By making this change now, we are getting on to a maintained version of D3 (v3 has not had any releases for 18 months), and it prepares us for using D3.express[71] in the future, for more dynamic visualization building.

3.1.4 ORCiD API

It has been announced that ORCiD are looking to shut down their v1.x endpoints by the end of 2017. During development to support the new v2 api, it was recognised that the way the settings are currently configured should be simplified. As a result, in order for the ORCiD integration to work in VIVO 1.10, you will need to update your runtime.properties - regardless of which version of the api that you want to use.

The new settings look like this:

```
orcid.clientId = 0000-0000-0000-000X
orcid.clientPassword = 00000000-0000-0000-0000-000000000000
orcid.webappBaseUrl = http://localhost:8080/vivo
orcid.externalIdCommonName = VIVO Cornell Identifier
orcid.apiVersion = 2.0
orcid.api = sandbox
```

Setting	Values	Notes
orcid.clientId	<your client id>	The client ID that you obtained from ORCiD for API access
orcid.clientPassword	<your password>	The password the you obtained from ORCiD for API access
orcid.webappBaseUrl	<your VIVO website>	The URL for your VIVO site
orcid.externalIdCommonName	<site description>	The text that you want to appear in ORCiD for links back to VIVO profiles
orcid.apiVersion	2.0 (or 1.2)	The version of the API to use. It is recommended that you use 2.0.
orcid.api	release or sandbox	Use sandbox for development, release for production.

71 https://medium.com/@mbostock/a-better-way-to-code-2b1d2876a3a0

3.1.5 List View Configurations

To produce complex views of data associated with properties (e.g. a person's publication list), VIVO allows the association of externalized SPARQL queries, and associated Freemarker template links.

The configuration for this are the files called "listViewConfig-*" in the <webapp>/configs/ directory.

Due to the performance of OPTIONAL clauses for some triple stores, in VIVO 1.x the configuration files usually consist of a SELECT query (to retrieve the data for the associated Freemarker template), and one or more CONSTRUCTs (using UNIONs) to create a smaller model that the SELECT query would then be performed against.

This creates additional buffering of an intermediate model, the overhead of performing multiple queries, and makes the configurations harder to write and maintain.

For triple stores with poor OPTIONAL performance, this is a result of the OPTIONAL clause returning large amounts of data that is then joined to the rest of the query.

This can be avoided by making the OPTIONAL clauses contain the restrictions that they will be joined to, in addition to them appearing outside for the join.

As this is not necessary for all triple stores, an element <precise-subquery> has been introduced, so that these additional statements can be filtered out for triple stores where they aren't required.

So,

```
SELECT ?menuItem
       ?linkText
 HERE {
    ?subject ?property ?menuItem .
    OPTIONAL {
        ?menuItem display:linkText ?linkText .
    }
}
```

can be rewritten as

```
SELECT ?menuItem
       ?linkText
 HERE {
    ?subject ?property ?menuItem .
    OPTIONAL {
        <precise-subquery>?subject ?property ?menuItem .</precise-subquery>
        ?menuItem display:linkText ?linkText .
    }
}
```

```
When all OPTIONAL clauses are rewritten like this, the <query-construct>
elements can be removed, for approximately 10% performance improvement, and a
larger reduction in Java processing overhead. This allows the web server to
scale to handle more requests.
```

If you have customized listViewConfig-*.xml files, you do not need to rewrite them for VIVO 1.10 - they will work unmodified. However, you will have a small performance improvement, and more readable, maintainable queries, if you choose to modify them to take advantage of the new features.

3.1.6 Vocabulary Services

<< Upgraded web services >>

<< Registering for a UMLS key >>

3.1.7 Servlet 3.0 Upgrade

The web.xml that ships with VIVO has been updated to use 3.0 semantics (the required version of Tomcat already supported 3.0).

If you have customisations that introduce new servlets, then you can still add the configuration to web.xml, or you can modify the servlet to use @WebServlet annotations.

If you are replacing a servlet, then you will need to map the existing servlet to an unused url, and map the new servlet by adding configuration for these servlets to web.xml.

Alternatively, if you want to disable a servlet, then you can set the web.xml back to 2.5 spec, and add all of the servlet configuration explicitly to the web.xml.

3.1.8 Java Dependencies

3.1.8.1 HttpClient

Due to OSGi bundling of some of the core dependencies, VIVO had been shipping three different versions of HttpClient, all of which were incompatible with each other, and sometimes generated errors depending on which code paths got executed first.

VIVO 1.10 brings convergence around HttpClient 4.5.3, removing the problems caused previously. If you have any customisations that depend on HttpClient (or the fluent api - fluent-hc), please ensure that you are using the versions that are already indlueded with VIVO. This may require some minor adjustments to your client code to make it compatible.

3.1.8.2 OSGi Dependencies

The OWLAPI previously included with VIVO was an OSGi bundle, and included the classes of a number of libraries (such as HttpClient above). This causes classloading conflicts. In VIVO 1.10, we are depending directly on the libraries that were being bundled, rather than the bundle in it's entirety.

In addition, Jena has OSGi dependencies for HttpClient, leading to more duplicate classes. These have been explicitly excluded - see dependencies/pom.xml[72] for how this is done.

72 https://github.com/vivo-project/Vitro/blob/develop/dependencies/pom.xml

3.1.8.3 JSON Parsers

Four JSON parsers - GSON (com.google.gson), Glassfish (javax.json.Json, Sourceforge.net (net.sf.json) and JSON.org (org.json) have been removed, to simplify the code base, reduce vulnerabilities and improve performance.

All JSON parsing in VIVO is now handled through Jackson.

If you have local customisations that use a different JSON parser, you can add the dependency to your projects, although it is recommended that migrating to Jackson is preferable for long term maintenance.

3.1.8.4 Replaced Dependencies

The CSV parser has been replaced with commons-csv.

3.1.8.5 Removed Dependencies

The following dependencies have been removed from VIVO. If you have customisations that require any of them, you will need to add them as dependencies in your own projects.

agrovocws-3.0.jar, asm-3.1.jar, aterm-java-1.8.2.jar, axis-1.3.jar, axis-jaxrpc-1.3.jar, axis-saaj-1.3.jar, bcmail-jdk14-1.38.jar, bcprov-jdk14-1.38.jar, bctsp-jdk14-1.38.jar, c3p0-0.9.2-pre4.jar, cglib-2.2.jarcommons-beanutils-1.7.0.jar, commons-discovery-0.2.jar, cos-05Nov2002.jar, csv-1.0.jar, cxf-xjc-runtime-2.6.2.jar, cxf-xjc-ts-2.6.2.jar, dom4j-1.6.1.jar, ezmorph-1.0.4.jar, gson-2.5.jar, jai_codec-1.1.3.jar, jai_core-1.1.3.jar, JavaEWAH-0.8.6.jar, javax.json-api-1.0.jar, jjtraveler-0.6.jar, jsonld-java-jena-0.2.jar, json-lib-2.2.2-jdk15.jar, lucene-analyzers-common-5.3.1.jar, lucene-core-5.3.1.jar, lucene-memory-5.3.1.jar, lucene-queries-5.3.1.jar, lucene-queryparser-5.3.1.jar, lucene-sandbox-5.3.1.jar, mail-1.4.jar, mchange-commons-java-0.2.2.jar, shared-objects-1.4.9.jar, sparqltag-1.0.jar, stax-api-1.0-2.jar, wsdl4j-1.5.1.jar

3.2 Building VIVO in 3 tiers

Draft

This page is being updated for 1.10 and Maven. Check back for more.

Add a third layer to the VIVO distribution, to keep all of your modifications in one place.

The three tiered build is very simple. In a standard two tier build VIVO replaces anything that is in Vitro as it adds its own special files to the mix. In a three tier we replace anything in Vitro or VIVO with the files that we want to add or modify for our version of VIVO. This allows us to point at newer VIVO versions without modifying the files within.

The method in this document has been copied from the EarthCollab work at: https://github.com/NCAR/2014-EarthCube-BuildingBlocks-EnablingCollaboration-14402930-vivo-source

The files that differ from a regular VIVO build (other than the ones you are adding and changing) are the build.xml and build.properties file.

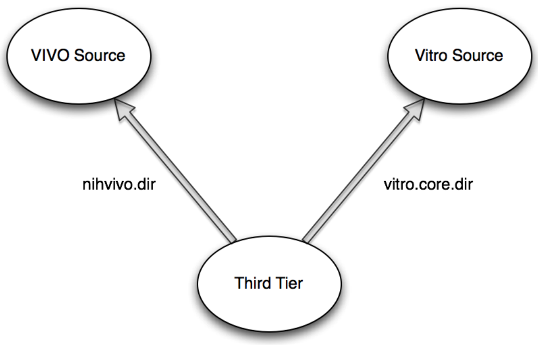
There are two standard ways to setup VIVO, for development or deployment.

3.2.1 Development

With development, you want to look at your VIVO against the VIVO and Vitro sources. This means getting the source for each project from the git repository.

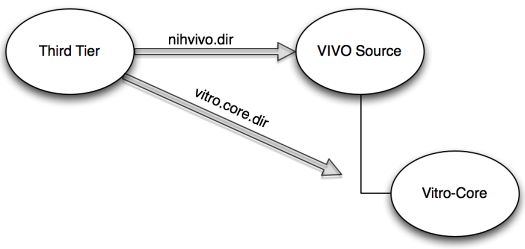
- https://github.com/vivo-project/Vitro
- https://github.com/vivo-project/VIVO

Your deploy.properties then points at the individual repositories. When you build it pulls the three repositories together and sends them as a single package to Tomcat.



3.2.2 Deployment

In deployment you have just your third tier and the released source for VIVO. In that released source is the vitro code that goes with the latest release of VIVO. Your deploy.properties for a deployment build will need to point to the internal vitro-core folder for the vitro code.



3.2.3 Project template

If you would like to get started with the three tired build process, there is a project template on Github[73] that includes the necessary build.xml, build.properties and directory structure. This template uses Git submodules[74] to pull in VIVO and Vitro from the main vivo-project repository.

73 https://github.com/lawlesst/vivo-project-template
74 http://git-scm.com/book/en/Git-Tools-Submodules

4 Exploring VIVO

4.1 Overview

As a first time VIVO user, you should take some time to familiarize yourself with the interface, learn how VIVO stores data, consider options for configuring VIVO, and learn how to create user accounts in VIVO.

When VIVO starts up the very first time, it will notice that its data store is empty, and will proceed to load data from `firsttime` directories. Data in these directories include the ontologies to be used by VIVO. These data are loaded exactly once. VIVO will then proceed to load data in the `everytime` and `filegraph` directories. Anytime VIVO is started after the first time, it will check for data in the `everytime` and `filegraph` directories and compare that data with the data in its data store and update the data store as necessary.

In this section we provide an exercise regarding sample data. You can load the sample data into your VIVO, explore the interface, and learn how the sample data is stored. When you are finished with the sample data, you can restore your VIVO to its "firsttime" state. From there you can review the configuration options, choose options best suited for your VIVO, and then create user accounts. You will then be ready to begin loading your data into your VIVO, providing your institution with data for finding experts and representing their scholarship.

4.2 Logging in to VIVO

To log into VIVO using the web browser, navigate to your institution’s instance of VIVO.

- Click the "Log in" link near the upper right corner.
- Enter your username (usually email or external authentication ID) and your password (see note below)
- Click the “Log in” button and you will be redirected to the Home page.

Note: *If you have not yet created any user accounts in VIVO, you can log in as the root user that you set up in the configuration file (rootUser.emailAddress in runtime.properties). If this is your first time logging in, the password will be "rootPassword". You will be required to set a new password to complete the login process.*

4.3 Sample Data

In Draft

This page is being written. Please do not use the content of this page until this banner is removed.

4.3.1 Overview

VIVO has many features and can display many kinds of data related to the scholarship of the individuals at your institution. VIVO provides sample data which can demonstrate the features of VIVO, and help familiarize you with the formats VIVO uses to store its data. The sample data includes a fictional university, fictional departments, fictional faculty members and collaborators, fictional publications and grants, memberships and other elements of scholarship common in VIVO. The sample data does not include an example of every type of thing that can be stored in VIVO, nor does it contain many faculty. It is intended to demonstrate the most common elements of VIVO.

4.3.2 Preparing Your VIVO

To use the sample data, and follow the examples here regarding the sample data, you will want your VIVO to contain only the sample data. Do not add the sample data to your data. VIVO uses data from a database you specify. We will create a database for the sample data, and tell VIVO to use that database when loading and using the sample data. At the end of this page, we will tell VIVO to use the database that you used when you installed VIVO. In this way, your data, and the sample data will always be separate.

In the steps that follow, we assume that you have installed VIVO according to the installation instructions, and that MySQL and Tomcat are running. If this is not the case, please complete the installation and test it, before attempting to use the instructions here.

To create a database for the sample data, and tell VIVO to use it, follow the steps below:

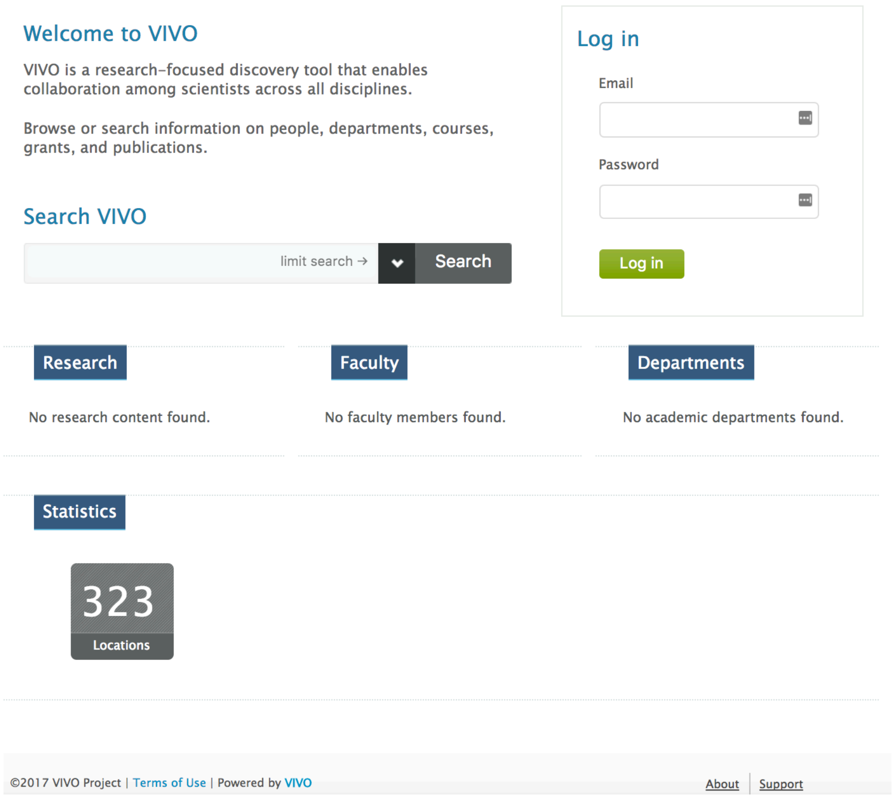
1. Record the name of the database you used to install VIVO. In the installation instructions, this is referred to as `vitrodb`, but you may have named it something else. After you have finished with the sample data, you will follow steps to reset VIVO to use this database.
2. Create a new database in MySQL

   Create sample database

   ```
   $ mysql -u root -p
   Welcome to the MySQL monitor.  Commands end with ; or \g.
   Your MySQL connection id is 2
   Server version: 5.7.9 MySQL Community Server (GPL)
   Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.
   Oracle is a registered trademark of Oracle Corporation and/or its
   affiliates. Other names may be trademarks of their respective
   owners.
   Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

   mysql> CREATE DATABASE sampledb CHARACTER SET utf8;
   mysql> GRANT ALL ON sampledb.* TO 'vitrodbUsername'@'localhost' IDENTIFIED BY 'vitrodbPassword';
   ```

3. Edit the `runtime.properties` file to tell VIVO the name of the database it should use. In this case `sampledb`.
4. Restart Tomcat

On completing these steps, your VIVO will be using an "empty" database. You may need to wait for VIVO to start the first time, as VIVO automatically loads data from files distributed with VIVO. You may ned to refresh your browser. You may need to wait will VIVO indexes the first time data. It may take several minutes for VIVO to restart, load the first time data, and index it. When VIVO has completed its work, and you have refreshed your browser, your home page should look like:


Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

Search VIVO

No research content found.

No faculty members found.

No academic departments found.

©2017 VIVO Project | Terms of Use | Powered by VIVO

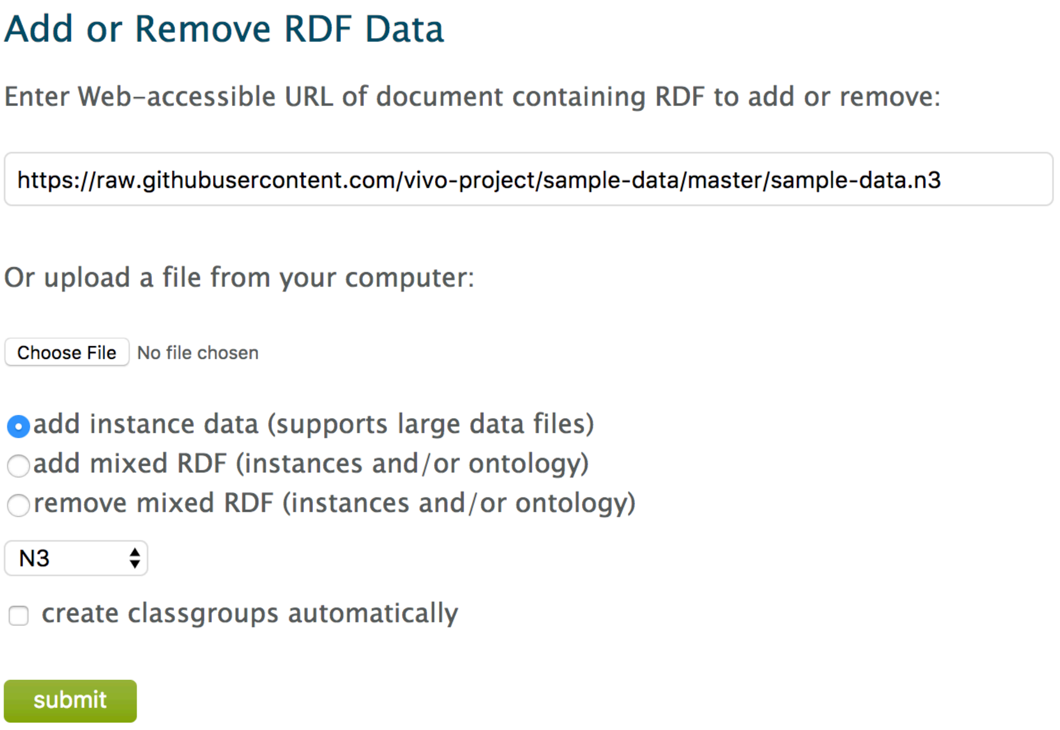
You are are now ready to load the sample data.

4.3.3 Loading the Sample Data

To load the sample data, follow the steps below.

1. Log in to your VIVO as a site admin
2. Goto to Site Admin / Add/Remove RDF Data

3. Enter the address of the sample data, https://raw.githubusercontent.com/vivo-project/sample-data/master/sample-data.n3, select "add instance data," and set the file type to "N3." See below



Add or Remove RDF Data

Enter Web-accessible URL of document containing RDF to add or remove:

https://raw.githubusercontent.com/vivo-project/sample-data/master/sample-data.n3

Or upload a file from your computer:

Choose File No file chosen

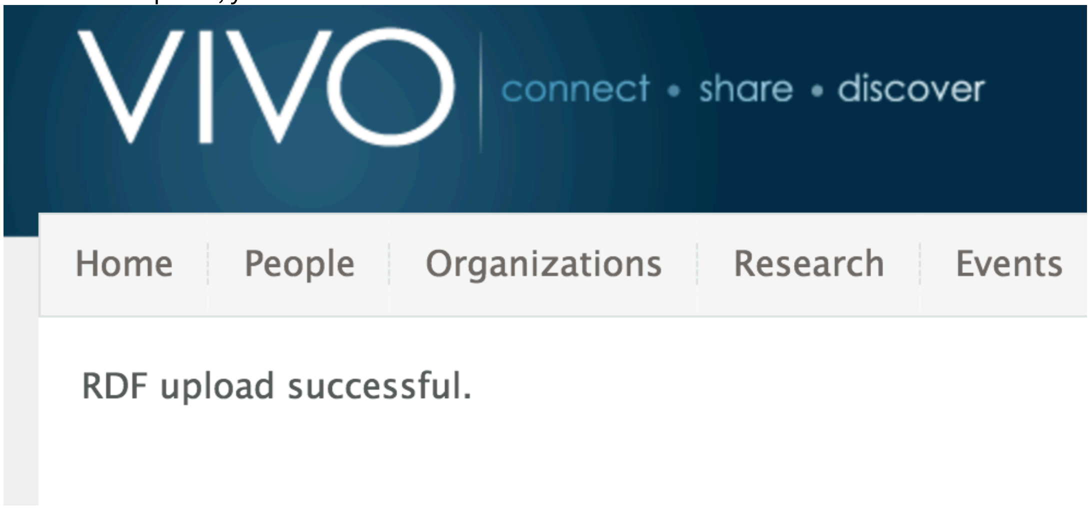
[✓ add instance data (supports large data files)]
add mixed RDF (instances and/or ontology)
remove mixed RDF (instances and/or ontology)

N3

☐ create classgroups automatically

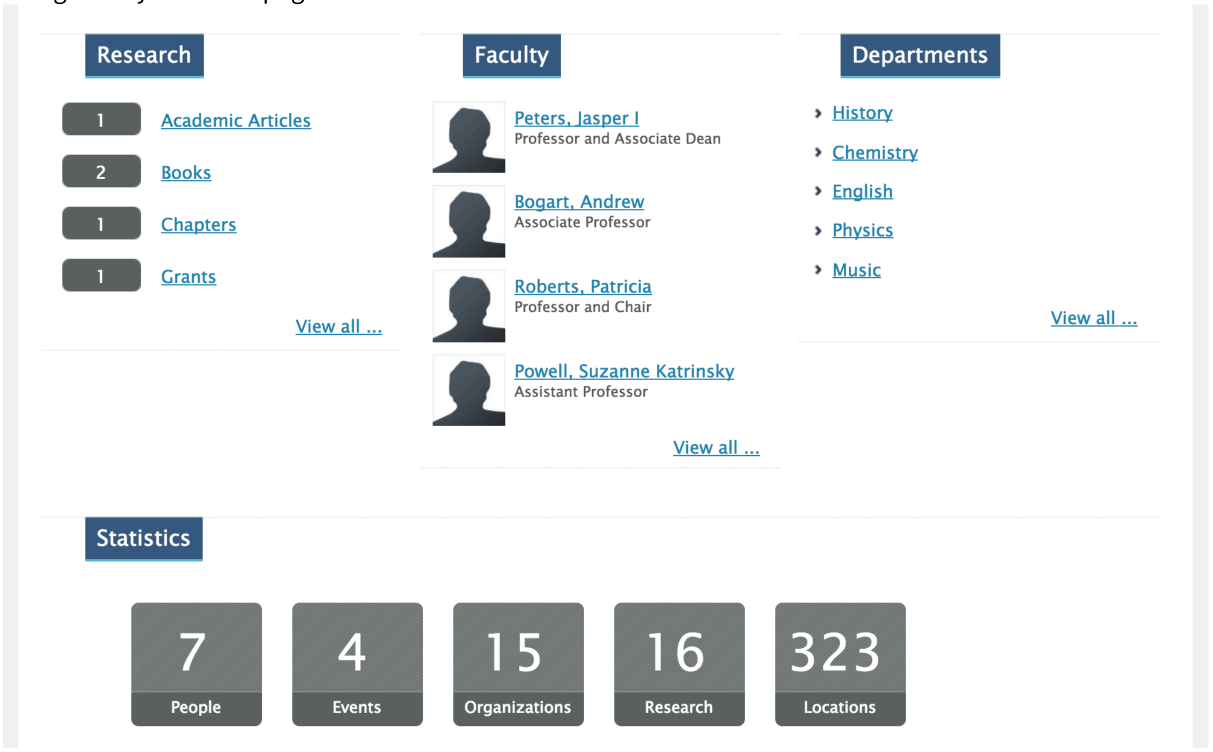
submit

4. Check to make sure you have the form filled out properly: 1) the URL for the sample data has been entered as shown; 2) "add instance" is selected; 3) "N3" is selected as the file type. Press Submit.
5. After a brief upload, you will see



RDF upload successful.

6. Navigate to your home page. You should see



7. That's it! Let's start exploring.

4.3.4 Exploring the Interface

4.3.5 Exploring the Data

4.3.6 Resetting Your Database

When you are finished exploring the sample data, you will want to reset VIVO to use the database you used when installing VIVO. You recorded the name of this database when following the steps above in Preparing Your Database. To reset your database, follow the steps below:

1. Shutdown Tomcat
2. edit runtime.properties and change the name of the database from the name of the sample database to the name of the database you used to install VIVO.
3. Start Tomcat

Now VIVO will be using the database you used when installing VIVO. If you would like to use the sample data again, just repeat these steps, naming the sample data database as the one you would like to use.

4.4 Restoring VIVO to First Time State

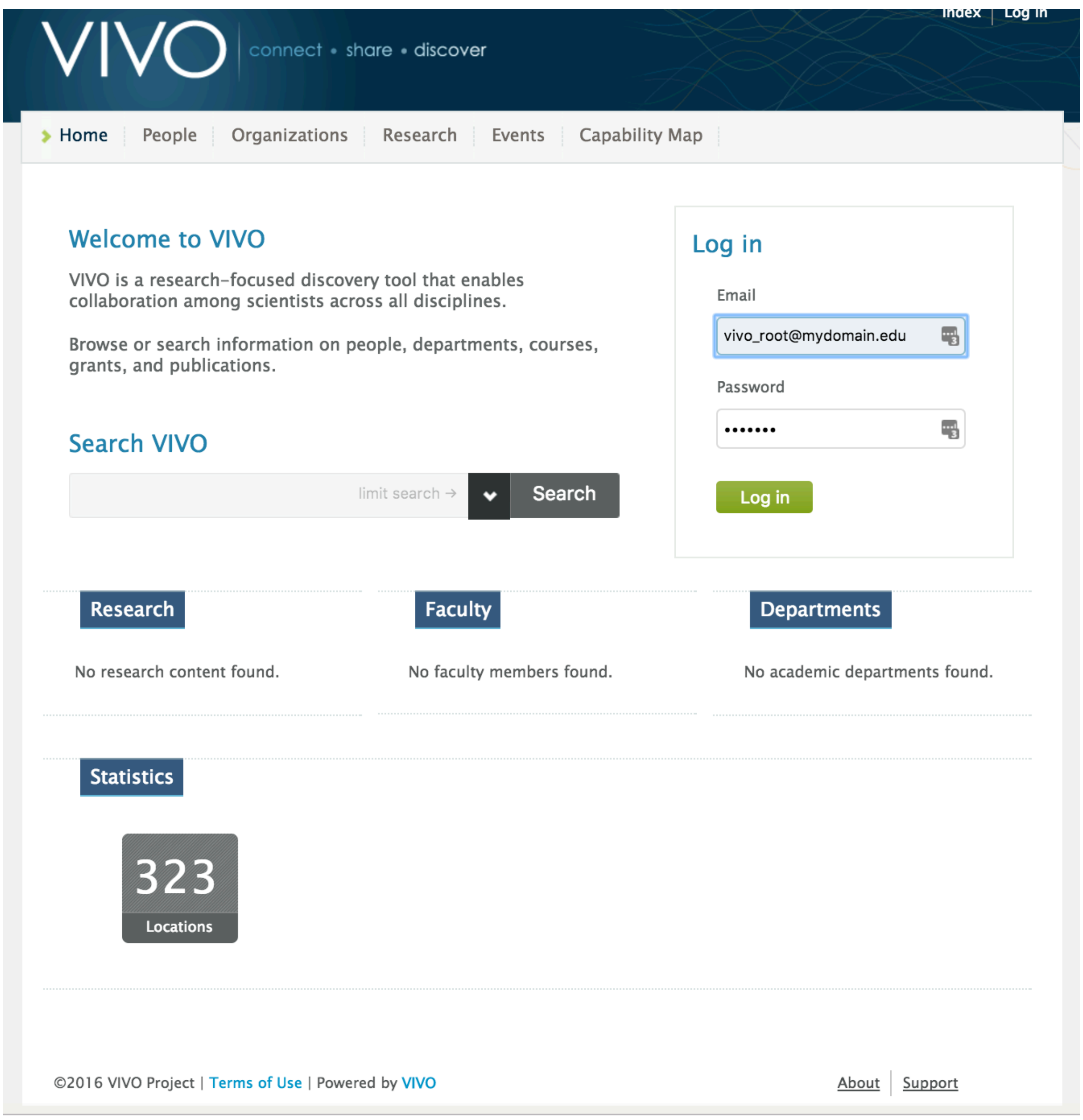
When Installing VIVO (see page 32), you created a MySQL database to hold the data and edited the `runtime.properties` file to tell VIVO the path to the data base and the username and password for root access to the database. To restore VIVO to first time state, simply drop the MySQL database, recreate it and restart Tomcat.

See below.

```
$ mysql -u root -p
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.9 MySQL Community Server (GPL)
Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> DROP DATABASE vitrodb;
mysql> CREATE DATABASE vitrodb CHARACTER SET utf8;
mysql> GRANT ALL ON vitrodb.* TO 'vitrodbUsername'@'localhost' IDENTIFIED BY 'vitrodbPassword';
```

Now restart Tomcat. VIVO will detect an empty data store, and proceed to load it from the `firsttime`, `everytime` and `filegraph` directories. Tomcat will start, but search indices may take additional time to complete. When reindexing is complete, refresh your browser to see your VIVO returned to first time state.

Your first time VIVO home should appear as shown below.


Index | Log in
VIVO connect • share • discover
Home | People | Organizations | Research | Events | Capability Map

Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

Search VIVO

Log in

Email: vivo_root@mydomain.edu

Password: •••••••

Research

No research content found.

Faculty

No faculty members found.

Departments

No academic departments found.

Statistics

323 Locations

©2016 VIVO Project | Terms of Use | Powered by VIVO

5 Preparing for Production

5.1 Overview

One you have an initial VIVO running, loaded sample data, and explored the interface, you should be ready to make some choices about your production VIVO.

In this section, we will see the choices that need to be made before launching a production VIVO site. Choices will lead to restarting VIVO to confirm things are working as expected, and clearing out data that was created during testing.

5.2 Minimum Configuration

5.2.1 Email

VIVO should be configured to send email using an smtp service of your choice. VIVO sends email from its contact form. Email also sends email to users when changes are made to their accounts. Modify the two line below in runtime.properties, then restart Tomcat.

Email parameters in runtime.properties

```
#
# Email parameters which VIVO can use to send mail. If these are left empty,
# the "Contact Us" form will be disabled and users will not be notified of
# changes to their accounts.
#

email.smtpHost = smtp.mydomain.edu
email.replyTo = vivoAdmin@mydomain.edu
```

5.2.2 Namespace

The namespace parameter is the single most important parameter in your VIVO configuration. It is used in every triple created by VIVO. It should match the domain name of your VIVO production site. So, if your VIVO production site will be reached on the Internet with a web address of http://vivo.mydomain.edu the Vitro.defaultNamepsace parameter is runtime.properties should be set as shown below.

Namespace parameter in runtime.properties

```
  #
  # This namespace will be used when generating URIs for objects created in the
  # editor. In order to serve linked data, the default namespace must be composed
  # as follows (optional elements in parentheses):
  #
  # scheme + server_name (+ port) (+ servlet_context) + "/individual/"
  #
  # For example, Cornell's default namespace is:
  #
  # http://vivo.cornell.edu/individual/
  #

Vitro.defaultNamespace = http://vivo.mydomain.edu/individual/
```

5.2.3 Additional Configuration

For additional configuration parameters see Configuration Reference (see page 295)

5.3 Create, Assign, and Use an Institutional Internal Class

5.3.1 Overview

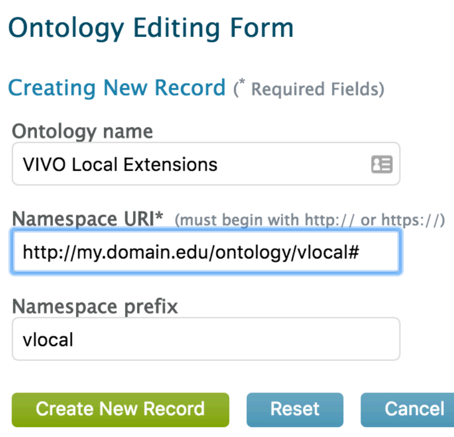
VIVO supports the concept of an Institutional Internal Class, a class that you can create and assign to your people, and other entities, to indicate that they are part of your institution. Using an Institutional Internal Class, you can limit VIVO's displays of entities to those in your institution.

5.3.2 Create an Institutional Internal Class

Create a local ontology if you do not already have one. If you have one, add a class to your existing local ontology.

Go to Site Admin > Ontology list > Add new ontology

Add your new ontology (see the example below)


Ontology Editing Form

Creating New Record (* Required Fields)

Ontology name	VIVO Local Extensions
Namespace URI* (must begin with http:// or https://)	http://my.domain.edu/ontology/vlocal#
Namespace prefix	vlocal

Ontology name: Whatever you want. The name you give will appear in the list of VIVO ontologies.

Namespace: Must be your domain name as specified in your runtime.properties, followed by "/ontology/" followed by a name of your choice, followed by the '#' sign.

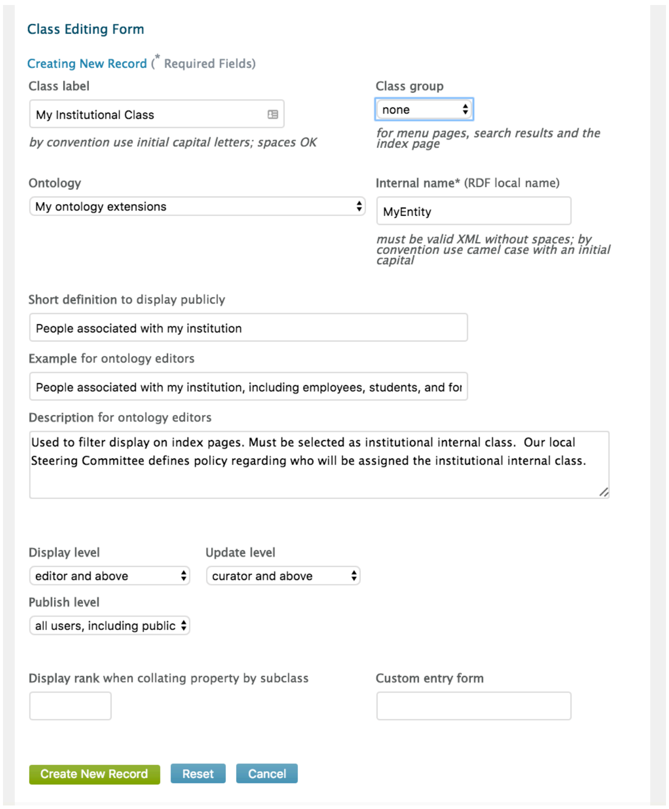
Namespace prefix: a short word. This word will appear in the prefix list in your SPARQL windows and will be used by you in any SPARQL queries referring to your local ontology.

Submit Changes

Add a new class to your local ontology

Go to 'Hierarchy of Classes Defined in This Namespace' > Add New Class

Add your new class (see the example below)



Class Editing Form

Creating New Record (* Required Fields)

Class label: My Institutional Class

by convention use initial capital letters; spaces OK

Class group: none

for menu pages, search results and the index page

Ontology: My ontology extensions

Internal name* (RDF local name): MyEntity

must be valid XML without spaces; by convention use camel case with an initial capital

Short definition to display publicly: People associated with my institution

Example for ontology editors: People associated with my institution, including employees, students, and for

Description for ontology editors: Used to filter display on index pages. Must be selected as institutional internal class. Our local Steering Committee defines policy regarding who will be assigned the institutional internal class.

Display level: editor and above

Update level: curator and above

Publish level: all users, including public

Display rank when collating property by subclass

Custom entry form

Class label: Text, describes the class. This will be visible on the VIVO interface.

Class Group: People. This allows the class to be used to restrict the display of people to those people who have the institutional class you are defining.

Ontology: Select your previously created local ontology from the drop down menu

Internal Name: This word will be used in your SPARQL queries, along with the local ontology prefix to refer to your local class. In this example, the complete reference in SPARQL would be vlocal:MyEntity.

Short definition to display publically: Use language your users will understand

Example for ontology editors: More detail. Can be very technical.

Description for ontology editors: Will appear in the ontology editor. Remind future ontology editors of the purpose of the class and how one will know what entities should be in the class.

Display level: Set to editor and above from the drop down

Update level: Set to curator and Above from the drop down

Publish level: All users including public

Once you are satisfied with the values, press Create New Record

5.3.3 Assign your Institutional Internal Class

Method 1: (Manual)

Go to the person in the UI

Click Edit Individual

Click Add Type

Select your Institutional Internal Class from the drop down

Method 2: (Bulk)

Create a set of RDF, one triple per person you would like to have in the institutional class. Each triple will look like

```
<personuri>  rdf:type vlocal:MyEntity .
```

Go to Site Admin -> Add/remove RDF Data and add your triples

5.3.4 Use your Institutional Internal Class

Define institutional internal class

Go to Site Admin > Institutional internal class

Select your new class from the dropdown menu

To restrict display to only those people in your institution,

Go to Site Admin > Page Management > People

Click the plus sign to expand the 'Browse Class Group' box

Check 'Only display people within my institution'

Click "Save this Content'

5.4 Adding User Accounts

The root user and users with the role Site Admin are allowed to create user accounts. To create a user, one has to follow these steps:

1. Enter the 'Site Administration'.
2. Choose 'User Accounts' from the menu 'Site Configuration'.
3. Click on 'Add new account'
4. Mandatory fields to enter are 'Email Address', 'First name', and 'Last name'.
5. Choose a role for the respective user. More information about the role management can be found in the 'System Administration' section of this documentation on the page 'Creating and Managing User Accounts'.
6. Click 'Add new account'

In normal operation, an email will be sent to the address entered notifying that an account has been created. It will include instructions for activating the account and creating a password.

For more information on user accounts, see Creating and Managing User Accounts (see page 256)

6 Using VIVO

VIVO [Pronunciation: /viːvəʊ/ or *vee-voh*] is member-supported, open source software, and an ontology for representing scholarship. VIVO supports recording, editing, searching, browsing and visualizing scholarly activity. VIVO encourages research discovery, expert finding, network analysis and assessment of research impact. VIVO is easily extended to support additional domains of scholarly activity.

Here we will describe the basic features of VIVO and how you can use them. Many VIVO sites customize VIVO to add local features, enriching the description of the scholarship of their institution. Here we describe only the common features of VIVO. You may wish to ask your local VIVO providers for documentation describing the VIVO at your institution.

The examples to follow use an uncustomized VIVO. Your VIVO may look different.

6.1 Navigating VIVO

VIVO is navigated and browsed using a primary menu located along the upper portion of the website. By default, VIVO has five items in the primary menu; Home, People, Organizations, Research, and Events. Your VIVO may have been configured with additional menu items.

Beginning at the Home menu, notice the contents of each menu. Each menu contains a list of VIVO Individuals associated to particular "classes". Each menu shows the count of individuals in parentheses next to the superclass and subclass names. In addition, the "Index" link on the upper right corner displays a list of “VIVO individuals” by class and the associated count in parentheses.

? Unknown Attachment

6.2 Editing Your Profile (*)

6.3 Using Search (*)

6.4 Using the Capability Map

6.4.1 Overview

The VIVO Capability Map provides a visual means for finding people working in particular areas and for navigating through concepts and people. Using the capability map, one might find people to invite to a workshop, colleagues for a grant application, members of a dissertation committee, or experts for a technical advisory group. The map is very easy to use.

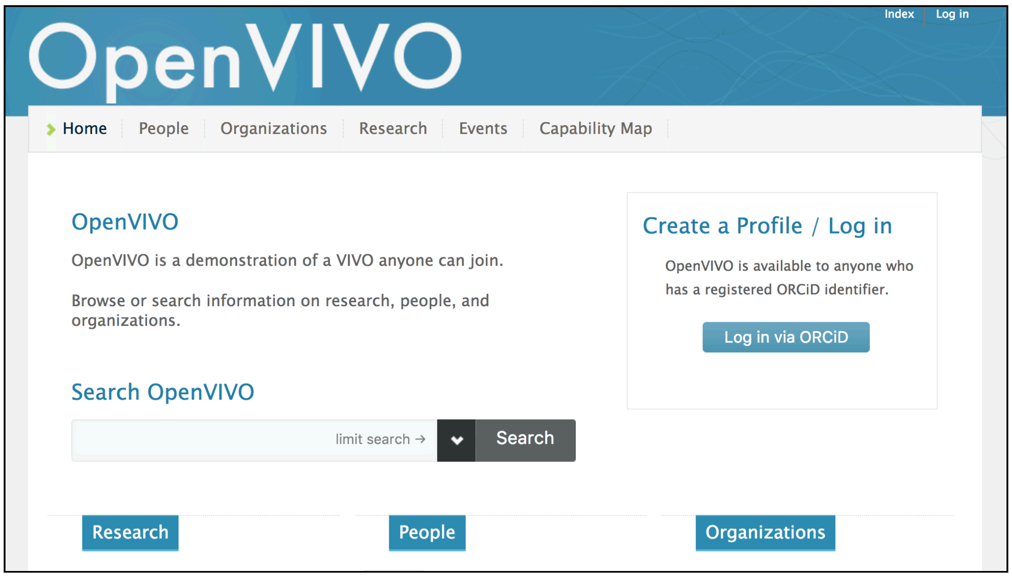
The capability map was originally developed by a team at the University of Melbourne for their VIVO, Find an expert[75]. The concept was developed for OpenVIVO[76] and introduced to VIVO in version 1.9.0.

The capability map uses D3.js[77], a visualization tool used by the NY Times and others to provide modern graphics that are interactive, and responsive – that is, work on any device in any modern browser from a phone to a desktop computer.

In the examples that follow, we will use OpenVIVO. If your local VIVO is at version 1.9.0 or higher, it will have the capability map as described here.

6.4.2 A Tour of the Capability Map

Think of a research area that interests you. Perhaps you study ontologies and want to find other people who are interested in ontology. Click on the Capability Map link in the menu bar on the VIVO home page (see below). You do not need to be logged on. If you are using your local VIVO and you do not see the capability map link on the home page, please contact your local VIVO support for more information.



OpenVIVO

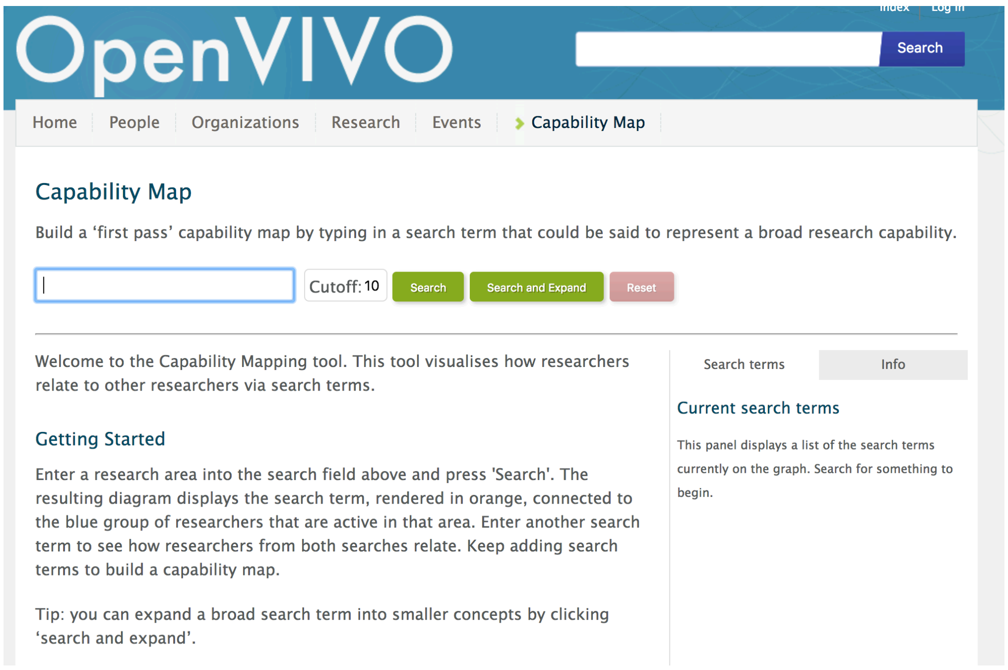
OpenVIVO is a demonstration of a VIVO anyone can join.

Browse or search information on research, people, and organizations.

OpenVIVO is available to anyone who has a registered ORCiD identifier.

You will see the Capability Map page. Everything you do with the Capability Map will be done from this page. There are directions on the page. You'll want to read them all. Scroll down the page to see the rest of the directions.

75 https://findanexpert.unimelb.edu.au/
76 http://openvivo.org
77 http://d3js.org



Capability Map

Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

Welcome to the Capability Mapping tool. This tool visualises how researchers relate to other researchers via search terms.

Getting Started

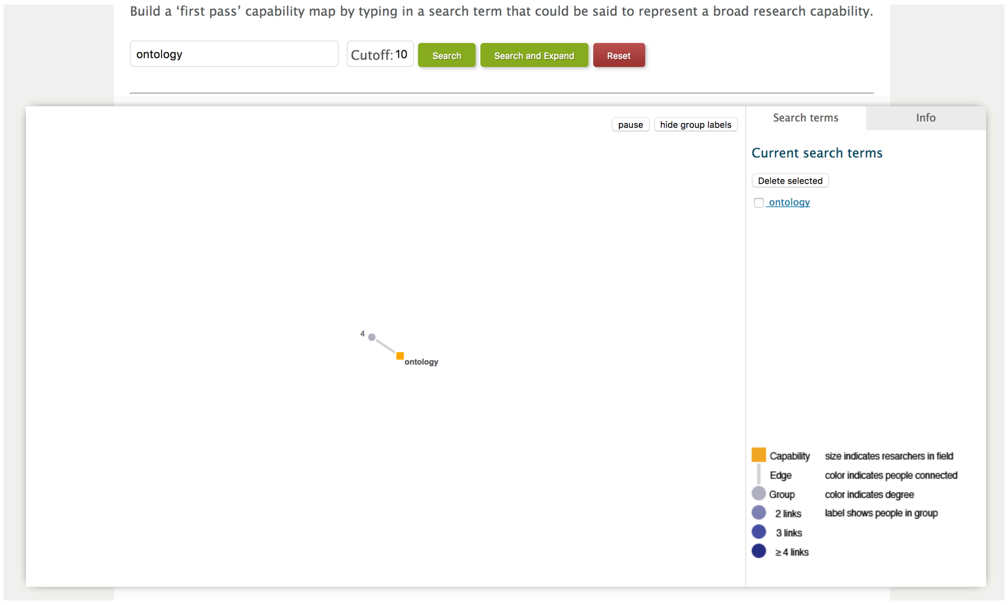
Enter a research area into the search field above and press 'Search'. The resulting diagram displays the search term, rendered in orange, connected to the blue group of researchers that are active in that area. Enter another search term to see how researchers from both searches relate. Keep adding search terms to build a capability map.

Tip: you can expand a broad search term into smaller concepts by clicking 'search and expand'.

Current search terms

This panel displays a list of the search terms currently on the graph. Search for something to begin.

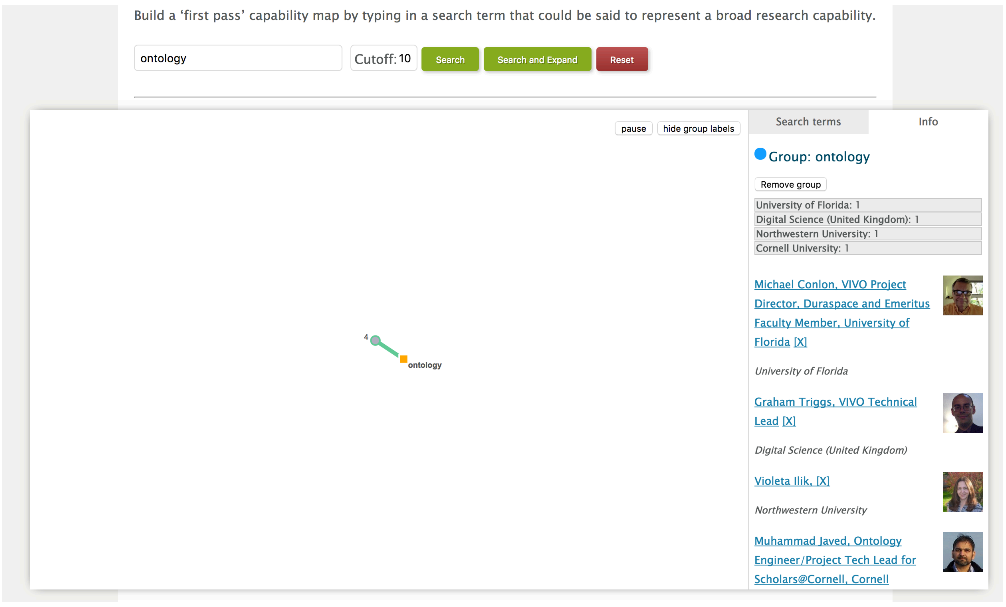
Type ontology into the text box and press Search. You should see something similar to below. OpenVIVO is a VIVO anyone can join. As a result, the data in OpenVIVO at the time you are following these examples may be different from the data at the time the examples were captured for this document.


Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.
ontology
Cutoff: 10
Search
Search and Expand
Reset
pause
hide group labels
Search terms
Info
Current search terms
Delete selected
ontology
4
ontology
Capability
size indicates researchers in field
Edge
color indicates people connected
Group
color indicates degree
2 links
label shows people in group
3 links
≥4 links

Doesn't look like much, but we're just getting started. we have a very simple "graph" – a network diagram. The diagram has a legend in the lower right. There are three kinds of things on the graph: 1) terms or capabilities (orange squares); 2) groups of people (purple circles), and 3) edges (grey lines). Terms or capabilities are also called "research areas" or "research interests". They are concepts that are associated with individuals through works such as publications, grants, datasets and presentations, and through self-identification. People with VIVO profiles can select their research areas of interest using the profile editing features of VIVO. Each orange square represents a concept, and each is labeled. The purple dots represent groups of people. People are in the group if they have the research interests that connect to the group. In the figure above, there is just one concept – ontology – and just one group – the group of people with ontology as a research interest. Note the number of people in the group is shown

(4). If there is just one person, the person's name will be shown. The edge indicates that the people in the group are "connected" to the term.

Click on the group of people. The right hand area indicates the people who are in the group. You can click on any of them to go to their VIVO page.


Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

ontology Cutoff: 10 Search | Search and Expand | Reset

pause | hide group labels | Search terms | Info

Group: ontology

Remove group

University of Florida: 1
Digital Science (United Kingdom): 1
Northwestern University: 1
Cornell University: 1

Michael Conlon, VIVO Project Director, Duraspace and Emeritus Faculty Member, University of Florida [X]

University of Florida

Graham Triggs, VIVO Technical Lead [X]

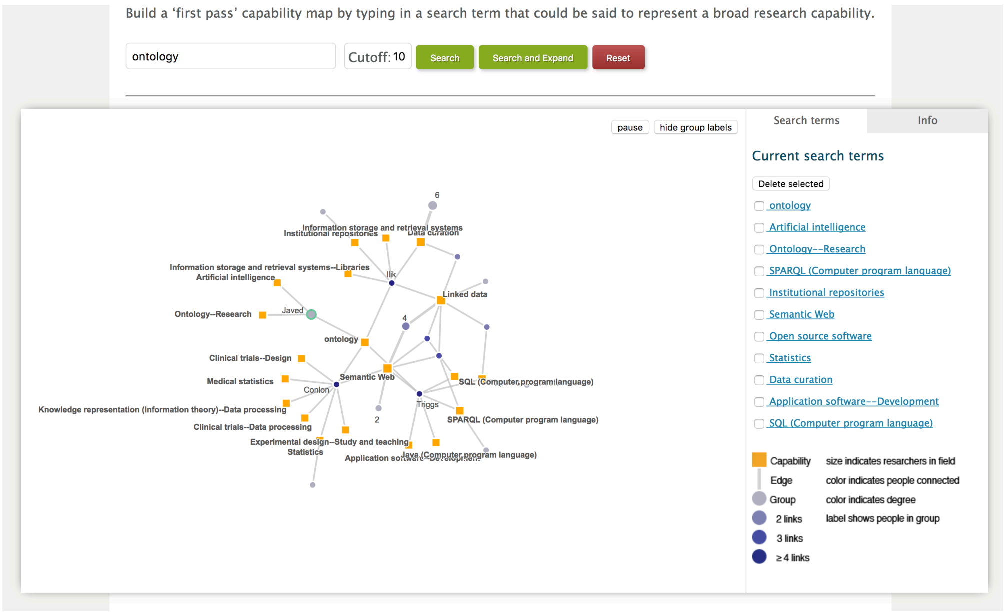
Digital Science (United Kingdom)

Violeta Ilik, [X]

Northwestern University

Muhammad Javed, Ontology Engineer/Project Tech Lead for Scholars@Cornell, Cornell

Let's add to the network. Click on Search and Expand. We now see the concepts of each of the four original people, with people associated with those concepts, connected in a graph showing common interests. Some people have many interests, some have just a few. Note that concepts are always connected to groups of people who have that concept has a research interest.


Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

ontology Cutoff: 10 Search | Search and Expand | Reset

pause | hide group labels | Search terms | Info

Current search terms

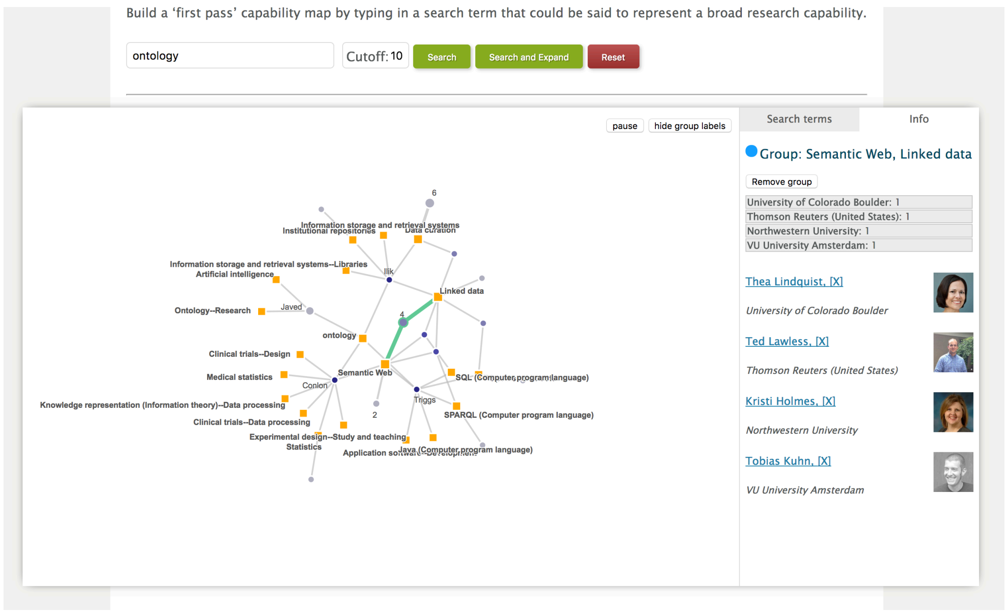
Delete selected

- ontology
- Artificial intelligence
- Ontology--Research
- SPARQL (Computer program language)
- Institutional repositories
- Semantic Web
- Open source software
- Statistics
- Data curation
- Application software--Development
- SQL (Computer program language)

🟧	Capability	size indicates researchers in field
	Edge	color indicates people connected
●	Group	color indicates degree
●	2 links	label shows people in group
●	3 links	
●	≥4 links	

6
Information storage and retrieval systems
Institutional repositories
Data curation
Information storage and retrieval systems--Libraries
Artificial intelligence
Ilik
Linked data
Ontology--Research
Javed
4
ontology
Clinical trials--Design
Semantic Web
Medical statistics
Conlon
SQL (Computer program language)
Knowledge representation (Information theory)--Data processing
Triggs
2
SPARQL (Computer program language)
Clinical trials--Data processing
Experimental design--Study and teaching
Statistics
Application software--Development
Java (Computer program language)

Click on the group of people near the center of the map, between linked data and semantic web. You should see a display similar to the one below. The group is highlighted along with the concepts common to the group (linked data and semantic web). Not that the right hand display shows the group as being defined as those people who are interested in both linked data and semantic web.



Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

ontology Cutoff: 10

Group: Semantic Web, Linked data

University of Colorado Boulder: 1
Thomson Reuters (United States): 1
Northwestern University: 1
VU University Amsterdam: 1

Thea Lindquist, [X]
University of Colorado Boulder

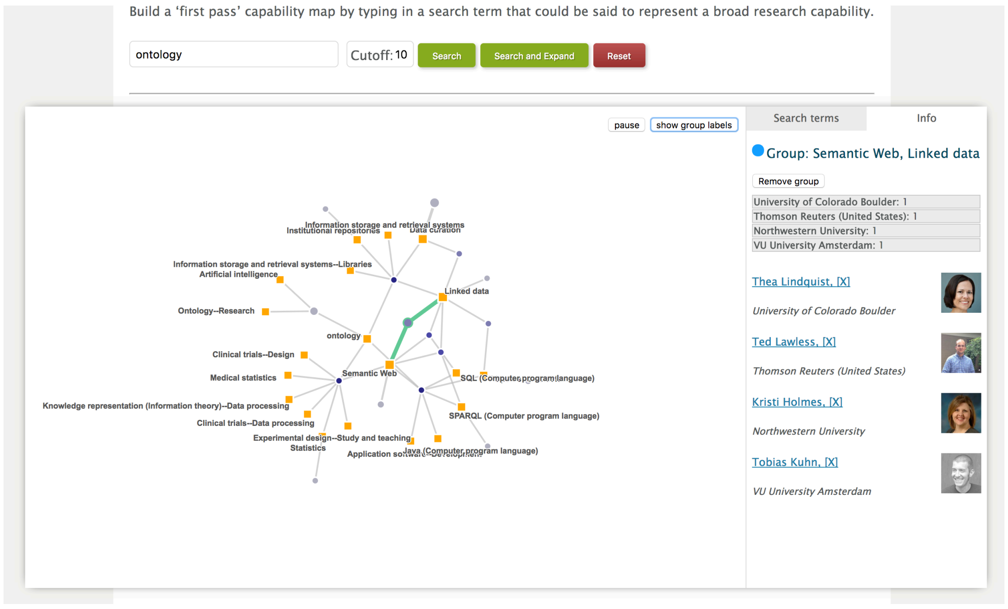
Ted Lawless, [X]
Thomson Reuters (United States)

Kristi Holmes, [X]
Northwestern University

Tobias Kuhn, [X]
VU University Amsterdam

Search
Search and Expand
Reset
pause
hide group labels
Search terms
Info
Remove group
Information storage and retrieval systems
Institutional repositories
Data curation
Information storage and retrieval systems--Libraries
Artificial intelligence
Ontology--Research
Javed
Ilik
Linked data
ontology
Clinical trials--Design
Medical statistics
Semantic Web
Conlon
Triggs
SQL (Computer program language)
SPARQL (Computer program language)
Knowledge representation (Information theory)--Data processing
Clinical trials--Data processing
Experimental design--Study and teaching
Statistics
Application software--Development
Java (Computer program language)
6
4
2

The map is a little cluttered with the names of the people who are in groups with one member. Click hide group labels. You see a map similar to the one below.



Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

ontology Cutoff: 10

Group: Semantic Web, Linked data

University of Colorado Boulder: 1
Thomson Reuters (United States): 1
Northwestern University: 1
VU University Amsterdam: 1

Thea Lindquist, [X]
University of Colorado Boulder

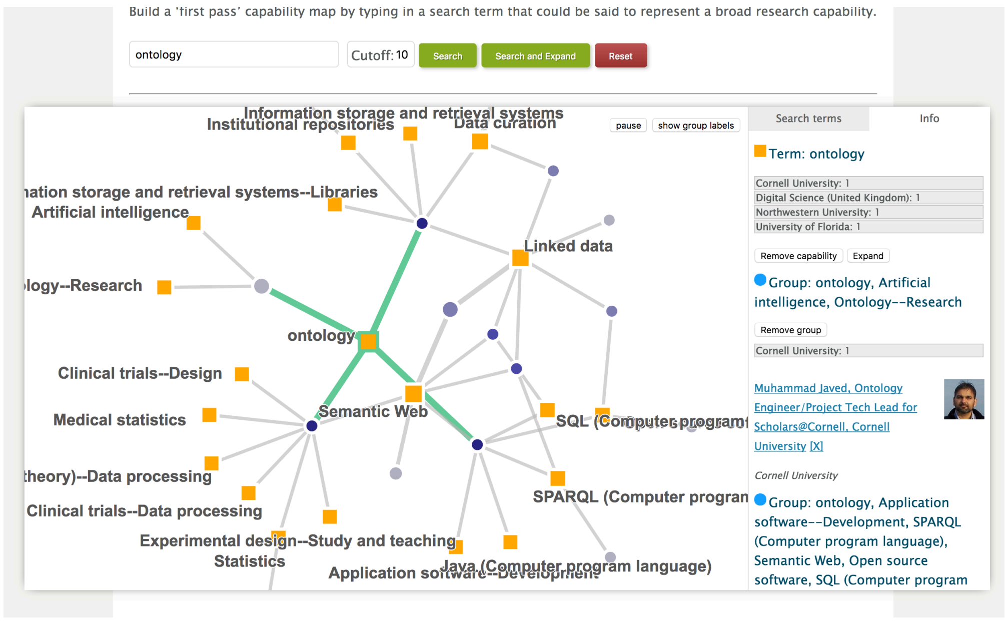
Ted Lawless, [X]
Thomson Reuters (United States)

Kristi Holmes, [X]
Northwestern University

Tobias Kuhn, [X]
VU University Amsterdam

Search
Search and Expand
Reset
pause
show group labels
Search terms
Info
Remove group
Information storage and retrieval systems
Institutional repositories
Data curation
Information storage and retrieval systems--Libraries
Artificial intelligence
Ontology--Research
Linked data
ontology
Clinical trials--Design
Medical statistics
Semantic Web
SQL (Computer program language)
SPARQL (Computer program language)
Knowledge representation (Information theory)--Data processing
Clinical trials--Data processing
Experimental design--Study and teaching
Statistics
Application software--Development
Java (Computer program language)

Double click on ontology. Two things happen. The map zooms in one ontology, and the ontology term is highlighted. One of the groups connected to ontology is highlighted.


Build a 'first pass' capability map by typing in a search term that could be said to represent a broad research capability.

ontology | Cutoff: 10 | Search | Search and Expand | Reset

pause | show group labels

Search terms | Info

Term: ontology

Cornell University: 1
Digital Science (United Kingdom): 1
Northwestern University: 1
University of Florida: 1

Remove capability | Expand

Group: ontology, Artificial intelligence, Ontology--Research

Remove group

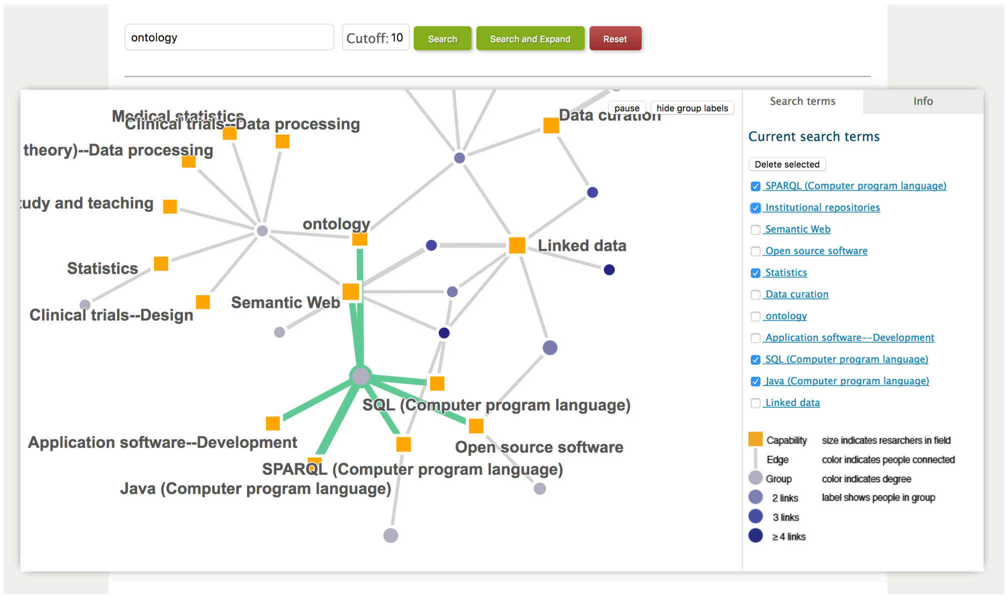
Cornell University: 1

Muhammad Javed, Ontology Engineer/Project Tech Lead for Scholars@Cornell, Cornell University [X]

Cornell University

Group: ontology, Application software--Development, SPARQL (Computer program language), Semantic Web, Open source software, SQL (Computer program
Information storage and retrieval systems
Institutional repositories
Data curation
ation storage and retrieval systems--Libraries
Artificial intelligence
Linked data
logy--Research
ontology
Clinical trials--Design
Semantic Web
Medical statistics
SQL (Computer program
theory)--Data processing
SPARQL (Computer progran
Clinical trials--Data processing
Experimental design--Study and teaching
Statistics
Java (Computer program language)
Application software--Development

suppose we wish to reduce the number of terms that have been generated by expanding the graph – some of these are closely related to ontology, while others may be less so. Terms aon on the graph because someone who has ontology an an interest has one of these terms as an additional interest. Click on SQL (Computer Program Language), then in the right hand panel, click on "Search Terms" This lists all the terms currently in the graph.


ontology | Cutoff: 10 | Search | Search and Expand | Reset

pause | hide group labels

Search terms | Info

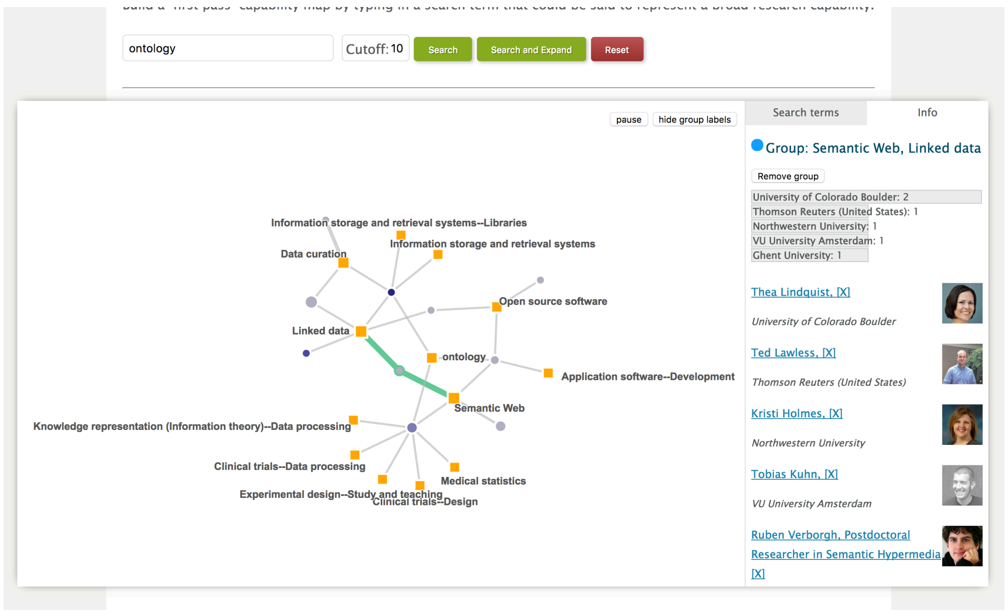
Current search terms

Delete selected

- [x] SPARQL (Computer program language)
- [x] Institutional repositories
- [ ] Semantic Web
- [ ] Open source software
- [x] Statistics
- [ ] Data curation
- [ ] ontology
- [ ] Application software--Development
- [x] SQL (Computer program language)
- [x] Java (Computer program language)
- [ ] Linked data

🟧	Capability	size indicates researchers in field
	Edge	color indicates people connected
●	Group	color indicates degree
●	2 links	label shows people in group
●	3 links	
●	≥ 4 links	
Medical statistics
Clinical trials--Data processing
theory)--Data processing
Data curation
tudy and teaching
ontology
Linked data
Statistics
Semantic Web
Clinical trials--Design
SQL (Computer program language)
Application software--Development
Open source software
SPARQL (Computer program language)
Java (Computer program language)

Select five of the terms as shown and press "Delete Selected" The pruned graph is shown below. We seem more clearly the group of people interested in linked data and the semantic web.


ontology
Cutoff: 10
Search
Search and Expand
Reset
pause
hide group labels
Search terms
Info
Remove group
Information storage and retrieval systems--Libraries
Information storage and retrieval systems
Data curation
Open source software
Linked data
ontology
Application software--Development
Semantic Web
Knowledge representation (Information theory)--Data processing
Clinical trials--Data processing
Medical statistics
Experimental design--Study and teaching
Clinical trials--Design

Group: Semantic Web, Linked data

University of Colorado Boulder: 2
Thomson Reuters (United States): 1
Northwestern University: 1
VU University Amsterdam: 1
Ghent University: 1

Thea Lindquist, [X]
University of Colorado Boulder
Ted Lawless, [X]
Thomson Reuters (United States)
Kristi Holmes, [X]
Northwestern University
Tobias Kuhn, [X]
VU University Amsterdam
Ruben Verborgh, Postdoctoral Researcher in Semantic Hypermedia, [X]

We can search and expand at any time to add groups and terms related to those on the graph. And if we want to start over, we can press Reset.

Using the Capability Map we can find groups of people interest in common research areas, and we can discover which research areas are often held in common across researchers.

6.5 Using Visualizations (*)

6.6 VIVO for Data Analysts

6.6.1 Background

VIVO represents data as triples. All data is represented and stored in the form subject, predicate, object. All entities are identified by URI. If you are unfamiliar with this method for data representation, see the references. A typical VIVO for a large research institution could have well over 10 million triples. Understanding which triples are needed for an analysis can be challenging. The VIVO community is here to help. Questions regarding data and data extraction using the techniques below can be posted to one of the VIVO Google Groups.

6.6.2 Getting Rectangles of Data

To get rectangles of data, use SPARQL queries. SPARQL is a simple query language designed for use with triple stores. A collection of SPARQL queries used to answer real-world questions is available here: SPARQL Queries (see page 91)

6.6.3 Getting Graphs of Data

The entire triple store can be unloaded for use in a local triple store, and for local query. This is recommended for sites wishing to make repeated analyst queries of the data. Community-editions of a triple stores are available with cost. Stardog is a popular, stable, and free triple store that can be used for this purpose. See http://stardog.com

To unload the triple store to a set of triples, use jena3tools, available here: https://github.com/vivo-project/jenatools

6.6.4 References

1. Borner, Conlon, Corson-Rikert, and Ding (eds) VIVO: A Semantic Approach to Scholarly Networking and Discovery, Morgan-Claypool Publioshers, 2012. 160 pages.
2. Allemang, and Hendler. Semantic Web for the Working Ontologist, second edition. Morgan-Kaufmann Publishers, 2011. 354 pages.
3. DuCharme. Learning SPARQL: Querying and Updating with SPARQL 1.1. O'Reilly, 2011. 235 pages.

7 Managing Data in Your VIVO

7.1 Importing Data to VIVO

7.1.1 Using the Convert CSV to RDF ingest tool

⚠ This guide describes just one (rather primitive) way to ingest data into VIVO. See https://wiki.duraspace.org/x/MYUQAg for a detailed discussion of data ingest.

Introduction

This guide will walk through the use of the 'Convert CSV to RDF' tool, a semi-automated method of converting comma separated or tab separated text files into RDF that can be displayed in VIVO. These files should include one row of data per record (e.g., a person or publication) and represent the fields or properties associated with each record in separate columns within the row, much as the values appear in a spreadsheet. The most common pattern of loading CSV files involves one CSV file per type of data to be loaded. Note, the current ingest tools involve working through a number of steps from original source data files to the appearance of new data in VIVO. The process requires some understanding of semantic web data modeling and some training.

Access the tool by navigating to the Site Administration section, clicking Ingest tools, then Convert CSV to RDF.

7.1.1.1 **Mapping Ontologies to other Ontologies**

When VIVO's ingest tools read in a CSV file, the data read from the file will be stored in the VIVO database as an extra "model," or secondary database managed by the Jena semantic web libraries underlying VIVO.

The next step is to link imported data sets using information stored with the source data. If an object not previously in VIVO has been found, a new record (individual) must be created in VIVO using the CONSTRUCT form of query in the SPARQL query language for RDF.

The query looks at the list of imported data and inserts statements creating a new individual wherever no match to an existing individual is found. The query of imported data is expressed using the ontology of the import; the CONSTRUCT statements use the class and property names of VIVO ontology. This accomplishes the mapping between the source ontology and the VIVO ontology.

When populating VIVO for the first time, all the data from a dataset can be added from the imported Jena model to VIVO by a CONSTRUCT query that again translates from the RDF in the imported model to the RDF in VIVO. When a dataset has already been mapped to data and there's a likelihood that the new data being imported may already exist in VIVO, the process becomes more complicated due to the need to match each prospective import against existing data.

7.1.1.2 **Example workflow**

The first time through an ingest process is a slow process of evaluating the source data, deciding what cleanup will be needed, and working through each step. The following example details each step in the process and how the VIVO software supports that step. As the process expands to include requirements to match against existing data, additional steps must be introduced to test for matches and perform different actions, usually as successive steps identified through different queries.

For this example, a sample set of data for people, their positions, and the organizations at which their positions reside has been provided at http://sourceforge.net/projects/vivo/files/Data%20Ingest/. The guide gives instructions on how to ingest the data, map the data to the VIVO/ISF ontology, and load the data into an RDF format. Upon completion linked data in VIVO will include people to positions and the positions to organizations.

Process:

1. Prepare CSV file
2. Create workspace models for ingesting and constructing data
3. Pull CSV file into RDF
4. Confirm data property URIs and RDF structure
5. Convert temporary RDF into VIVO/ISF ontology using SPARQL CONSTRUCT
6. Load data to the current web model

Step 1: Prepare CSV file

CSV template files can be downloaded here (or from the Source Forge links included):

- organization.csv[78] (or http://sourceforge.net/projects/vivo/files/Data%20Ingest/organization.csv/download)
- position.csv[79] (or http://sourceforge.net/projects/vivo/files/Data%20Ingest/position.csv/download)
- people.csv[80] (or http://sourceforge.net/projects/vivo/files/Data%20Ingest/people.csv/download)

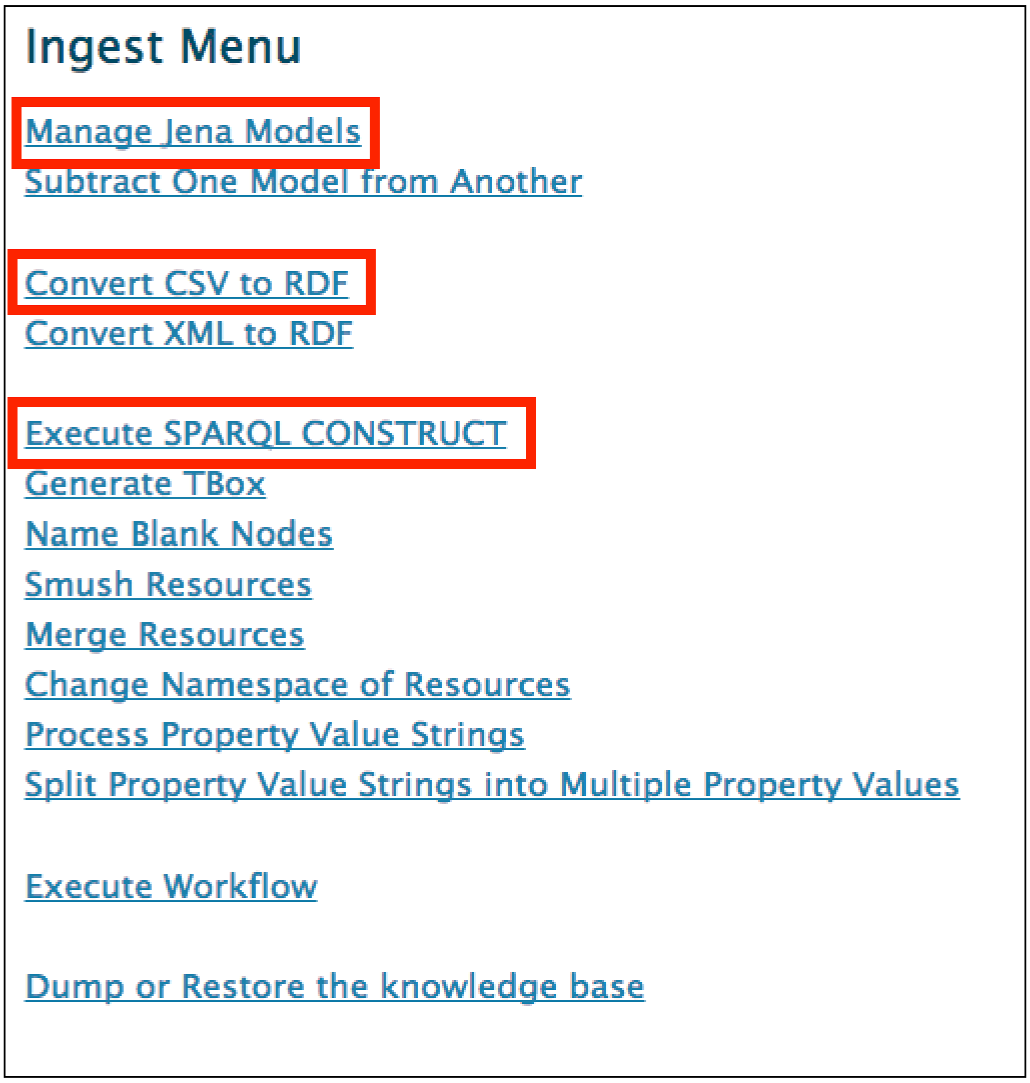
For the purposes of this walkthrough, you can leave the csv files as is if you wish.

Note: You can optionally create a local ontology and class specifically for the 'person_ID' and 'org_ID' fields included in the example CSV files. See the 'Add New Data Properties' section of the Ontology Editors Guide here: https://wiki.duraspace.org/x/2AQGAg.

Step 2: Create Workspace Models

Highlighted in red are the three Ingest Menu options we will be using for this demonstration.

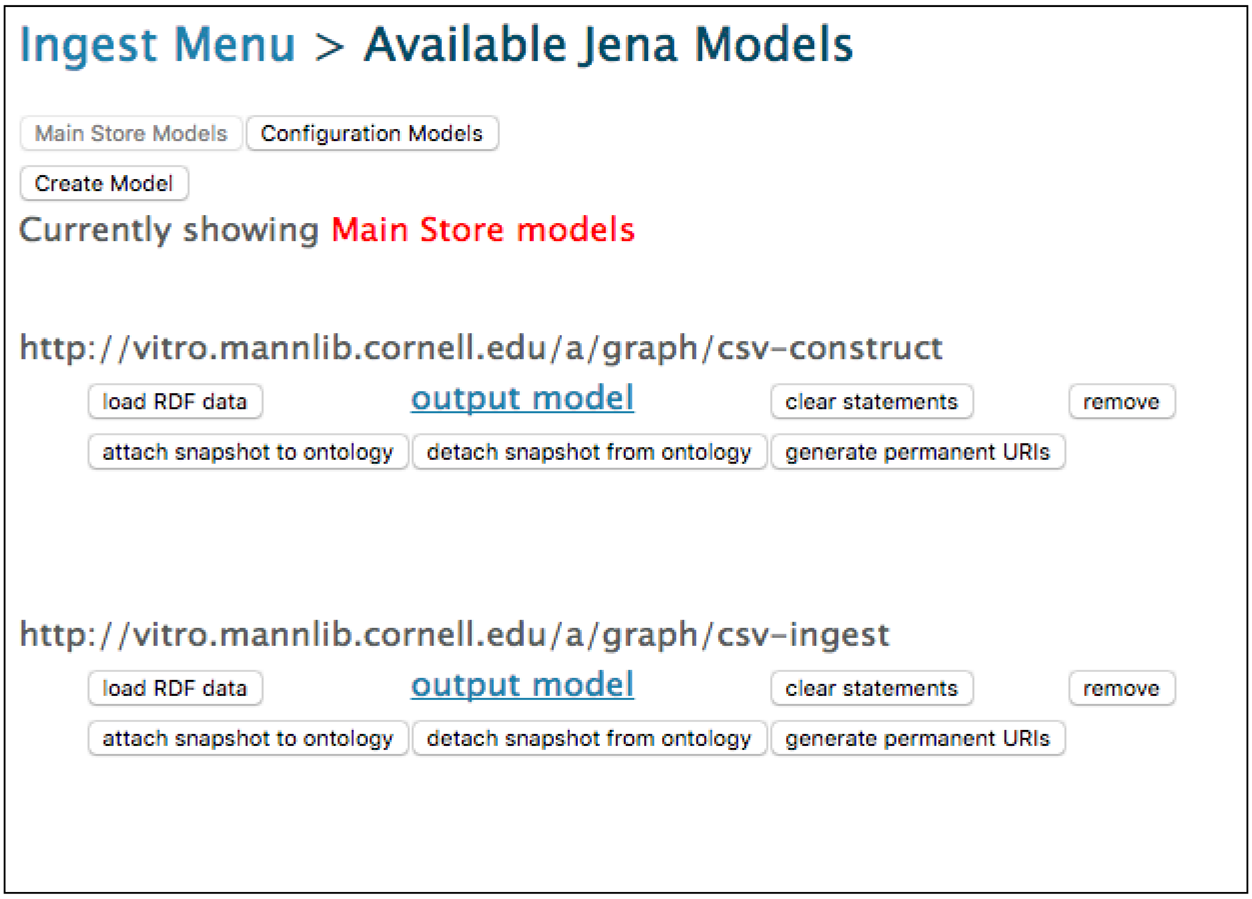
78 https://wiki.duraspace.org/download/attachments/96995768/organization.csv?api=v2&modificationDate=1522787187840&version=1
79 https://wiki.duraspace.org/download/attachments/96995768/position.csv?api=v2&modificationDate=1522787187845&version=1
80 https://wiki.duraspace.org/download/attachments/96995768/people.csv?api=v2&modificationDate=1522787187846&version=1


Ingest Menu
Manage Jena Models
Subtract One Model from Another
Convert CSV to RDF
Convert XML to RDF
Execute SPARQL CONSTRUCT
Generate TBox
Name Blank Nodes
Smush Resources
Merge Resources
Change Namespace of Resources
Process Property Value Strings
Split Property Value Strings into Multiple Property Values
Execute Workflow
Dump or Restore the knowledge base

We will create two temporary data models titled 'csv-ingest' and 'csv-construct' to keep our work separate from the main VIVO models.

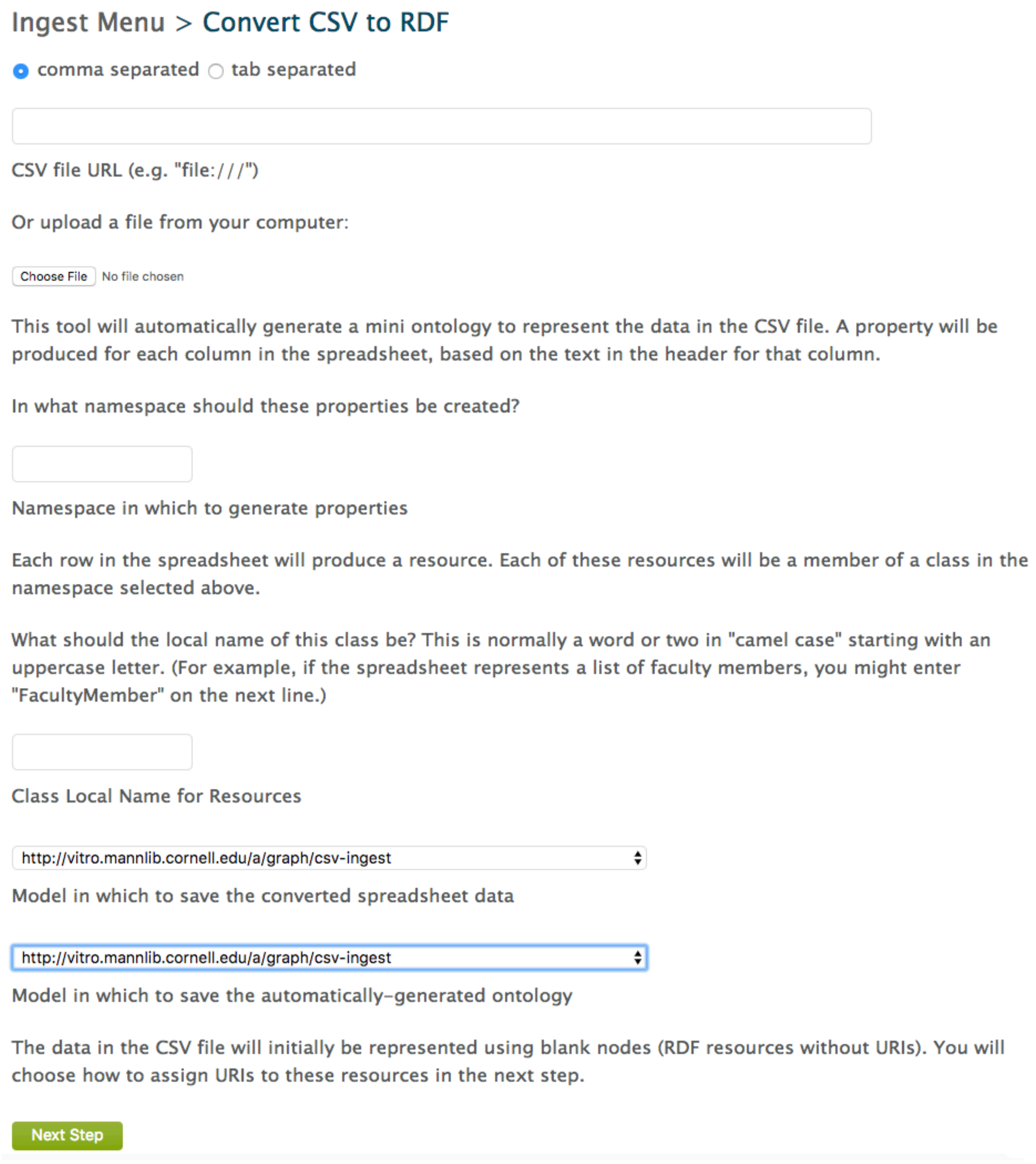
1. Select "Ingest Tools" from the Advanced Tools Menu
2. Select "Manage Jena Models"
3. Click the "Create Model" button then type in a name for your model, 'csv-ingest'.
4. Repeat step 3 and name the model 'csv-construct'

You should now see your new models on the main 'Manage Jena Models' page.


Ingest Menu > Available Jena Models
Main Store Models
Configuration Models
Create Model
Currently showing Main Store models
http://vitro.mannlib.cornell.edu/a/graph/csv-construct
load RDF data
output model
clear statements
remove
attach snapshot to ontology
detach snapshot from ontology
generate permanent URIs
http://vitro.mannlib.cornell.edu/a/graph/csv-ingest
load RDF data
output model
clear statements
remove
attach snapshot to ontology
detach snapshot from ontology
generate permanent URIs

Step 3: Pull CSV File into RDF

Now click 'Convert CSV to RDF' on the Ingest Menu. Begin by supplying a URL for your CSV file, or by uploading the file directly from your computer. Start with people.csv. Complete the fields in the form (explanation follows graphic below).


Ingest Menu > Convert CSV to RDF

[✓ comma separated] tab separated

CSV file URL (e.g. "file:///")

Or upload a file from your computer:

This tool will automatically generate a mini ontology to represent the data in the CSV file. A property will be produced for each column in the spreadsheet, based on the text in the header for that column.

In what namespace should these properties be created?

Namespace in which to generate properties

Each row in the spreadsheet will produce a resource. Each of these resources will be a member of a class in the namespace selected above.

What should the local name of this class be? This is normally a word or two in "camel case" starting with an uppercase letter. (For example, if the spreadsheet represents a list of faculty members, you might enter "FacultyMember" on the next line.)

Class Local Name for Resources

http://vitro.mannlib.cornell.edu/a/graph/csv-ingest

Model in which to save the converted spreadsheet data

http://vitro.mannlib.cornell.edu/a/graph/csv-ingest

Model in which to save the automatically-generated ontology

The data in the CSV file will initially be represented using blank nodes (RDF resources without URIs). You will choose how to assign URIs to these resources in the next step.

Namespace for Classes and Properties ('in what namespace should these properties be created?'

This namespace can be temporary since we will later map the tool's output to the VIVO/ISF ontology. For example, you can use

```
http://localhost/vivo/
```

Class Name

The class name is also a temporary value for this example. This value does not follow the created entity since you will shift the properties from the format they come in into the ontologies format. For this example, the suggested class names are “ws_ppl”, “ws_org”, and “ws_post”.

Destination Models

The data and ontology model option dropdown menus should list the model to ingest into. This is where we select one of those 'workspace' models we created earlier, "csv-ingest" for example.

After clicking convert we can check our RDF conversion through two methods, the first method being the quickest.

1. From the Ingest Menu, select the 'Manage Jena Models' and then select the ingest model's 'output model.' This is something you can open with WordPad and see the created triples for your CSV file.
2. Also from the Manage Jena Model we can attach the ingest model to the current webapp. Then we can go back to the Site Admin, select Class Hierarchy link, select 'All Classes' and navigate to the Class Name you created (individual, for example).

Click 'Next Step.' Here you will create the URI for your new individuals.

Select URI prefix

It is most convenient if this matches the URL for your VIVO instance plus '/individual/', e.g. http://vivo.university.edu/individual/.

URI suffix

This can be a random number or a created based on a pattern plus the value from your CSV file.

Now, click 'Convert CSV.' The CSV data should now be converted into temporary RDF and inserted into the 'csv-ingest' model.

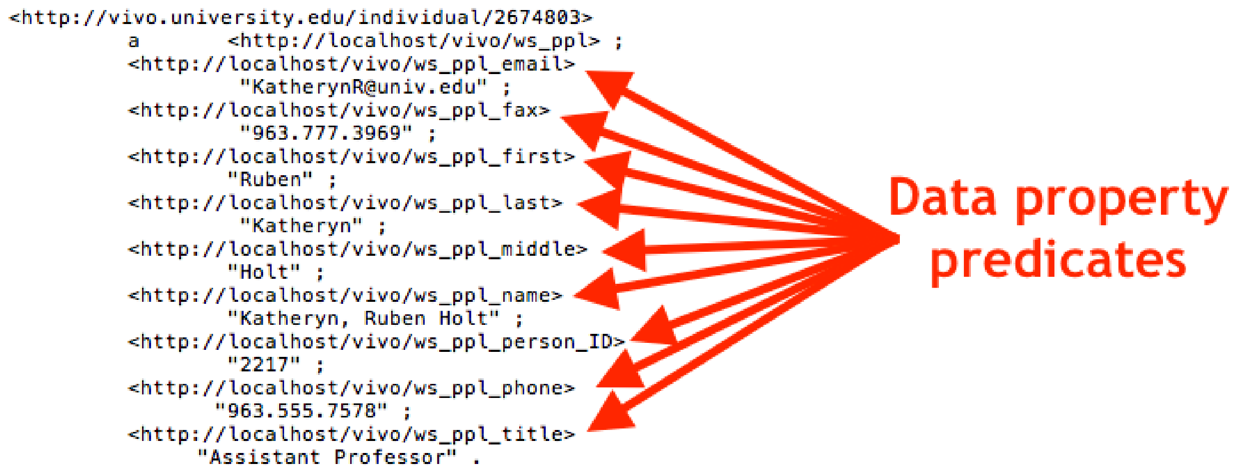
You can now repeat step 3 for both organizations.csv and positions.csv before continuing to assure positions will be attached to both people and organizations, or for simplicity, you can ignore organizations and positions for now.

Step 4: Confirm data property URIs and RDF structure

The CSV to RDF tool converts the CSV into temporary RDF that we can query using SPARQL. This temporary RDF cannot be displayed in VIVO. We will transform the RDF into the VIVO/ISF ontology format in the next step. First, take a look at the temporary RDF we have created.

Ingested Data URIs

Confirm the data property URI's that were created for each of the columns in your csv file by navigating back to the 'Manage Jena Models' page and clicking 'output model' below csv-ingest. A description of the format is described in the figure below. The predicates are the properties we need for our SPARQL Query. If you used the 'http://localhost/vivo/' (see page 71) format used above, it should look similar to this:



```
<http://vivo.university.edu/individual/2674803>
          a       <http://localhost/vivo/ws_ppl> ;
          <http://localhost/vivo/ws_ppl_email>
                   "KatherynR@univ.edu" ;
          <http://localhost/vivo/ws_ppl_fax>
                   "963.777.3969" ;
          <http://localhost/vivo/ws_ppl_first>
                  "Ruben" ;
          <http://localhost/vivo/ws_ppl_last>
                   "Katheryn" ;
          <http://localhost/vivo/ws_ppl_middle>
                  "Holt" ;
          <http://localhost/vivo/ws_ppl_name>
                  "Katheryn, Ruben Holt" ;
          <http://localhost/vivo/ws_ppl_person_ID>
                  "2217" ;
          <http://localhost/vivo/ws_ppl_phone>
                 "963.555.7578" ;
          <http://localhost/vivo/ws_ppl_title>
               "Assistant Professor" .
```

We will use these data property predicates (e.g. <http://localhost/vivo/ws_ppl_email>) in the 'WHERE' part of the next step.

Step 5: Convert temporary RDF into VIVO/ISF ontology using SPARQL CONSTRUCT

Now, we must query and CONSTRUCT new RDF that is mapped to the VIVO/ISF ontology. Diagrams to help visualize the VIVO/ISF ontology model are available on the wiki at https://wiki.duraspace.org/x/ycCdB. Some helpful links to information on SPARQL queries can be found at https://wiki.duraspace.org/x/lwUGAg. Basically, we will query the database for the triples in Step 4 to pull out the resource URIs and store them in variables (e.g. http://vivo.university.edu/individual/2674803 → ?person) in the WHERE part of the query. We then CONSTRUCT new triples that are in the proper VIVO/ISF format using those variables for the resource URIs (e.g. ?person) and data values (e.g. ?fullname).

Return to the ingest menu and click 'Execute SPARQL CONSTRUCT'
Example SPARQL query for people.csv:

```
CONSTRUCT {

?person <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/
core#FacultyMember> .
?person <http://www.w3.org/2000/01/rdf-schema#label> ?fullname .

?person <http://purl.obolibrary.org/obo/ARG_2000028> ?vcard .
?vcard <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Individual> .
?vcard <http://www.w3.org/2006/vcard/ns#hasName> ?vcard_name .
?vcard_name <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/
ns#Name> .

?vcard_name <http://www.w3.org/2006/vcard/ns#givenName> ?first .
?vcard_name <http://vivoweb.org/ontology/core#middleName> ?middle .
?vcard_name <http://www.w3.org/2006/vcard/ns#familyName> ?last .

?vcard <http://www.w3.org/2006/vcard/ns#hasEmail> ?vcard_email .
?vcard_email <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/
ns#Email> .
?vcard_email <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/
ns#Work> .
?vcard_email <http://www.w3.org/2006/vcard/ns#email> ?email .
}

WHERE {

?person <http://localhost/vivo/ws_ppl_name[81]> ?fullname .

OPTIONAL { ?person <http://localhost/vivo/ws_ppl_first[82]> ?first . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_middle[83]> ?middle . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_last[84]> ?last . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_email[85]> ?email . }
?person <http://localhost/vivo/ws_ppl_person_ID[86]> ?hrid .
BIND(URI(CONCAT(STR( ?person ), "_vcard")) AS ?vcard ) .
BIND(URI(CONCAT(STR( ?person ), "_vcardname")) AS ?vcard_name ) .
BIND(URI(CONCAT(STR( ?person ), "_vcardemail")) AS ?vcard_email ) .

}
```

Next:

- Select Source Model (‘csv-ingest’)
- Select Destination Model (‘csv-construct’)
- Select "Execute CONSTRUCT"
- Upon completion, the system will report ‘n statements CONSTRUCTed’.

Note, the BIND statements in the query allow us to create unique URIs for the associated vCard objects required in VIVO v1.6+.

81 http://vivo.coop-plus.eu/ws_ppl_name
82 http://vivo.coop-plus.eu/ws_ppl_first
83 http://vivo.coop-plus.eu/ws_ppl_middle
84 http://vivo.coop-plus.eu/ws_ppl_last
85 http://vivo.coop-plus.eu/ws_ppl_email
86 http://vivo.coop-plus.eu/ws_ppl_person_ID

Step 6: Load to Webapp

The live webapp that is indexed does not allow for models to just be attached. Attaching models works well for seeing what we have constructed or ingested or smushed, but those models are lost when Tomcat refreshes.

The final step is to output the final model and add it into the current model

1. From the Ingest Menu, select "Manage Jena Models"
2. Click "output model" below 'csv-construct'
3. Save the resulting file
4. Navigate back to the Site Administration page
5. Select Add/Remove RDF
6. Browse to the file previously saved
7. Select N3 as import format*
8. Confirmation should state 'Added RDF from file people_rdf. Added 415 statements.'

- The output engine uses N3, not RDF/XML. This is important to note when adding the 'output mode' RDF data into the webapp. RDF/XML is the default setting for the drop down list as most ontologies are written in RDF/XML.

The ingested data should now display in both the index and the search results. It is part of the main webapp and will be retained upon a Tomcat restart. The founding steps of data ingest have been completed. Repeat steps 4 and 5 for organizations and people using the SPARQL queries supplied in the appendix below.

7.1.1.3 **Appendix A: SPARQL Queries**

People Construct:

```
CONSTRUCT {

?person <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/core#FacultyMember> .
?person <http://www.w3.org/2000/01/rdf-schema#label> ?fullname .

?person <http://purl.obolibrary.org/obo/ARG_2000028> ?vcard .
?vcard <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Individual> .
?vcard <http://www.w3.org/2006/vcard/ns#hasName> ?vcard_name .
?vcard_name <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Name> .

?vcard_name <http://www.w3.org/2006/vcard/ns#givenName> ?first .
?vcard_name <http://vivoweb.org/ontology/core#middleName> ?middle .
?vcard_name <http://www.w3.org/2006/vcard/ns#familyName> ?last .

?vcard <http://www.w3.org/2006/vcard/ns#hasEmail> ?vcard_email .
?vcard_email <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Email> .
?vcard_email <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2006/vcard/ns#Work> .
?vcard_email <http://www.w3.org/2006/vcard/ns#email> ?email .
}
WHERE {
?person <http://localhost/vivo/ws_ppl_name[87]> ?fullname .
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_first[88]> ?first . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_middle[89]> ?middle . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_last[90]> ?last . }
OPTIONAL { ?person <http://localhost/vivo/ws_ppl_email[91]> ?email . }
?person <http://localhost/vivo/ws_ppl_person_ID[92]> ?hrid .
BIND(URI(CONCAT(STR( ?person ), "_vcard")) AS ?vcard ) .
BIND(URI(CONCAT(STR( ?person ), "_vcardname")) AS ?vcard_name ) .
BIND(URI(CONCAT(STR( ?person ), "_vcardemail")) AS ?vcard_email ) .
}
```

Organization Construct:

```
CONSTRUCT {
?org <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type_uri .
?org <http://localhost/vivo/ontology/vivo-local#orgID[93]> ?deptID .
?org <http://www.w3.org/2000/01/rdf-schema#label> ?name .
}

WHERE {
?org <http://localhost/vivo/ws_org_org_ID[94]> ?deptID .
?org <http://localhost/vivo/ws_org_org_name[95]> ?name .
?org <http://localhost/vivo/ws_org_org_vivo_uri[96]> ?type .
BIND(URI(?type) as ?type_uri)
}
```

Basic Position to People & Organization Construct:

Note: The example query does not account for start dates

87 http://vivo.coop-plus.eu/ws_ppl_name
88 http://vivo.coop-plus.eu/ws_ppl_first
89 http://vivo.coop-plus.eu/ws_ppl_middle
90 http://vivo.coop-plus.eu/ws_ppl_last
91 http://vivo.coop-plus.eu/ws_ppl_email
92 http://vivo.coop-plus.eu/ws_ppl_person_ID
93 http://vivo.coop-plus.eu/ontology/vivo-local#orgID
94 http://vivo.coop-plus.eu/ws_org_org_ID
95 http://vivo.coop-plus.eu/ws_org_org_name
96 http://vivo.coop-plus.eu/ws_org_org_vivo_uri

```
CONSTRUCT {

?position <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://vivoweb.org/ontology/
core#FacultyPosition> .
?position <http://www.w3.org/2000/01/rdf-schema#label> ?poslabel .
?position <http://vivoweb.org/ontology/core#relates> ?org .
?position <http://vivoweb.org/ontology/core#relates> ?person .

}
WHERE {

?org <http://localhost/vivo/ws_org_org_ID[97]> ?deptID .
?org <http://localhost/vivo/ws_org_org_name[98]> ?name .
?position <http://localhost/vivo/ws_post_department_ID[99]> ?postOrgID .
?org <http://localhost/vivo/ws_org_org_ID[100]> ?orgID .
FILTER((?postOrgID)=(?orgID))


?position <http://localhost/vivo/ws_post_department_ID[101]> ?orgID .
?position <http://localhost/vivo/ws_post_person_ID[102]> ?posthrid .
?position <http://localhost/vivo/ws_post_job_title> ?poslabel .
?person <http://localhost/vivo/ws_ppl_person_ID[103]> ?perhrid .
FILTER((?posthrid)=(?perhrid))

}
```

7.1.2 Data types for string and language

VIVO 1.10 implements RDF 1.1 and Jena 3. These changes impact the datatypes for strings and the use of the lang tag to indicate the language of the string. Please read these recommendations carefully. Jena2tools and Jena3tools will convert from previous representations to the representations recommended here.

97 http://vivo.coop-plus.eu/ws_org_org_ID
98 http://vivo.coop-plus.eu/ws_org_org_name
99 http://vivo.coop-plus.eu/ws_post_department_ID
100 http://vivo.coop-plus.eu/ws_org_org_ID
101 http://vivo.coop-plus.eu/ws_post_department_ID
102 http://vivo.coop-plus.eu/ws_post_person_ID
103 http://vivo.coop-plus.eu/ws_ppl_person_ID

7.1.2.1 Literal Values

Jena 3 improves Jena's RDF 1.1 compatibility. Specifically, literal values are always stored internally with datatypes. "Untyped" string literals are the same as the identical value typed as xsd:string. See the following document for more information

> https://jena.apache.org/documentation/migrate_jena2_jena3.html

For VIVO, this means that the two triples:

```
<subj> <pred> "value"
<subj> <pred> "value"^^xsd:string
```

will be treated as the same triple and only stored once in your triple store. As a result, when using the procedure described below to upgrade your triple store from an earlier version of VIVO, you may find that the number of triples in your triple store after the upgrade is lower than the number before the upgrade.

Any code, tools, parsers, utilities, or queries that expect to differentiate between these two triples will produce results different than produced previously – RDF 1.1 no longer distinguishes between these two triples. In particular, queries that limit results based on the xsd:string datatype will likely produce larger result sets, as previously untyped triples are now typed as xsd:string internally.

Another way of saying this – any triple loaded into VIVO or Vitro that does not have a type will be typed as xsd:string internally.

Exports from VIVO and Vitro will never include the xsd:string datatype. Literal values that do not have explicit types are always assumed to be xsd:string.

As a result, we recommend that input process for VIVO do not include xsd:string datatypes on literals. While they may be correct, and will result in the literal value being typed as xsd:string internally, export processes will not include the xsd:string on output.

In RDF 1.0, a type could not be asserted with a language identifier. In RDF 1.1, a type can be asserted with a language identifier. Untyped input with language identifiers were left as untyped internally in RDF 1.0. In RDF 1.1, untyped input with language identifiers are assumed to have type rdf:langString. Exports from VIVO for triples with language tags will never include the rdf:langString datatype. Literal values with language tags are always assumed to be rdf:langString.

As a result, we recommend that input process for VIVO do not include datatypes on triples with language types. While asserting rdf:langString is correct, and will result in the literal value being typed as rdf:langString internally, export processes will not include the rdf:langString on output.

Code, tools, parsers, utilities based on RDF 1.0 should not be used with Vitro and VIVO 1.10.x All code, tools, parsers, and utilities must support RDF 1.1.

On start-up of version 1.10.x, the triple store is checked to insure that it has been upgraded. If untyped literals are found in the triple store, an error message will appear in the browser and the application will not start. The test applies only to the content store. It is possible that your content store could pass this test, but your configuration triple store remains incompatible with Jena 3 and RDF 1.1. In such a case, your application may become unstable.

7.1.2.2 Recommendations

1. String literals should be untyped in RDF input to VIVO. Use "xxx" rather than "xxx"^^xsd:string
2. Lang tags should be used with untyped string literals in RDF input to VIVO. Use "xxx"@fr-CA rather than "xxx"@fr-CA^^xsd:langString

3. Lang tags should be used on all strings which might render differently in different languages. Use "United States"@en-US. For strings which are not rendered differently in different languages use a simple untyped string literal. For example "0000-0001-2345-321X"

7.2 Exporting Data from VIVO

7.2.1 Exporting All Data

To export all data data from VIVO, use jena3tools, a command line utility provided with VIVO. jena3tools can export the VIVO content and configuration into files in a format of your choosing. See http://github.com/vivo-project/jenatools for more information.

7.2.2 Exporting Selected Data

To export selected data from VIVO, use a SPARQL query. See SPARQL Queries (see page 91). SPARQL can export data in a variety of formats, including CSV, JSON, and RDF/XML.

7.3 Managing Person Identifiers

VIVO provides several means for specifying various identifiers with people. Sign on to VIVO as an editor and open the Identity tab. You will see various identifiers supported by VIVO.

The table below lists the identity options provided by VIVO. Additional identifiers can be added by Extending the VIVO ontology[104]. See the Notes below the table for additional information regarding identity and identifiers in VIVO.

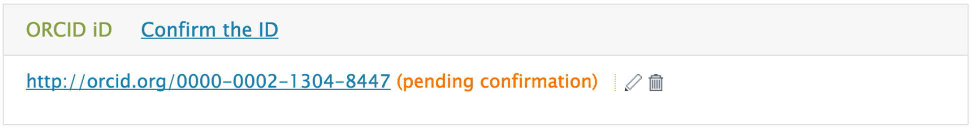
Identifier	**Description**	**Data or Object**	**Predicate**	**Creates External Link**
sameAs	owl sameAs assertion. See http://www.w3.org/TR/owl-ref/#sameAs-def	Object	owl:sameAs	No
ORCID iD	The Open Researcher and Contributor ID See http://orcid.org	Object	vivo:orcidId	Yes
eRA Commons ID	The US NIH Electronic Research Administration Commons ID. See https://commons.era.nih.gov/	Data	vivo:eRACommonsId	No
ISI Researcher ID	Thomson Reuters Research ID See http://www.researcherid.com/	Data	vivo:researcherId	No

104 https://wiki.duraspace.org/display/VIVO/Extending+the+VIVO+ontology

Identifier	Description	Data or Object	Predicate	Creates External Link
Scopus ID	Elsevier Scopus Author Identifier. See http://help.scopus.com/Content/h_autsrch_intro.htm	Data	vivo:scopusId	Yes
Health Care Provider ID	Generic field for holding a health care provider ID of the institution's choice.	Data	obo:ARG_0000197	No

7.3.1 Notes

1. Many identifiers are configured by default to not be publicly displayed. To change the display level of an identifier, go to Site Admin > Property Groups, select the identifier in the identity section and then click Edit Property Record. Alternatively, make the change in initialTBoxAnnotations.n3.
2. sameAs can be configured as supported or unsupported in VIVO. See VIVO v1.6 release planning[105]
3. ORCID can be configured for integration with the ORCID. See Activating the ORCID integration (see page 272)
    a. To add ORCID identifiers using RDF, assert the triple associating the personURI with orcidURI: <personURI> <vivo:orcidId> <orcidURI>, where orcidURI looks like http://orcid.org/xxxx-xxxx-xxxx-xxxx
    b. Add a second triple to assert that the orcidURI is a thing: <orcidURI> a owl:Thing .
    c. When these two triples are added for a person, the VIVO interface will report the ORCID as unconfirmed.


ORCID iD Confirm the ID

http://orcid.org/0000-0002-1304-8447 (pending confirmation)

    d. The user can logon to VIVO, select "Confirm the ID" and enter their ORCID password. The ORCID iD will then be confirmed in VIVO.
    e. Or you can confirm the ORCID by adding another triple to VIVO: <orcidURI> vivo:confirmedOrcidId <personURI>
4. For a SCOPUS ID, VIVO provides a link to SCOPUS at http://www.scopus.com/authid/detail.url?authorId=nnnnnnnn

7.4 Managing Organization Hierarchy

105 https://wiki.duraspace.org/display/VIVO/VIVO+v1.6+release+planning#VIVOv1.6releaseplanning-sameAs

7.4.1 Overview

VIVO can be used to mange the hierarchical structure, or organizational chart, of any organization. We describe how VIVO represents organizational structure, and how to create data that can be loaded into VIVO to represent the structure of an organization. The Sample Data (see page 51) has a sample university, with sample colleges and departments, illustrating the techniques described here.

7.4.2 "hasPart, partOf"

VIVO uses the Basic Formal Ontology "hasPart" to indicate that organization A "hasPart" organization B. Sample University has a College of Science. We say College of Science "partOf" Sample University. VIVO will automatically assert Sample University "hasPart" College of Science.

The Basic Formal Ontology has been implemented as part of the Open Biomedical Ontologies (OBO). They have coded "hasPart" as "BFO_0000051" and "partOf" is "BFO_0000050" VIVO uses the OBO representation which can be found here: http://purl.obolibrary.org/obo/

7.4.3 Your Organizational Data

To manage organizations in VIVO, you will need to create data representing your organizations and your organizational structure. This is easy to do using a spreadsheet. Simply create one row for each organization in your institution. For example, the table below shows two colleges as part of Sample University, and each college has one or more departments. The Department of Chemistry has a division – Organic Chemistry.

Your URI will contain your domain name. In this example we have structured the URI. VIVO URI always start with the domain, followed by "/individual/" followed by a URI of your choice. Here we have used a tag "org" to indicate URIs for organizations. This will make the URI easier to find. We then use the name of the organization, with each word capitalized. URI must be unique. If two organizations within your institution have the same name, you will need to make unique URI for each – you might add a "1" to the end of one of the URI, for example.

Add as many rows as it takes to represent the organizations within your institution. You can start with some, load them to VIVO (see below), and create more rows and reload, eventually building up a complete set of organizations.

OrgURI	**Type**	**Name**	**PartOfURI**
http://vivo.mydomain.edu/individual/orgSampleUniversity	university	Sample University	
http://vivo.mydomain.edu/individual/orgCollegeOfScience	college	College of Science	http://vivo.mydomain.edu/individual/orgSampleUniversity
http://vivo.mydomain.edu/individual/orgPhysics	department	Physics	http://vivo.mydomain.edu/individual/orgCollegeOfScience
http://vivo.mydomain.edu/individual/orgChemistry	department	Chemistry	http://vivo.mydomain.edu/individual/orgCollegeOfScience
http://vivo.mydomain.edu/individual/orgOrganicChemistry	division	Organic Chemistry	http://vivo.mydomain.edu/individual/orgChemistry

OrgURI	Type	Name	PartOfURI
http://vivo.mydomain.edu/individual/orgCollegeOfTheArts	college	College of the Arts	http://vivo.mydomain.edu/individual/orgSampleUniversity
http://vivo.mydomain.edu/individual/orgTheaterAndDance	department	Theater and Dance	http://vivo.mydomain.edu/individual/orgCollegeOfTheArt

7.4.4 Making Triples

To load data into VIVO, you will make triples out of your organizational data. You can use tools such as Karma, SAS, python, the VIVO Pump, VIVO Harvester, XSLT, your own scripts, or a text editor.

To load your organizational data into VIVO, you will transform your spreadsheet into triples. Each row of the spreadsheet will result in three triples: 1) a triple that asserts that the OrgURI is an organization of a particular type; 2) a triple that asserts that the organization has a particular name; and 3) a triple that asserts that the organization is part of another organization. Notice that the first row in the spreadsheet will generate two triples – Sample University is not part of another organization. The other six rows will generate 3 triples each. We will have 20 triples in all 2 + 3 * 6 = 20. The triples are shown below. Notes follow.

Sample Triples

```
<http://vivo.mydomain.edu/individual/orgSampleUniversity> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#University> .
<http://vivo.mydomain.edu/individual/orgSampleUniversity> <http://www.w3.org/2000/01/rdf-schema#label>
"Sample University"@en-US .
<http://vivo.mydomain.edu/individual/orgCollegeOfScience> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#College> .
<http://vivo.mydomain.edu/individual/orgCollegeOfScience> <http://www.w3.org/2000/01/rdf-schema#label>
"College of Science"@en-US .
<http://vivo.mydomain.edu/individual/orgCollegeOfScience> <http://purl.obolibrary.org/obo/BFO_0000050>
<http://vivo.mydomain.edu/individual/orgSampleUniversity> .
<http://vivo.mydomain.edu/individual/orgPhysics> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://
vivoweb.org/ontology/core#Department> .
<http://vivo.mydomain.edu/individual/orgPhysics> <http://www.w3.org/2000/01/rdf-schema#label> "Physics"@en-
US .
<http://vivo.mydomain.edu/individual/orgPhysics> <http://purl.obolibrary.org/obo/BFO_0000050>  <http://
vivo.mydomain.edu/individual/orgCollegeOfScience> .
<http://vivo.mydomain.edu/individual/orgChemistry> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#Department> .
<http://vivo.mydomain.edu/individual/orgChemistry> <http://www.w3.org/2000/01/rdf-schema#label>
"Chemistry"@en-US .
<http://vivo.mydomain.edu/individual/orgChemistry> <http://purl.obolibrary.org/obo/BFO_0000050>  <http://
vivo.mydomain.edu/individual/orgCollegeOfScience> .
<http://vivo.mydomain.edu/individual/orgOrganicChemistry> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#Division> .
<http://vivo.mydomain.edu/individual/orgOrganicChemistry> <http://www.w3.org/2000/01/rdf-schema#label>
"Organic Chemistry"@en-US .
<http://vivo.mydomain.edu/individual/orgOrganicChemistry> <http://purl.obolibrary.org/obo/BFO_0000050>
<http://vivo.mydomain.edu/individual/orgChemistry> .
```

```
<http://vivo.mydomain.edu/individual/orgCollegeOfTheArts> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#College> .
<http://vivo.mydomain.edu/individual/orgCollegeOfTheArts> <http://www.w3.org/2000/01/rdf-schema#label>
"College of the Arts"@en-US .
<http://vivo.mydomain.edu/individual/orgCollegeOfTheArts> <http://purl.obolibrary.org/obo/BFO_0000050>
<http://vivo.mydomain.edu/individual/orgSampleUniversity> .
<http://vivo.mydomain.edu/individual/orgTheaterAndDance> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#Department> .
<http://vivo.mydomain.edu/individual/orgTheaterAndDance> <http://www.w3.org/2000/01/rdf-schema#label>
"Theater and Dance"@en-US .
<http://vivo.mydomain.edu/individual/orgTheaterAndDance> <http://purl.obolibrary.org/obo/BFO_0000050>
<http://vivo.mydomain.edu/individual/orgCollegeOfTheArts> .
```

7.4.4.1 Notes regarding the triples

1. The triples are shown in N-triples format[106] and should be stored in a file with the file type ".nt" Elements of the triples are either URI (in brackets) or strings (in double quotes with a language tag). Each triple ends with a period.
2. Each triple is of the form subject predicate object. The subject is always a URI (in brackets) and is a URI you specify in your table of data.
3. The predicates are from the ontologies VIVO uses to represent data. They are also always URI (in brackets). There are three kinds of assertions being made. Each has its own predicate:
   a. The predicate to assert a type is <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   b. The predicate to assert a name, known as a label, is <http://www.w3.org/2000/01/rdf-schema#label>
   c. The predicate to assert that the organization is part of another organization is <http://purl.obolibrary.org/obo/BFO_0000050>

7.4.5 Managing the Triples in VIVO

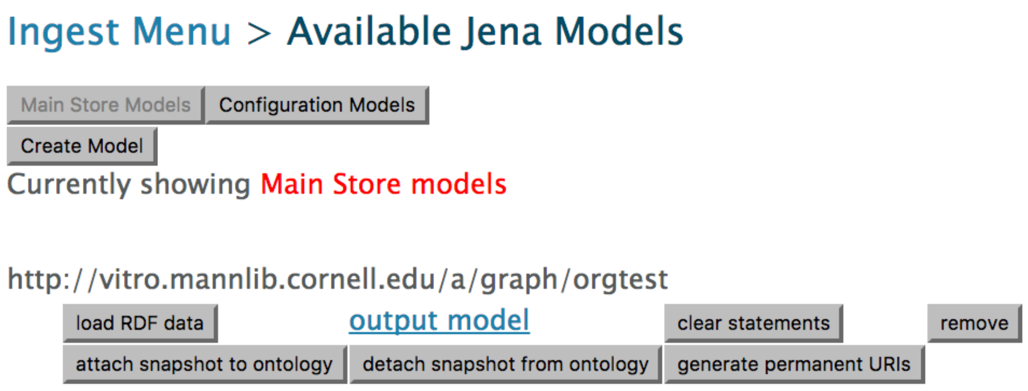
Now that we have triples representing our organizations and their organization hierarchy, its easy to load them into VIVO and update them as necessary. We will load the triples into a "named graph." This is just a collection of triples with a name. We will use a named graph to make it easy to update – we'll just empty the named graph and reload it. That way we can be sure that the data in the file of triples is the data in the named graph.

7.4.5.1 Add the triples to VIVO

1. Sign on VIVO as a system administrator.
2. Navigate to Site Admin > Advanced Data Tools > Ingest Tools > Manage Jena Models
3. At the top, click on the button "Create Model"
4. Enter the name of your model. Your name must be valid as part of a URI (no spaces). Your might call your model "orgtest"

106 https://en.wikipedia.org/wiki/N-Triples

5. Vitro creates a model with your name. Find it in the list of models. You should see:



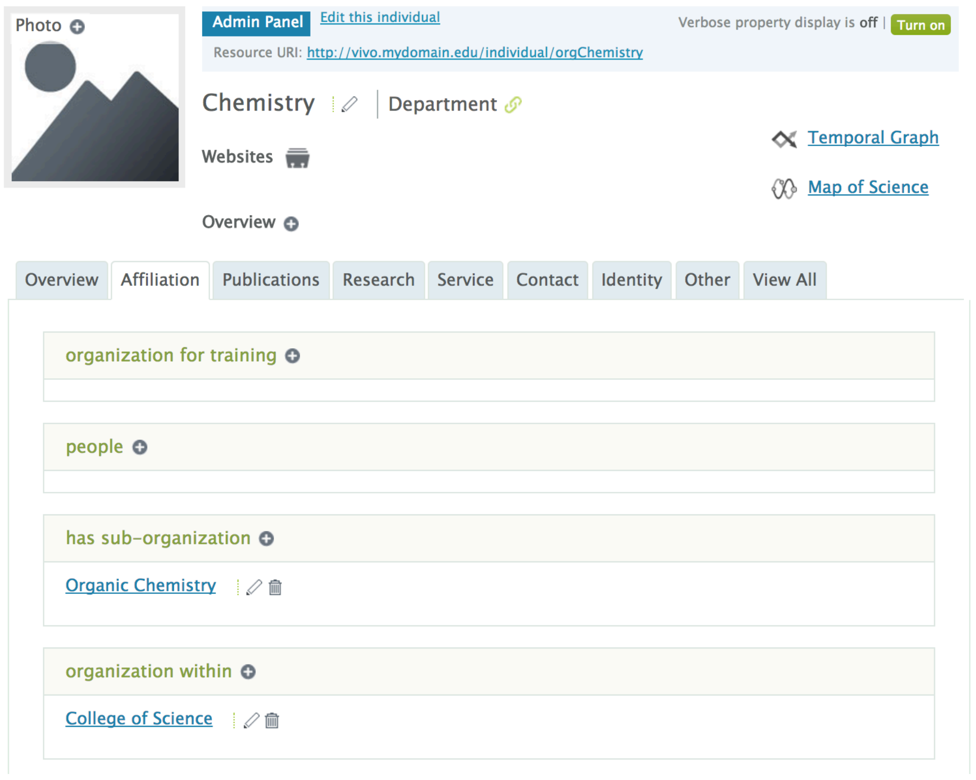
Ingest Menu > Available Jena Models

Currently showing Main Store models

http://vitro.mannlib.cornell.edu/a/graph/orgtest

6. Click "load RDF data"
7. Click Choose file and select your file of triples
8. Click on the selector to indicate that your file is in "N-triples" format
9. Click Load Data

You're done! You have your organizations and your organizational hierarchy data in VIVO. You can navigate to one of your organizations and see the hierarchical information:



Resource URI: http://vivo.mydomain.edu/individual/orgChemistry

Verbose property display is off

Chemistry | Department

organization for training

people

has sub-organization

Organic Chemistry

organization within

College of Science

7.4.5.2 Updating your triples in VIVO

When you have organizational data to update – a new organization has been added, an organizational change has been made, an organization has a new name, you discovered an error in your data, or for any other reason, update the file with your triples and follow the steps below to replace your organizational data in VIVO with the organizational data in your file.

1. Sign on VIVO as a system administrator.
2. Navigate to Site Admin > Advanced Data Tools > Ingest Tools > Manage Jena Models
3. Find your organization named graph it in the list of models.
4. Carefully click "clear statements" for your organization named graph. Be careful not to clear the statements of any other models.
5. Click "Load RDF data"
6. Click Choose file and select your file of triples
7. Click on the selector to indicate that your file is in "N-triples" format
8. Click Load Data

Your previous organization data has been replaced with your new organizational data.

7.4.6 Some Closing Observations

The basic technique for data management described here – creating triples, loading them into a named graph, and updating the graph when data changes – can be used to manage any of your VIVO data. You can put people in one graph, publications in another, grants in another, datasets in another, and manage each by creating triples and updating graphs. You may wish to create repeatable processes for each of the kinds of data you are managing. These processes should be based on tools of your choice – Karma, XSLT, or scripts you write.

The Ontology Reference (see page 310) provides details regarding how VIVO represents entities. Practice will lead to familiarity and the VIVO community is always happy to help with any questions you may have.

7.5 Managing Data Packages

7.5.1 Overview

Data packages are sets of data represented using VIVO RDF in one of the supported VIVO RDF file formats – Turtle (.ttl), Triples (.nt), Notation3 (.n3) or RDF-XML (.rdf) Data packages are typically produced as semi-static – they can be loaded into VIVO and updated as needed. Data packages typically deliver statements about entities outside the management of the particular VIVO.

VIVO manages data packages by creating a new graph for each package, containing the asserted triples for the data package and named with the name of the data package file. The VIVO inferencer creates inferred triples for the data packages and stored the inferred triples in the inference graph. When changes are made to the data package file, the VIVO inferencer must be run to bring the inference graph up to date with the changes made in the asserted graph for the data package.

An example of a data package would be the Grid data, representing the research organizations of the world. This data set, maintained by Digital Science, contains more than 65,000 university, research institutes, funding agencies and other organizations involved in research across the world. The data set contains the official name and alternate names as well as abbreviations of names of the each organization, its geographic location, its type, date of founding, parent, child and affiliated organizations, as well as persistent identifiers for the organization. The data is available as a data package for VIVO at https://github.com/openvivo/grid-rdf

7.5.2 Add a data package

To add a data package to VIVO,

1. Place a copy of the data package file in `vivo/home/rdf/abox/filegraph`
2. Restart Tomcat. VIVO will add a new graph to the triple store containing the asserted triples in the data package file. See Graph Reference (see page 305) for additional detail. The VIVO inferencer will infer additional triples regarding the data package and add those triples to the vitro-kb-info graph. Again see Graph Reference (see page 305). Note: the inferencer may take quite awhile to complete. Adding a package with tens of thousands of new entities, each with dozens of attributes may take hours to reinference.

7.5.3 Update a data package

To update a data package:

1. Place a copy of the updated data package file in `vivo/home/rdf/abox/filegraph`
2. Restart Tomcat. VIVO will compare the contents of the triple store with the contents of the data package file, and update the triples in the associate graph as needed. VIVO will then reinference the triple store. Note: the inferencer may take quite awhile to complete.

7.5.4 Delete a data package

To delete a data package:

1. Remove the data package file from `vivo/home/rdf/abox/filegraph`
2. Restart Tomcat. VIVO will detect that the file is no longer present and remove the associated graph. The inferencer will be run and triples in the inference graph associated with the deleted data package file will be removed from the inference graph.

7.5.5 Available Data Packages

Data packages are available at the following locations:

1. Research organizations of the world. From http://gid.ac Available as CC-0 data. https://github.com/openvivo/grid-rdf
2. Journals of the world. Compiled from CrossRef and NIH PubMed. More than 40K journals, each with title and ISSN. https://github.com/OpenVIVO/OpenVIVOjournals
3. Dates. Dates with simple URI, known URI. Avoid creating multiple date entities for the same date. Link all references to a date to a single date entity. https://github.com/OpenVIVO/date-rdf
4. Cities of the United States. Data for all cities in the United States with population 100K or more. Includes lat/long. https://github.com/mconlon17/vivo-add-cities

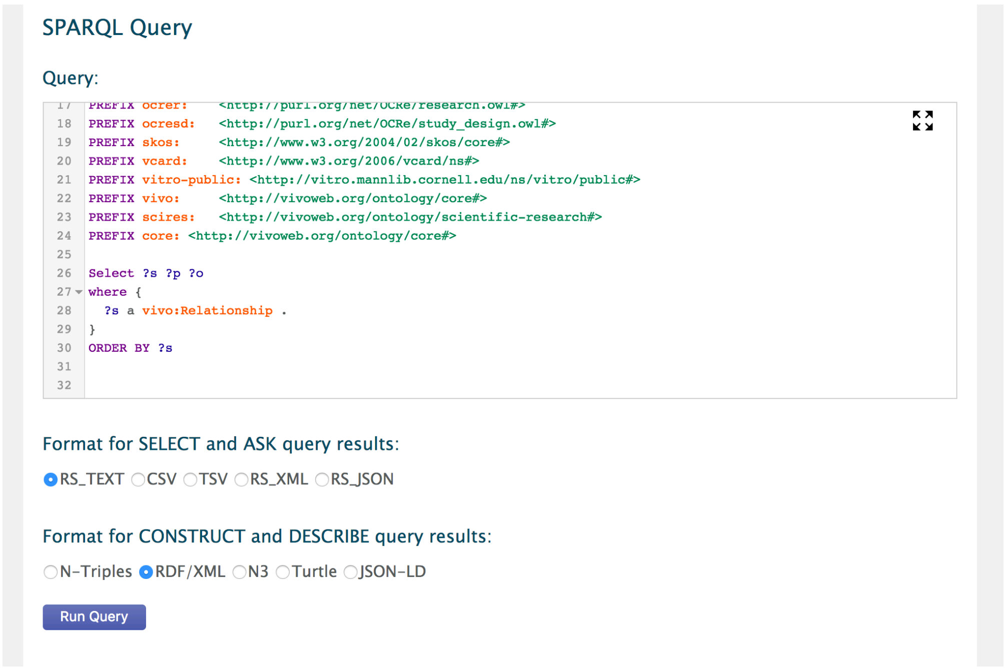
7.6 SPARQL Queries

7.6.1 Overview

SPARQL is a query language for RDF-based systems such as Vitro and VIVO. Using SPARQL one can extract any information from VIVO or Vitro, producing reports, or providing data for other software such as visualizations or user interfaces.

7.6.2 Running SPARQL queries

To run a SPARQL query, navigate to Site Admin > Sparql query. You will see a page similar to the one below. Note that the text is in various colors. VIVO uses the YASQE editor for SPARQL. You can read more about its features at their web site. See http://yasqe.yasgui.org/ At the top of the Query window are SPARQL prefix declarations for VIVO. These provides abbreviations for various URLs you use in your SPARQL queries. For more on SPARQL, see Learning SPARQL. http://learningsparql.com   Below the prefix declarations is the actual SPARQL query.



SPARQL Query

Query:

```
PREFIX ocrer:    <http://purl.org/net/OCRe/research.owl#>
PREFIX ocresd:   <http://purl.org/net/OCRe/study_design.owl#>
PREFIX skos:     <http://www.w3.org/2004/02/skos/core#>
PREFIX vcard:    <http://www.w3.org/2006/vcard/ns#>
PREFIX vitro-public: <http://vitro.mannlib.cornell.edu/ns/vitro/public#>
PREFIX vivo:     <http://vivoweb.org/ontology/core#>
PREFIX scires:   <http://vivoweb.org/ontology/scientific-research#>
PREFIX core: <http://vivoweb.org/ontology/core#>

Select ?s ?p ?o
where {
  ?s a vivo:Relationship .
}
ORDER BY ?s


```

Format for SELECT and ASK query results:

[✓ RS_TEXT] CSV TSV RS_XML RS_JSON

Format for CONSTRUCT and DESCRIBE query results:

N-Triples [✓ RDF/XML] N3 Turtle JSON-LD

The query in the figure above asks for a sorted list of the entities in VIVO that have type Relationship. RS_TEXT has been selected as an output format. This will display in a browser window. Note that you can select CSV, or TSV. These will download to your computer. You may also select RS_XML and RS_JSON – these will also display in your browser window.

Running the above Query in OpenVIVO generates over 10,000 rows of output. The beginning of the output in JSON format is shown below:

First few lines of JSON output for preceding query

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.4225/03/58ca600d726bd-authorship1" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.4225/03/58ca600d726bd-authorship2" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.6084/m9.figshare.2002020-
authorship1" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.6084/m9.figshare.2002200-
authorship1" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://openvivo.org/a/doi10.6084/m9.figshare.2002200-
authorship2" }
      } ,
```

7.6.3 Using SPARQL for reporting

SPRQL can be used to extract data from VIVO for reporting. Using the TSV (tab separated values) output format, the results of a SPARQL query can be downloaded from VIVO and uploaded to a spreadsheet, reporting or presentation tool.

The query below makes a contact list for all people in particular academic unit.

```
#
# Find all the people with a position in the CTSI or any CTSI sub-unit,
# and list them alphabetically with phone, email, gatorlink, eracommons if any
#
SELECT ?person (MIN(DISTINCT ?xname) AS ?name)
    (MIN(DISTINCT ?xphone) AS ?phone)
    (MIN(DISTINCT ?xemail) AS ?email)
    (MIN(DISTINCT ?xgatorlink) AS ?gatorlink)
    (MIN(DISTINCT ?xeracommons) AS ?eracommons)
WHERE {
  {?pos vivo:relates <http://vivo.ufl.edu/individual/n8763427> . ?pos a vivo:Position .}
  UNION
  {<http://vivo.ufl.edu/individual/n8763427> obo:BFO_0000051 ?sub .
```

```
   ?pos vivo:relates ?sub . ?pos a vivo:Position .}
  ?pos vivo:dateTimeInterval ?dt .
  OPTIONAL {?dt vivo:end ?end . }
  FILTER (!BOUND(?end))  # current positions do not have end dates
  ?pos vivo:relates ?person . ?person a foaf:Person .
  ?person rdfs:label ?xname .
  ?person a ufVivo:UFCurrentEntity .
  ?person obo:ARG_2000028 ?vcard .
  OPTIONAL { ?vcard vcard:hasEmail ?email_thing . ?email_thing vcard:email  ?xemail .}
  OPTIONAL { ?vcard vcard:hasTelephone ?tel_thing . ?tel_thing vcard:telephone ?xphone .}
  OPTIONAL { ?person ufVivo:gatorlink ?xgatorlink .}
  OPTIONAL { ?person vivo:eRACommonsId ?xeracommons .}
}
GROUP BY ?person
ORDER BY ?name
```

For additional examples, some much simpler than the example above, see Mike Conlon[107]'s web site http://mconlon17.github.io/sparql

7.6.4 Using SPARQL to clean data

Using SPARQL queries, one can find triples meeting a criteria for improvement. You might query for people without labels, for example.

CONSTRUCT statements can be used to make triples that you would like to remove from your VIVO. Run the query to make the triples, download them, then use System Admin > Add/Remove RDF data to Remove the triples you have constructed. In a similar manner, you can construct improved triples and add them to you VIVO.

7.6.5 DESCRIBE queries

Vitro SPARQL supports DESCRIBE queries, but DESCRIBE is not well-defined by W3C standards, allowing implementation specific variations. In Vitro, a request to DESCRIBE a URL will return all the triples with that URL as the subject. So, for example:

A sample DESCRIBE query

```
DESCRIBE <http://openvivo.org/a/doi10.4225/03/58ca600d726bd>
```

returns

```
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.obolibrary.org/obo/ARG_2000028> <http://
openvivo.org/a/doi10.4225/03/58ca600d726bd-vcard> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/core#relatedBy> <http://
openvivo.org/a/doi10.4225/03/58ca600d726bd-authorship2> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://purl.obolibrary.org/obo/BFO_0000031> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.obolibrary.org/obo/RO_0002353> <http://
openvivo.org/a/eventVIVO2017> .
```

107 https://wiki.duraspace.org/display/~mconlon

```
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/abstract> "<div>This is
the pre-print version of a paper accepted in Open Repository Conference in Brisbane, Australia, June
2017.<b><br></b></div><div><br></div><b>Abstract\u00A0</b><div><b><br></b><div>Research Graph is an open
collaborative project that builds the capability for connecting researchers, publications, research grants
and research datasets (data in research). \u00A0VIVO is an open source, semantic web platform and a set of
ontologies for representing scholarship. \u00A0To provide interoperability between Research Graph data and
VIVO systems we modelled the Research Graph metamodel using the VIVO Integrated Semantic Framework. To
evaluate the mapping, we used the model to connect figshare RDF records to data collections in Research
Data Australia using Research Graph API. In addition, we are working toward loading Research Graph data
into a VIVO instance. \u00A0VIVO provides a search capability, and pages for first class entities in the
Research Graph model -- researcher, dataset, grant, and publication. \u00A0The result provides a
visualisation solution for co-authors, co-funding, timeline, and a capability map for finding expertise
related to concepts of interest. \u00A0The resulting linked open data will be made freely available and can
be used in other tools for additional discovery.<br></div></div>" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/doi>
"10.4225/03/58ca600d726bd" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/core#datePublished>
<http://openvivo.org/a/date2017-03-16> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/core#relatedBy> <http://
openvivo.org/a/doi10.4225/03/58ca600d726bd-authorship1> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/freetextKeyword>
"Research Discovery" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2002/07/owl#Thing> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/core#dateCreated> <http://
openvivo.org/a/date2017-03-16> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://purl.org/ontology/bibo/Document> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/freetextKeyword> "Linked
Open Data" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vitro.mannlib.cornell.edu/ns/vitro/
0.7#mostSpecificType> <http://vivoweb.org/ontology/core#ConferencePaper> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://purl.obolibrary.org/obo/BFO_0000001> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/2000/01/rdf-schema#label> "Creating
an open linked data model for Research Graph using VIVO Ontology" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://purl.org/ontology/bibo/freetextKeyword> "Open
research outputs" .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://purl.obolibrary.org/obo/IAO_0000030> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://purl.obolibrary.org/obo/BFO_0000002> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/core#dateTimeValue>
<http://openvivo.org/a/date2017-03-16> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://vivoweb.org/ontology/core#dateModified>
<http://openvivo.org/a/date2017-03-16> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#ConferencePaper> .
<http://openvivo.org/a/doi10.4225/03/58ca600d726bd> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://purl.org/ontology/bibo/Article> .
```

7.6.6 ASK Queries

Vitro SPARQL supports ASK queries which return either true (there are triples that satisfy the pattern), or false (there are no triples that satisfy the pattern). The query below will return true in most VIVOs and false in a new VIVO.

ASK Query

```
ASK { ?s a vivo:Relationship . }
```

7.6.7 Additional SPARQL Resources

- YASQUE, "YASQUE Home Page," web site. Last Accessed June 17, 2017. http://yasqe.yasgui.org/
- Conlon, M. "Sample SPARQL: SPARQL scripts for getting information from your VIVO," web site. Last Accessed June 17, 2017. http://mconlon17.github.io/sparql/
- DuCharme, B. "Learning SPARQL," Wiley Publishing, 2011. 235 pages. http://www.learningsparql.com/
- Apache Jena, "SPARQL Tutorial," web site. Last accessed June 17, 2017. https://jena.apache.org/tutorials/sparql.html

7.7 How to remove data from a specific graph

Using the Ingest Tools,

1. Go to Manage Jena Models and add a new, temporary model.
2. Use the "load RDF data" button below it to add to this temporary model the RDF you ultimately want to delete.
3. Go to back to the Ingest Tools menu, select Subtract One Model from Another.
4. Set "model to be subtracted from" and "model in which difference should be saved" to the graph you wanted to delete from in the first place. Set "model to subtract" to the temporary graph you just created.
5. Run the subtraction.
6. Go back to Manage Jena Models and remove the temporary model.

7.8 Removing Entities from VIVO

- Journal (see page 98)

7.8.1 General Method

To remove entities from VIVO, run SPARQL queries to retrieve the triples for the entities as RDF. Then go to Site Administration -> Advanced Data Tools -> Add or Remove RDF Data to upload the RDF to remove the triples for the entities.

Entities that are involved in relationships will need more attention. The relationship involving the entity should also be removed.

7.8.2 Examples

7.8.2.1 Remove publications by type

Run the following SPARQL CONSTRUCT queries to retrieve the triples associated with the entities:

Article

```
construct {
             ?s ?p ?o .
} where {
     ?s rdf:type bibo:Article .
     ?s ?p ?o  .
}
```

Book

```
construct {
             ?s ?p ?o .
} where {
     ?s rdf:type bibo:Book .
     ?s ?p ?o  .
}
```

Case Study

```
construct {
             ?s ?p ?o .
} where {
     ?s rdf:type vivo:CaseStudy .
     ?s ?p ?o  .
}
```

Conference Paper

```
construct {
             ?s ?p ?o .
} where {
     ?s rdf:type vivo:ConferencePaper .
     ?s ?p ?o  .
}
```

Editorial Article

```
construct {
             ?s ?p ?o .
} where {
     ?s rdf:type vivo:EditorialArticle .
     ?s ?p ?o  .
}
```

Proceedings

```
construct {
             ?s ?p ?o .
} where {
     ?s rdf:type bibo:Proceedings .
     ?s ?p ?o  .
}
```

Review

```
construct {
             ?s ?p ?o .
} where {
     ?s rdf:type vivo:Review .
     ?s ?p ?o  .
}
```

Academic Article

```
construct {
             ?s ?p ?o .
} where {
     ?s rdf:type bibo:AcademicArticle .
     ?s ?p ?o  .
}
```

7.8.2.2 Remove Other Entities

Journal

```
construct {
             ?s ?p ?o .
} where {
     ?s rdf:type bibo:Journal .
     ?s ?p ?o  .
}
```

8 Extending and Localizing VIVO

8.1 Overview

VIVO, and Vitro, its underlying technology, are open and flexible. There is significant opportunity to extend VIVO and/or Vitro to accommodate the needs of your institution. No extensions are needed – VIVO contains a comprehensive information representation for scholarship. Vitro provides a general purpose platform for semantic data management.

Some of the topics in this section are very common – most sites want to localize their branding, many sites use external authentication, and many sites use VIVO in languages other than english. Other topics are more advanced and less common – creating custom editing forms, for example.

Take what you need and leave the rest.

8.2 Internationalization

8.2.1 VIVO Language Support

Multiple language support can mean many things. When a VIVO site supports a language other than English, that support includes:

- Text that is displayed in the VIVO pages.
    - For example, menus, selections, prompts, tool-tips and plain text.
- Terms in the Ontology, which are frequently displayed as links or section headings.
    - Labels and descriptions of properties and classes
- Text in the data model.
    - For example, if a book title is available in both French and English, a French-speaking user sees the French title. If a title is available only in English, it is displayed, without regard to the user's preference in languages.

Languages can be selected in a variety of ways, depending on the installation parameters:

- A VIVO installer can configure VIVO to use one of the supported languages.
- Different users may see different languages, depending on the settings in their web browser.
- Different users may select a language from a list of available languages.

Language support in VIVO is being implemented in phases:

- Phase 1 includes read-only support of public pages:
    - Pages that are visible to users who are not logged in.
    - Also includes support of some administrative pages.
    - This is currently available.
- Phase 2 will also provide read-write support of profile pages:
    - Users will be able to edit language-specific data in profile pages.
- Phase 3 will support administrative pages
    - Creating user accounts, manipulating RDF data and other administrative functions.

- Phase 4 will support "back-end" pages.
    - Used to edit the ontology, or to do low-level editing on individual entities.

VIVO language files are available for English, Spanish, Brazilian Portuguese, and German. If you need support for another language, please inquire of the VIVO mailing lists, to see if another group is already developing the files you need.

8.2.2 Adding a language to your VIVO site

8.2.2.1 Adding language files to VIVO

VIVO is distributed with English as the only supported language. VIVO also includes a set of "pseudo-language" files, as a demonstration of how language support is implemented.

Additional language files are available in the Git repositories at https://github.com/vivo-project/Vitro-languages and https://github.com/vivo-project/VIVO-languages.

If the repository contains files for the language you want, in the VIVO release that you are using, you can just download those files and install them.

8.2.2.2 Translating VIVO into your language

First, contact the VIVO development team (see page 0): we would love to talk to you. We will tell you if anyone else is working on your language, and we will be happy to help with any questions you may have.

When you are ready to go ahead, you must determine the "locale" of your translation. Then you prepare translations of twenty-one files, as shown below.

8.2.2.3 The locale

Your locale is an internationally recognized code that specifies the language you choose, and the region where it is spoken. For example, the locale string `fr_CA` is used for French as spoken in Canada, and `es_MX` is used for Spanish as spoken in Mexico. Recognized codes for languages and regions can be found by a simple Google search. Here is a list of locales that are recognized by the Java programming language[108]. You may also use this definitive list of languages and regions[109], maintained by the Internet Assigned Numbers Authority.

The locale code will appear in the name of each file that you create. In the files that contain RDF data, the locale code will also appear at the end of each line.

⚠ When the locale code appears in file names, it contains an underscore (`en_US`). When it appears inside RDF data files, it contains a hyphen (`en-US`).

8.2.2.4 The language files

You can get the Spanish or the English files from the VIVO and Vitro language repositories, to use as a template for your own files.

108 http://www.oracle.com/technetwork/java/javase/javase7locales-334809.html
109 http://www.iana.org/assignments/language-subtag-registry

The example that follow assume that you are creating files for the Estonian language, as spoken in Estonia, with the locale `et_EE`.

Text strings (.properties)

These files contain about 1500 words and phrases that appear in the VIVO web pages. The page templates contain more than just text – they contain programming logic and display specifiers.

These words and phrases have been removed from the page templates, so no programming knowledge is required to translate them.

There is one file for phrases used in Vitro, the core around which VIVO is built, and one file for phrases that are specific to VIVO. In the example, these two files are both called `all_et_EE.properties`.

Example file names

```
[Vitro]/webapp/languages/et_EE/i18n/all_et_EE.properties
[VIVO]/languages/et_EE/themes/wilma/i18n/all_et_EE.properties
```

Example content

```
minimum_image_dimensions = Minimaalne pildi mõõdud: {0} x {1} pikslit
cropping_caption = Profiilifoto näeb alloleval pildil.
```

Freemarker Templates (.ftl)

Almost all of the text in the Freemarker templates is supplied by the text strings in the properties files. However, some Freemarker templates are essentially all text, and it seemed simpler to create a translation of the entire template. These include the `help` and `about` pages, the `Terms of Use` page, and the emails that are sent to new VIVO users.

Example file names

```
[Vitro]/webapp/languages/et_EE/templates/freemarker/search-help_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/termsOfUse_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-acctCreatedEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-acctCreatedExternalOnlyEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-confirmEmailChangedEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-firstTimeExternalEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-passwordCreatedEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-passwordResetCompleteEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-passwordResetPendingEmail_et_EE.ftl
[VIVO]/languages/et_EE/templates/freemarker/aboutMapOfScience_et_EE.ftl
[VIVO]/languages/et_EE/templates/freemarker/aboutQrCodes_et_EE.ftl
[VIVO]/languages/et_EE/templates/freemarker/mapOfScienceTooltips_et_EE.ftl
```

Example content

```
<section id="terms" role="region">
    <h2>kasutustingimused</h2>
    <h3>Hoiatused</h3>

    <p>
        See ${termsOfUse.siteName} veebisait sisaldab materjali; teksti informatsiooni
        avaldamine tsitaadid, viited ja pildid ikka teie poolt ${termsOfUse.siteHost}
        ja erinevate kolmandatele isikutele, nii üksikisikute ja organisatsioonide,
        äri-ja muidu. Sel määral copyrightable Siin esitatud infot VIVO veebilehel ja
        kättesaadavaks Resource Description Framework (RDF) andmed alates VIVO at
        ${termsOfUse.siteHost} on mõeldud avalikuks kasutamiseks ja vaba levitamise
        tingimuste kohaselt
        <a href="http://creativecommons.org/licenses/by/3.0/"
                target="_blank" title="creative commons">
            Creative Commons CC-BY 3.0
        </a>
        litsentsi, mis võimaldab teil kopeerida, levitada, kuvada ja muuta derivaadid
        seda teavet teile anda laenu ${termsOfUse.siteHost}.
    </p>
</section>
```

RDF data (.n3)

Data in the RDF models includes labels for the properties and classes, labels for property groups and class groups, labels for menu pages and more.

Example file names

```
[VIVO]/languages/et_EE/rdf/applicationMetadata/firsttime/classgroups_labels_et_EE.n3
[VIVO]/languages/et_EE/rdf/applicationMetadata/firsttime/propertygroups_labels_et_EE.n3
[VIVO]/languages/et_EE/rdf/display/everytime/PropertyConfig_et_EE.n3
[VIVO]/languages/et_EE/rdf/display/firsttime/aboutPage_et_EE.n3
[VIVO]/languages/et_EE/rdf/display/firsttime/menu_et_EE.n3
[VIVO]/languages/et_EE/rdf/tbox/firsttime/initialTBoxAnnotations_et_EE.n3
```

Example content

```
<http://vivoweb.org/ontology#vitroClassGrouppeople>
    <http://www.w3.org/2000/01/rdf-schema#label> "inimesed"@et-EE .
<http://vivoweb.org/ontology#vitroClassGrouppublications>
    <http://www.w3.org/2000/01/rdf-schema#label> "teadus"@et-EE .
<http://vivoweb.org/ontology#vitroClassGrouporganizations>
    <http://www.w3.org/2000/01/rdf-schema#label> "organisatsioonid"@et-EE .
<http://vivoweb.org/ontology#vitroClassGroupactivities>
    <http://www.w3.org/2000/01/rdf-schema#label> "tegevused"@et-EE .
```

The selection image (.png, .jpeg, .gif)

If you allow the user to select a preferred language, you must supply an image for the user to click on. Typically, this image is of the flag of the country where the language is spoken.

Example file names

```
[VIVO]/languages/et_EE/themes/wilma/i18n/images/select_locale_et_EE.gif
```

Example content

8.2.2.5 How can I contribute my language files to the VIVO community?

If you are planning to create a translation of VIVO, you should coordinate with the VIVO developers. When your files are ready, you can make them available to the development team in any way you choose. Note that the VIVO project will release your files under the Apache 2 License[110]. They will require a Contributor Agreement stating that you agree to those terms.

8.2.3 Adding language support to your local modifications

If you make changes to the VIVO code or templates, you may want to add language support to your changes. This is only necessary if your site supports multiple languages, or if you plan to contribute your code to the VIVO community.

8.2.3.1 Language in the data model

The usual form of language support in RDF is to include multiple labels for a single individual, each with a language specifier.

In fact, any set of triples in the data model are considered to be equivalent if they differ only in that the objects are strings with different language specifiers. If language filtering is enabled, VIVO will display the value that matches the user's preferred locale. If no value exactly matches the locale, the closest match is displayed.

Consider these triples in the data:

```
<http://abc.edu/individual/subject1> <http://abc.edu/individual/property1> "coloring" .
<http://abc.edu/individual/subject1> <http://abc.edu/individual/property1> "colouring"@en-UK .
<http://abc.edu/individual/subject1> <http://abc.edu/individual/property1> "colorear"@es .
```

VIVO would display these values as follows:

User's preferred locale	displayed text
en_UK	colouring
en_CA	colouring

110 http://www.apache.org/licenses/LICENSE-2.0.html

User's preferred locale	displayed text
`es_MX`	colorear
`fr_FR`	coloring

⚠ VIVO has limited language support for editing values in the GUI. It is possible to edit language-specific labels for individuals. Language-specific values for other properties must be ingested into VIVO.

8.2.3.2 Language support in VIVO pages

This section deals with the framework of the VIVO pages: the page titles, the prompts, the tool tips, the error messages; everything that doesn't come from the data model. These pieces of text are not stored in RDF, so we need a different mechanism for managing them.

The mechanism we use is based on the Java language's built-in framework for Internationalization. You can find more information in the Java tutorials for resource bundles[111] and properties files[112].

"Internationalization" is frequently abbreviated as "I18n", because the word is so long that there are 18 letters between the first "I" and the last "n".

In the I18n framework, displayed text strings are not embedded in the Java classes or in the Freemarker template. Instead, each piece of text is assigned a "key" and the code will ask the framework to provide the text string that is associated with that code. The framework has access to sets of properties files, one set for each supported language, and it will use the appropriate set to get the correct strings.

For example, suppose that we have:

- The text that will appear in an HTML link, used to cancel the current operation, with the key `cancel_link`.
- The title of a page used to upload an image, with the key `upload_image_page_title`.
- The text of a prompt message, telling users how big an image must be, with the key `minimum_image_dimensions`.

The default properties file might show the English language versions of these properties, like this:

Excerpt from all.properties

```
cancel_link = Cancel
upload_image_page_title = Upload image
minimum_image_dimensions = Minimum image dimensions: {0} x {1} pixels
```

Notice that the actual image dimensions are not part of the text string. Instead, placeholders are used to show where the dimensions will appear when they are supplied. This allows us to specify the language-dependent parts of a message in the properties file, while waiting to specify the language-independent parts at run time.

A Spanish language properties file might show the Spanish versions of these properties in a similar manner:

111 http://docs.oracle.com/javase/tutorial/i18n/resbundle/concept.html
112 http://docs.oracle.com/javase/tutorial/i18n/resbundle/propfile.html

Excerpt from all_es.properties

```
cancel_link = Cancelar
upload_image_page_title = Subir foto
minimum_image_dimensions = Dimensiones mínimas de imagen: {0} x {1} pixels
```

To use these strings in Java code, start with the I18n class, and the key to the string. Supply values as needed to replace any placeholders in the message.

Using I18n strings from Java code

```
protected String getTitle(String siteName, VitroRequest vreq) {
    return I18n.text(vreq, "upload_image_page_title);
}

private String getPrompt(HttpServletRequest req, int width, int height) {
    return I18n.text(req, "minimum_image_dimensions", width, height);
}
```

Similarly, using text strings in a Freemarker template begins with the `i18n()` method.

Using I18n strings in a Freemarker template

```
<#assign text_strings = i18n() >

<a href="../cancel" >
    ${text_strings.cancel_link}
</a>

<p class="note">
    ${text_strings.minimum_image_dimensions(width, height)}
</p>
```

Here is the appearance of the page in question, in English and in Spanish:

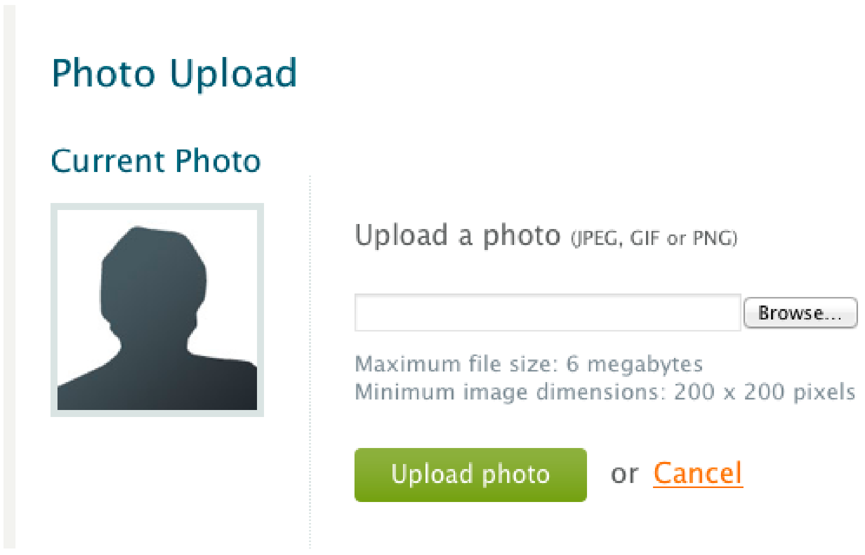


Photo Upload

Current Photo

Upload a photo (JPEG, GIF or PNG)

Maximum file size: 6 megabytes
Minimum image dimensions: 200 x 200 pixels

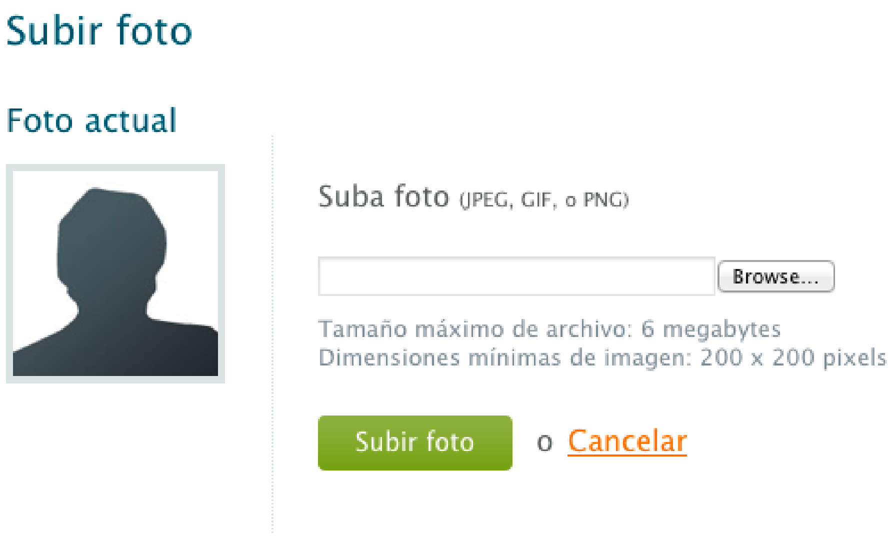

Subir foto

Foto actual

Suba foto (JPEG, GIF, o PNG)

Tamaño máximo de archivo: 6 megabytes
Dimensiones mínimas de imagen: 200 x 200 pixels

Structure of the properties files

The properties files that hold text strings are based on the Java I18n framework for resource bundles. Here is a tutorial on resource bundles[113].

Most text strings will be simple, as shown previously. However, the syntax for expressing text strings is very powerful, and can become complex. As an example, take this text string that handles both singular and plural:

A complex text string

```
deleted_accounts = Deleted {0} {0, choice, 0#accounts |1#account |1<accounts}.
```

113 http://docs.oracle.com/javase/tutorial/i18n/resbundle/concept.html

The text strings are processed by the Java I18n framework for message formats. Here is a tutorial on message formats[114]. Full details can be found in the description of the MessageFormat[115] class.

Local extension: application vs. theme

The Java I18n framework expects all properties files to be in one location. In VIVO, this has been extended to look in two locations for text strings. First, it looks for properties files in the current theme directory. Then, it looks in the main application area. This means that you don't need to include all of the basic text strings in your theme. But you can still add or override strings in your theme.

If your VIVO theme is named "frodo", then your text strings (using the default bundle name) would be in

- *webapp*`/themes/frodo/i18n/all.properties`
- *webapp*`/i18n/all.properties`

If you specify a complex locale for VIVO, this search pattern becomes longer. For example, if your user has chosen Canadian French as his language/country combination, then these files (if they exist) will be searched for text strings:

- *webapp*`/themes/frodo/i18n/all_fr_CA.properties`
- *webapp*`/i18n/all_fr_CA.properties`
- *webapp*`/themes/frodo/i18n/all_fr.properties`
- *webapp*`/i18n/all_fr.properties`
- *webapp*`/themes/frodo/i18n/all.properties`
- *webapp*`/i18n/all.properties`

When VIVO finds a text string in one of these files, it uses that value, and will not search the remaining files.

8.2.3.3 Language in Freemarker page templates

Here is some example code from `page-home.ftl`

Excerpt from page-home.ftl

```
<section id="search-home" role="region">
    <h3>${i18n().intro_searchvivo} <span class="search-filter-selected">filteredSearch</span></h3>
    <fieldset>
        <legend>${i18n().search_form}</legend>
        <form id="search-homepage" action="${urls.search}" name="search-home" role="search" method="post" >
            <div id="search-home-field">
                <input type="text" name="querytext" class="search-homepage" value="" autocapitalize="off" /
>
                <input type="submit" value="${i18n().search_button}" class="search" />
                <input type="hidden" name="classgroup"  value="" autocapitalize="off" />
            </div>
            <a class="filter-search filter-default" href="#" title="${i18n().intro_filtersearch}">
                <span class="displace">${i18n().intro_filtersearch}</span>
            </a>
            <ul id="filter-search-nav">
                <li><a class="active" href="">${i18n().all_capitalized}</a></li>
                <@lh.allClassGroupNames vClassGroups! />
```

114 http://docs.oracle.com/javase/tutorial/i18n/format/messageintro.html
115 http://docs.oracle.com/javase/7/docs/api/index.html?java/text/MessageFormat.html

```
            </ul>
        </form>
    </fieldset>
</section> <!-- #search-home -->
```

This code lays out all of the formatting and markup, but the actual strings of text are retrieved from the property files, depending on the current language and locale. Here are the English-language strings used by this code:

English properties used in the example

```
intro_searchvivo = Search VIVO
search_form = Search form
search_button = Search
intro_filtersearch = Filter search
all_capitalized = All
```

Language-specific templates

Most Freemaker templates are constructed like the one above; the text is merged with the markup at runtime. In most cases, this results in lower maintenance efforts, since the markup can be re-structured without affecting the text that is displayed.

In some cases, however, the template is predominantly made up of text, with very little markup. In these cases, it is simpler to rewrite the entire template in the chosen language.

The Freemarker framework has anticipated this. When a template is requested, Freemarker will first look for a language-specific version of the template that matches the current locale. So, if the current locale is `es_MX`, and a request is made for `termsOfUse.ftl`, Freemarker will look for these template files:

Search order for `termsOfUse.ftl` **Current locale is `es_MX`**
`termsOfUse_es_MX.ftl`
`termsOfUse_es.ftl`
`termsOfUse.ftl`

8.2.3.4 Language in Java code

Java code has access to the same language properties that Freemarker accesses. Here is an example of using a language-specific string in Java code:

Excerpt from UserAccountsAddPageStrategy.java

```
  FreemarkerEmailMessage email = FreemarkerEmailFactory.createNewMessage(vreq);
  email.addRecipient(TO, page.getAddedAccount().getEmailAddress());
  email.setSubject(i18n.text("account_created_subject", getSiteName()));
```

The properties files contain this line:

English language properties used in the example

```
account_created_subject = Your {0} account has been created.
```

Note how the name of the VIVO site is passed as a parameter to the text message.

8.2.3.5 Language in JSPs

Up through VIVO release 1.7, no attempt has been made to add language support to JSPs.

8.2.3.6 Language in JavaScript files

There is no convenient way to access the `i18n` framework from JavaScript files. One workaround is to assign values to JavaScript variables in the Freemarker template, and then access those values from the JavaScript.

For example, the template can contain this:

Excerpt from page-home.ftl

```
    <script>
        var i18nStrings = {
            countriesAndRegions: '${i18n().countries_and_regions}',
            statesString: '${i18n().map_states_string}',
    </script>
```

And the script can contain this:

Excerpt from homePageMaps.js

```
        if ( area == "global" ) {
            text = " " + i18nStrings.countriesAndRegions;
        }
        else if ( area == "country" ) {
            text = " " + i18nStrings.statesString;
        }
```

8.2.4 Tools you can use

8.2.4.1 i18nChecker

This is a set of Ruby scripts that are distributed with VIVO, in the `utilities/languageSupport/i18nChecker` directory. Use them to scan your language properties files and your freemarker templates. The scripts look for common errors in the files.

Scanning language properties files:

- Warn if a specialized file has no default version.
- Warn about duplicate keys, keys with empty values.
- Warn about keys that do not appear in the default version.
- If the "complete" flag is set,
    - Warn if the default version is not found.
    - Warn about missing keys, compared to the default version.

Scanning Freemarker templates:

- Warn about visible text that contains other than blank space or Freemarker expressions.
- Visible text is:
    - Anything that is not inside a tag and not between <script> tags
    - title="" attributes on any tags
    - alert="" attributes on <img> tags
    - alt="" attributes on <img> tags
    - value="" attributes on <input> tags with submit attributes

8.2.5 VIVO en Español

La herramienta Web Semántica VIVO ha demostrado ser útil para la vinculación de profesionales y científicos en las diferentes ramas de la ciencia. Más allá de un simple directorio o red social, VIVO posee capacidades de visualización y de intercambio de información importantes. VIVO es una herramienta "open source".

Con el soporte de VIVO para múltiples idiomas, que viene con la versión 1.6, esta página será el centro de información sobre VIVO en Español para hispanohablantes:

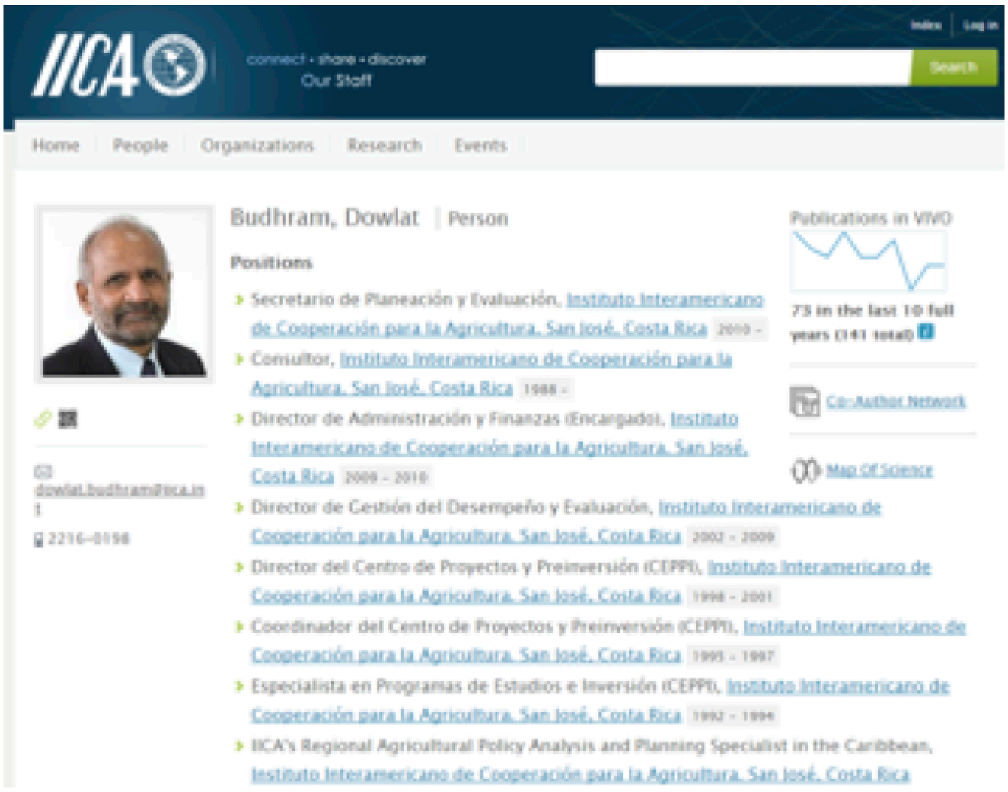
- ¿Qué es VIVO?[116]
- ¿Cómo instalar? (see page 114)
- FAQ
- Socios colaboradores: *El proyecto de adaptación de la herramienta VIVO en español ha sido un trabajo coordinado entre la Universidad de Cornell, el Departamento de Agricultura de Estados Unidos y el Instituto Interamericano de Cooperación para la Agricultura (*IICA[117]*) iniciado en 2013.*
- Listas de correo
- Comunidad de desarrolladores

Únase a las listas de correo[118] para preguntas sobre la implementación de VIVO, en Inglés o Español.

8.2.5.1 ¿Qué es VIVO?

¿Qué es VIVO?

116 https://wiki.duraspace.org/pages/viewpage.action?pageId=34665478
117 http://www.iica.int
118 https://lists.sourceforge.net/lists/listinfo/vivo-imp-issues



Budhram, Dowlat | Person

Positions

- Secretario de Planeación y Evaluación, Instituto Interamericano de Cooperación para la Agricultura, San José, Costa Rica 2010 -
- Consultor, Instituto Interamericano de Cooperación para la Agricultura, San José, Costa Rica 1988 -
- Director de Administración y Finanzas (Encargado), Instituto Interamericano de Cooperación para la Agricultura, San José, Costa Rica 2009 - 2010
- Director de Gestión del Desempeño y Evaluación, Instituto Interamericano de Cooperación para la Agricultura, San José, Costa Rica 2002 - 2009
- Director del Centro de Proyectos y Preinversión (CEPPI), Instituto Interamericano de Cooperación para la Agricultura, San José, Costa Rica 1998 - 2001
- Coordinador del Centro de Proyectos y Preinversión (CEPPI), Instituto Interamericano de Cooperación para la Agricultura, San José, Costa Rica 1995 - 1997
- Especialista en Programas de Estudios e Inversión (CEPPI), Instituto Interamericano de Cooperación para la Agricultura, San José, Costa Rica 1992 - 1994
- IICA's Regional Agricultural Policy Analysis and Planning Specialist in the Caribbean, Instituto Interamericano de Cooperación para la Agricultura, San José, Costa Rica

dowlat.budhram@iica.int

2216-0198

Publications in VIVO

73 in the last 10 full years (141 total)

Co-Author Network

Map Of Science

VIVO es una plataforma web semántica de acceso abierto que permite descubrir la investigación y el saber técnico en las múltiples disciplinas y extremos administrativos, por medio de perfiles profesionales vinculados e información relacionada. VIVO fue desarrollado originalmente por la Universidad de Cornell, que luego del 2009 en conjunto con otras cinco universidades en Estados Unidos, lo ampliaron como una herramienta capaz de integrar perfiles entre varias instituciones. Asimismo, su adopción facilita la colaboración entre personas no solo en el ámbito interno de las organizaciones, sino entre los diferentes sectores.   En 2013, el IICA con ayuda del Departamento de Agricultura de los Estados Unidos y la Universidad de Cornell iniciaron la adaptación de la herramienta al idioma español.

VIVO se completa con información acerca de investigadores, técnicos u otros individuos que les permite destacar sus áreas de experiencia, desplegar credenciales académicas, visualizar sus redes de trabajo y mostrar información sobre publicaciones, proyectos, servicios y más. Los perfiles profesionales y sus descripciones pueden ser importados de manera programada de fuentes oficiales tales como registros institucionales, repositorios locales y otras bases de datos bibliográficas.

VIVO y otras aplicaciones compatibles producen una extensa red de conocimiento entre instituciones, organizaciones y agencias, las que en sus búsquedas contribuyen al trabajo colaborativo, las sinergias y a la apertura del conocimiento. El software abierto de VIVO (en sus versiones en español e inglés) y sus ontologías están disponibles públicamente, así como los materiales de soporte para implementar, adoptar o desarrollar. Para más información, visite http://vivoweb.org.

La ciencia tangible

VIVO provee de herramientas de visualización y análisis de redes que maximizan los beneficios con la utilización de los datos disponibles. Esta herramienta permite que datos de alta calidad, como son investigadores, sus colaborares y fuentes sean revelados, de tal manera que ofrece una visualización elegante del esfuerzo investigador a nivel local, multinacional o global.

Visualizaciones que muestran publicaciones	Redes de coautores y coinvestigadores	El mapa de la ciencia muestra las áreas temáticas fuertes de una organización o un individuo

¿Por qué utilizar VIVO?

Cualquier individuo tendrá acceso a un buscador de VIVO vía web. Investigadores, académicos, técnicos, administradores, agencias financieras, donantes y la sociedad civil se beneficiarán de utilizar VIVO y sus datos debido a que pueden:

- Crear equipos de investigación interdisciplinarios
- Identificar oportunidades de apoyo financiero
- Reclutar personal especializado
- Localizar publicaciones
- Planificar recursos, servicios y presupuestos
- Visualizar redes complejas y relaciones de trabajo

Fuentes de información

A diferencia de otras plataformas o redes sociales, VIVO se sustenta en datos incluidos automáticamente de fuentes institucionales oficiales que se pueden complementar con información adicionada en forma manual. Por ejemplo, el cosechador le facilita el trabajo a los equipos locales de implementación, puesto que recupera información de otros sistemas relacionados con publicaciones, recursos humanos, eventos, entre otros.

Metabuscador de VIVO

Este es un singular sitio que permite encontrar, entre los diferentes VIVO, personas, artículos, eventos, organizaciones y conceptos en diferentes organizaciones. El metabuscador de VIVO provee búsquedas relevantes a lo largo de los servidores o servicios en la nube que utilizan las ontologías de VIVO u otras provenientes de sistemas similares.

Un "hub de conocimiento" agrícola

El esfuerzo iniciado con el IICA, busca motivar a sus 34 Estados Miembros para que integren la mayor cantidad de perfiles profesionales en el campo agrícola en un solo lugar. La experiencia de más de una década en gestión de información documental en el marco de la Alianza SIDALC (www.sidalc.net[119]) y las más de 172 instituciones asociadas, ubican al Instituto en una posición preferente para articular una plataforma VIVO multilingüe que vincule mejor a los científicos, técnicos y otros actores relevantes en el sector agropecuario. Este esfuerzo

119 http://www.sidalc.net

hemisférico se sumaría al global de AGRIVIVO[120] el cual lidera la Organización de Naciones Unidas para la Alimentación y la Agricultura, así como al Movimiento Mundial CIARD[121] sobre Coherencia de Información Agrícola para el Desarrollo.

Con la cooperación de:

	USDA	IICA

8.2.5.2 ¿Cómo instalar?

Install the necessary software

Before installing VIVO sure you have the following programs installed on your computer:

- Java (SE) 1.7.x Java Platform (JDK)[122]
    - VIVO has not been tested with OpenJDK
- Apache Tomcat 6.x or 7.x http://tomcat.apache.org
- Apache Ant 1.8 or higher, http://ant.apache.org[123]
- MySQL 5.1 or higher, http://www.mysql.com

120 http://www.agrivivo.net
121 http://www.ciard.net
122 http://www.oracle.com/technetwork/es/java/javase/downloads/index.html
123 http://ant.apache.org/bindownload.cgi

Check if you have enabled and ANT_HOME variables JAVA_HOME system environment. The configuration of these requirements depends on the operating system you are using. Check the installation guides for each program to make the correct settings.

The following browsers are supported for this release:

- Mac:
  - Chrome 30.0.1599.69 or higher
  - FireFox 3.6.28, 10.0.12, 24
  - Opera 12.02
  - Safari 5.0.3
- PC:
  - Chrome 25.1364.2 or higher
  - FireFox 10.0.12, 24
  - Internet Explorer 8, 9, 10
  - Opera 12.02

It worked?

You can try installing the programs by typing the following commands:

```
[~] # Java -version  java version "1.7.0_25"  Java (TM) SE Runtime Environment (build 1.7.0_25-b15)  Java
HotSpot (TM) 64-Bit Server VM (build 2325-b01, mixed mode)  [~] # Mysql --version  mysql View 14.14 Distrib
5.5.36, for Linux (x86_64) using readline 5.1  [~] # Ant -version  Apache Ant (TM) version 1.9.1 compiled
on May 15 2013
```

Each of these commands will print versions you have installed. If any of these commands print an error message, you must check the installation.

Create an empty database and account database

Decide on a database name, user name and password. You will need these values for this step and again when specify runtime properties.

Login to your MySQL server and creates a new database that uses the character encoding in UTF-8 format. In the MySQL command line you can create the database and the user with the following commands substituting values for database, user and password. Usually the computer name is called localhost.

```
CREATE DATABASE CHARACTER SET utf8 for database;  GRANT ALL ON * TO for database user @ 'localhost'
 IDENTIFIED BY 'password.';
```

build LIVE

Download the source code VIVO

Download the source code from the available links LIVE download with names: rel-1.6.zip or rel-1.6.gz and unzip on your web server. You can download the file from the following link: http://vivoweb.org/download

Specifies the properties of construction

Within the VIVO distribution directory, renames the file build.properties example.build.properties. Edit file to meet your installation, as described in the next section.

These properties are used when compiling VIVO and when deployed within Tomcat. These will be incorporated into LIVE when fully compiled. If you want to change these properties later, you must stop the Tomcat service, repeat step compile and deploy, in the end, you must restart the Tomcat to see these changes.

⚠ Windows: To install on a Windows operating system, you must include the letter of the hard drive, you must use the slash "/" and not the backslash "\" in the directory path, for example c: / tomcat

Property Name	**vitro.core.dir**
Description	The directory where Vitro is located. In the simple installation, it is assigned to ./vitro-core, the current directory.
default	Any
Example value	./vitro-core

Property Name	**tomcat.home**
Description	The directory where tomcat is installed.
default	Any
Example value	/ Usr / local / tomcat

Property Name	**webapp.name**
Description	The name of your VIVO application. This is not the name that will be displayed to the user. This name appears in the URL used to access LIVE, and the name of the directory path VIVO within tomcat.
default	Any
Example value	alive

Property Name	vitro.home
Descriptio n	It is the directory where VIVO will store the data that are created. This includes uploaded files (usually images) and Solr search indexes. Make sure the directory exists and is writable by the Tomcat server.
default	Any
Example value	/ Usr / local / live / home

Compile and deploy

In previous steps, you have defined directory location VIVO, specifying property values in the build.properties file vitro.home. If the directory does not exist, create it now.

In the command line within the VIVO distribution directory, type the following command:

```
ant all
```

VIVO to build and deploy in the Tomcat webapps directory.

The build script can run up to five minutes, and create more than 100 output lines, the process includes several steps:

- Collect files distribution source directory.
- Compile the Java source code.
- Running unit tests.
- Prepare the Solr search engine.
- Vivo deployed to Tomcat and Solr.

It worked?

If the outgoing message is successful, then the construction has been completed successfully. Proceed to the next step.

```
BUILD SUCCESSFUL

Total time: 1 minute 49 seconds
```

If the output ends with an error message, building failed. Find the fault of the error, correct the problem, and run the script again.

```
BUILD FAILED

Total time: 35 seconds
```

Construction output may include warning messages. Java compiler can warn of outdated code. Unit tests can produce warning messages, and some tests can be ignored if you do not produce consistent results. If the output ends with a success message, these messages will be ignored.

run LIVE

---

configuring Tomcat

Assign parameters JVM

VIVO copy small sections of your base RDF data within memory to serve web requests quickly (the copy in memory and the database remains in sync when editing).

VIVO may require more memory than it has assigned Tomcat by default. Like many facilities Tomcat, the file or setenv.bat setenv.sh in Tomcat bin directory is a convenient place to allocate memory settings instead. If this file does not exist within Tomcat directories, you can create it.

For example:

setenv.sh

```
export CATALINA_OPTS = "- Xms512m -Xmx512m -XX: MaxPermSize = 128m"
```

This tells tomcat to assign an initial value of 512 megabytes, 512 megabytes maximum value, and a space of 128 megabytes to PermGem. Lower values may be insufficient, especially for small installation tests.

Set safety limits

Vivo is a multithreaded web application that can require more wires than are allowed in the default configuration of your Linux installation. Make sure your system can support the required number of threads by editing the following lines in the file /etc/security/limits.conf:

```
apache hard nproc 400  tomcat6 hard nproc 1500
```

Configure URI encoding

LIVE handled properly for international characters, you have to configure Tomcat under standard UTF-8 characters.

Edit the conf / server.xml file and add the following attributes for each element Connector:

URIEncoding = "UTF-8"

```
<Server ...>     <Service ...>        <Connector ... URIEncoding = "UTF-8" />           ...        </
Connector>     </ Service>  </ Server>
```

⚠ Some versions of Tomcat bring this attribute included as default.

Be careful when creating elements of context

Each Web application in the distribution of VIVO (VIVO and Solr) includes a file "context fragment" containing information for the Web application deployment.

Tomcat allows these fragments override it by adding elements of context Context context the server.xml file. If you decide to do this, make sure the new context item includes the necessary deployment parameters from context chunk replaced.

Look at the section titled Live Running behind an Apache Server for an example replacement snippet LIVE context.

Runtime Properties

In the process of building VIVO, specifically in the compilation and deployment, a file called example.runtime.properties in the home directory LIVE (specified by vitro.home in the build.properties file) was created, rename this file at runtime .properties and edit the file according to your installation, as described below.

These properties are loaded when you start VIVO. If you want to change these properties at a later date, you need to restart Tomcat for the changes to take effect. No need to repeat step compile and deploy.

⚠ Windows: To install on a Windows operating system, you must include the letter of the hard drive, you must use the slash "/" and not the backslash "\" in the directory path, for example c: / tomcat

basic properties

These properties define some fundamental aspects of the installation of VIVO. Many sites will need to modify these values.

Property Name	**Vitro.defaultNamespace**
Description	The RDF default namespace for this installation. VIVO installation RDF makes its resources available for harvest using linked data. Requests for RDF resource URI redirected to HTML or RDF as that specified by the customer. To make this possible, the default namespace VIVO must have a certain structure and start with a public web address VIVO installation. For example, if the web address VIVO facility is http://vivo.example.edu/ the namespace must be assigned to http://vivo.example.edu/individual in order to support linked data. Similarly, if LIVE is installed in http://www.example.edu/vivo the namespace must be assigned to http://www.example.edu/vivo/individual * The namespace must end with "individual /" (including the slash).
default	Any

Example value	http://vivo.midominio.edu/individual/

Property Name	**rootUser.emailAddress**
Description	Specifies the email address of the primary user account of the VIVO application. This user will have a temporary initial password: rootpassword. You will be prompted to create a new password at first logon. Note: The primary user account has access to all data and all operations LIVE. Data views can be amazing when the main user. It is better to create a site administrator account for use in each administrative task.
default	Any
Example value	vivoAdmin@midominio.edu[124]

Property Name	**VitroConnection.DataSource.url**
Description	Specifies the JDBC URL for your database. Changes the end of the URL with the name of your database (If this is not "live").
default	Any
Example value	jdbc: mysql: // localhost / live (see page 114)

Property Name	**VitroConnection.DataSource.username**
Description	Change the user name that matches the authorized user you created for MySQL.
default	Any
Example value	username

Property Name	**VitroConnection.DataSource.password**

124 mailto:vivoAdmin@midominio.edu

Description	Change the password match which gave high in MySQL.
default	Any
Example value	features Password

Property Name	**email.smtpHost**
Description	Specifies an SMTP service that the application will use to send email (optional). If this is left empty, the contact form will be hidden and disabled, and users will not be notified of changes in their accounts.
default	Any
Example value	smtp.servidor.edu

Property Name	**email.replyTo**
Description	Specifies an email address which will appear as the sender on notifications via e-mail users (optional). If a user answers the notification, this address will receive the answer. If an email address is invalid user, this address will receive the error message. If this is left empty, users will not be notified of changes in their accounts.
default	Any
Example value	vivoAdmin@midominio.edu[125]

Connecting the Solr search index

VIVO and search index are currently two different web applications and simple installation puts the two in the same instance of Tomcat. Still, you have to tell the VIVO application how to get to the Solr Web application.

Property Name	**vitro.local.solr.url**

125 mailto:vivoAdmin@midominio.edu

Description	The URL in the context of Solr used in local search VIVO. You should consist of: servername scheme + + + port + vivoweb_app_name "solr" In the standard installation, the context of Solr will be on the same server as VIVO, and the same instance of Tomcat. The route has to be webapp.name LIVE (specified below) + "solr"
default	Any
Example value	http: // localhost: 8080 / vivosolr[126]

additional properties

The runtime.properties file can accept many additional properties, but are not needed for this simple installation. If you choose any of the installation options, you'll probably need to configure some of these properties.

start Tomcat

Many of the facilities running Tomcat can be started the following files startup.sh or startup.bat in the Tomcat bin directory. Start Tomcat and go to your browser to http: // localhost: 8080 / live[127] to test the application.

Note that Tomcat may require several minutes to start VIVO.

When you start VIVO, some diagnostic tests run. If a problem is detected VIVO home page redirect to the home page status describing the problem. You can stop Tomcat, correct the problem and proceeds to step compile and deploy. If the problem is not serious, the start status page can provide a link to "continue" which will allow VIVO use despite problems.

If the start was successful, you will see the homepage of VIVO.

If tomcat does not start, or the VIVO application is not visible, check the files in the Tomcat logs directory.

Error messages can be found in [tomcat] /logs/catalina.out [tomcat] /logs/vivo.all.log or [tomcat] /logs/localhost.log.

⚠ Remember that Tomcat must have permissions to read and write files, and files in the root directory of VIVO. This means you have to use a particular script or particular user account to start Tomcat.

PDF file of this document[128]

8.2.6 VIVO in Mandarin

With VIVO's support for multiple languages, this page provides information about VIVO in Mandarin

- VIVO 1.5 Install Guide in Mandarin (see page 122)

126 http://localhost:8080/vivosolr
127 http://localhost:8080/vivo
128 https://wiki.duraspace.org/download/attachments/96995812/%C2%BFC%C3%B3mo%20instalar_-v31-20140423_1420.pdf?api=v2&modificationDate=1522787189599&version=1

- Resource Bundle File in Mandarin[129]

8.3 Customizing the Interface

8.3.1 Introduction

8.3.1.1 Making changes to VIVO

The VIVO application is a popular tool for research networking. Most VIVO sites put their own changes into VIVO, in order to create a distinctive appearance, or to satisfy their particular needs.

VIVO supports an assortment of tools and techniques for making these changes. Some changes can be accomplished while VIVO is running, simply by setting values on a form. Other changes require you to add or modify configuration files that control the application. Still other changes are accomplished by editing the VIVO code, re-building, and re-deploying the application.

8.3.1.2 VIVO is already customized

Customization is built in to the heart of VIVO. VIVO itself is a customization of a more basic product called Vitro.

Here is how Vitro has been customized to become VIVO

Vitro	**VIVO**
No ontology	Includes an ontology for Research Networking
Minimal theme	Rich theme.
Default display rules	Annotations are used to: • Assign data properties to groups • Arrange property groups on the page

129 https://wiki.duraspace.org/download/attachments/96995815/all_zh.properties?api=v2&modificationDate=1522787189835&version=1

Vitro	VIVO
Default permissions	Display and editing permissions are customized, based on the ontology
Default editing forms	Editing is customized to the ontology
Default search index	Search index contains additional fields, specific to VIVO
Default functionality	Additional functionality: visualizations, interface to Harvester, QR codes, etc.
In total: A general-purpose tool for working with Semantic Data.	**In total:** A specialized tool for Research Networking

8.3.2 Adding your own customizations

How do you add your changes to VIVO? Perhaps more important, how do you keep your changes when you upgrade to a newer release of VIVO?

8.3.2.1 Working in the GUI

When you use forms in VIVO, the values you enter are kept in the triple-store. They will be retained when you upgrade to a new release. If the new release uses a different format to store the values, your changes will be migrated to the new format.

8.3.2.2 RDF files

Some customizations require that you add or modify an RDF file in your VIVO home directory. In general, it's best to create a new file to contain the RDF statements, so you can easily carry your changes to a new VIVO release.

A "clean" build of VIVO will erase the RDF files in your VIVO home directory. You will need to re-create these files after the migration.

8.3.2.3 Changes to the source files

As with the RDF files, you should favor new files over changes to existing files. This will make it easier to carry your changes to a new release.

8.3.3 Tool summary

8.3.3.1 Required skills

The customization tools require different levels of knowledge. Some are as simple as filling out a web form. Most require the ability to write HTML, with additions from the Freemarker template engine. Some require Java programming.

As the tools are described, these terms will be used to specify the skills needed:

	Knowlege required
Basic	Requires an understanding of VIVO concepts.
Web development	The usual technologies for writing web sites, including HTML, CSS, and JavaScript. Knowledge of the Freemarker template engine.
RDF	Modify or create RDF data files, using RDF/XML, Turtle, or N3 format.
SPARQL	Create queries against the triple-store, using SPARQL.
Java	Create or modify Java code.
OpenSocial	Create or modify OpenSocial gadgets, written in JavaScript.

8.3.3.2 The tools

	What does it do?	**How?**	**Required skills**
Creating a custom theme (see page 181)	Create your own "brand" for VIVO. • Change colors, logo, headings, footers, and more.	CSS files, JavaScript files, and templates for HTML.	Web development
Annotations on the ontology (see page 138)	Control how data is displayed. • Property groups, labels, display order, hidden properties, and more.	Interactive.	Basic
Home page customizations (see page 127)	Choose from home page options. • Add a geographic focus map.	Edit your home page template to include a selection of sub-templates.	Web development
Menu and page management (see page 136)	Add new pages to VIVO. • Static pages, navigation pages, or dynamic reports.	Interactive.	Web development, optional SPARQL
Profiles for classes (see page 154)	Use one type of profile page for people and another for organizations.	Create page templates. Configure VIVO to associate them with classes.	Web development, RDF

Multiple profile types for foaf:Person (see page 209)	Provide a choice of formats for profile pages. • Each page owner selects the format for his own page.	Edit page templates. Perhaps connect to a Website image capture service.	Web development
Enriching profile pages with SPARQL queries (see page 204)	Display additional data on a profile page.	Write a SPARQL query. Create a template to display the results. Configure VIVO to use it.	Web development, SPARQL, RDF
Enhancing page templates with SPARQL queries (see page 201)	Display additional data in any page template.	Write a SPARQL query. Modify a template to display the results. Configure VIVO to use it.	Web development, SPARQL, RDF
Custom list views (see page 159)	Change how certain properties are displayed • Change the layout for that property • Display additional data with each value.	Write a SPARQL query. Create a template to display the results. Configure VIVO to use it.	Web development, SPARQL, RDF
Custom short views (see page 167)	Change how search results are displayed • Display depends on the type of result (Person, Document, etc.). Also change display on index pages and browse pages.	Write a SPARQL query. Create a template to display the results. Configure VIVO to use it.	Web development, SPARQL, RDF
Custom entry forms (see page 186)	Create data entry forms • Add or edit complex data structures.	Write a generator class in Java. Create a template for the editing form.	Web development, SPARQL, RDF, Java
Using Open Social Gadgets (see page 213)	Create optional content for profile pages. • Each page owner configures the gadgets for his own page.	Create gadgets from JavaScript, or install existing gadgets.	Web development, OpenSocial

Language support (see page 99)	Languages other than English • Use VIVO in Spanish • Allow viewers to choose their preferred language. • Implement other languages.	Create files of phrases in the desired language, or install existing files.	Basic

8.3.4 Home page customizations

8.3.4.1 Introduction

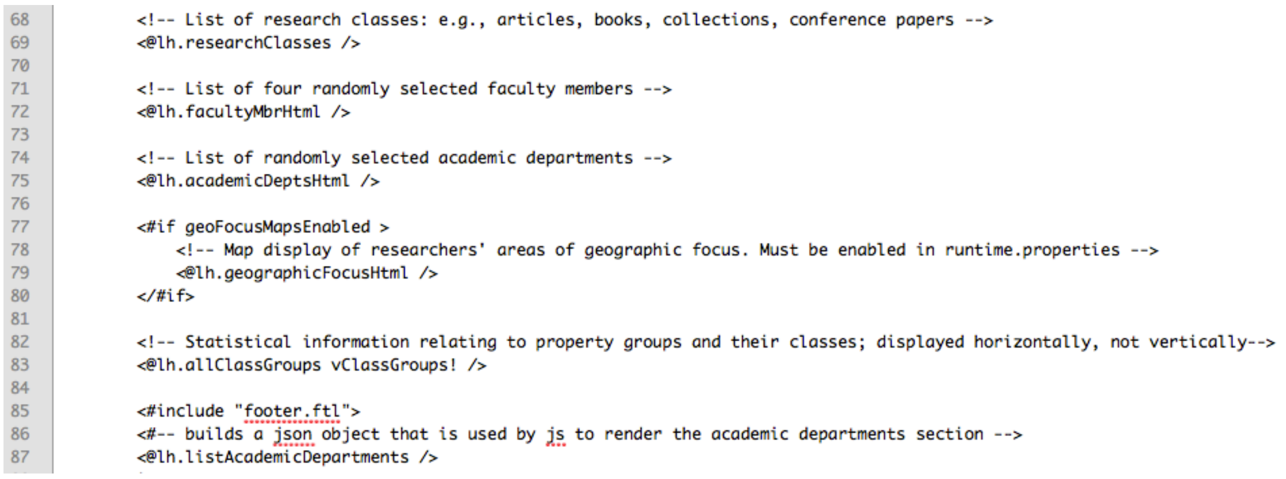
You can modify the "Research," "Faculty" and "Departments" sections of the home page, as well as expand the map section to include country-specific and state or province-specific maps.

8.3.4.2 The page-home.ftl Template File

The new sections of the home page are all referenced as macros in the page-home.ftl template file. The macros themselves are all located in the lib-home-page.ftl file, which is imported into the page-home.ftl file via this line:

```
<#import "lib-home-page.ftl" as lh>
```

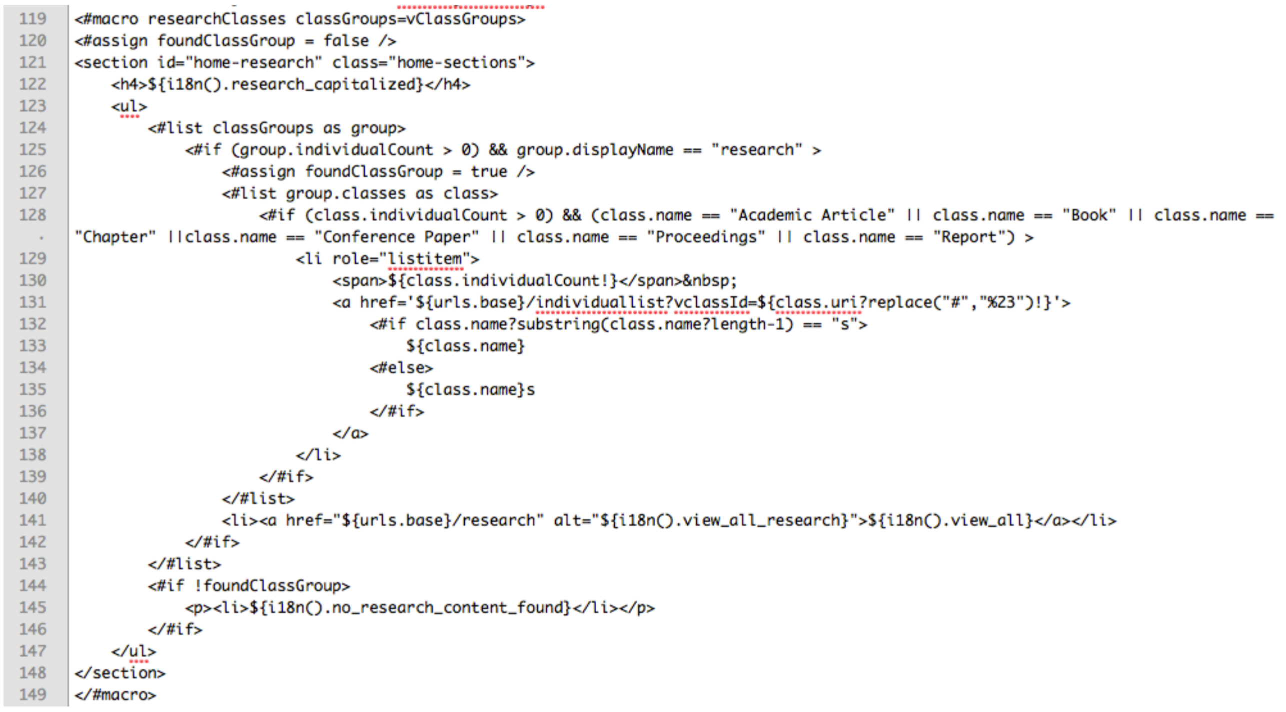
The code below is from the page-home.ftl template and shows how the macros are referenced. So, for example, if you wanted to modify the order in which these sections appear on the home page, you would move the macro references accordingly.



```
        <!-- List of research classes: e.g., articles, books, collections, conference papers -->
        <@lh.researchClasses />

        <!-- List of four randomly selected faculty members -->
        <@lh.facultyMbrHtml />

        <!-- List of randomly selected academic departments -->
        <@lh.academicDeptsHtml />

        <#if geoFocusMapsEnabled >
            <!-- Map display of researchers' areas of geographic focus. Must be enabled in runtime.properties -->
            <@lh.geographicFocusHtml />
        </#if>

        <!-- Statistical information relating to property groups and their classes; displayed horizontally, not vertically-->
        <@lh.allClassGroups vClassGroups! />

        <#include "footer.ftl">
        <#-- builds a json object that is used by js to render the academic departments section -->
        <@lh.listAcademicDepartments />
```

8.3.4.3 The Research Section

It's possible that your VIVO installation has defined some of its own classes within the Research Class group. Cornell's VIVO, for example, has a Library Collection class and a Media Contributions class. If your installation does include its own classes in this group, you can display these in the Research section of the home page by modifying the researchClasses macro in the lib-home-page.ftl file. As shown in line 128 below, the classes that get displayed are hard-coded into the macro. Simply exchange the name of your classes with some or all of the ones below. You could also add your classes to the existing list.

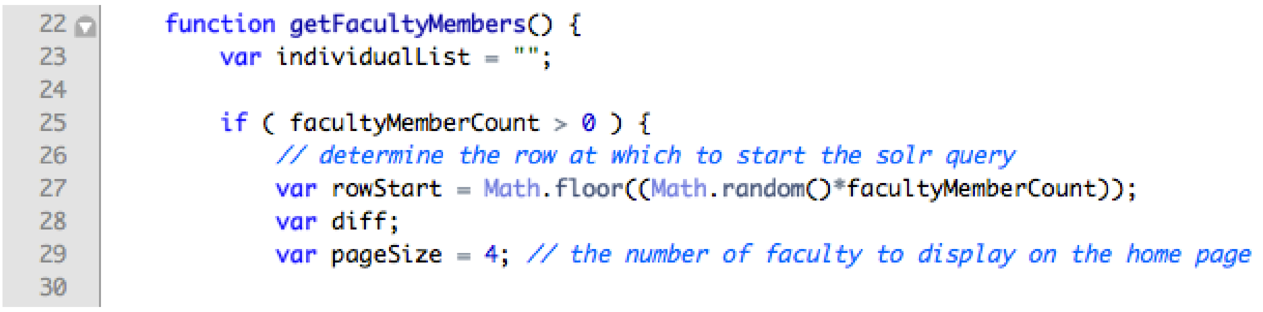


```
119 <#macro researchClasses classGroups=vClassGroups>
120 <#assign foundClassGroup = false />
121 <section id="home-research" class="home-sections">
122     <h4>${i18n().research_capitalized}</h4>
123     <ul>
124         <#list classGroups as group>
125             <#if (group.individualCount > 0) && group.displayName == "research" >
126                 <#assign foundClassGroup = true />
127                 <#list group.classes as class>
128                     <#if (class.individualCount > 0) && (class.name == "Academic Article" || class.name == "Book" || class.name ==
    "Chapter" ||class.name == "Conference Paper" || class.name == "Proceedings" || class.name == "Report") >
129                         <li role="listitem">
130                             <span>${class.individualCount!}</span>&nbsp;
131                             <a href='${urls.base}/individuallist?vclassId=${class.uri?replace("#","%23")!}'>
132                                 <#if class.name?substring(class.name?length-1) == "s">
133                                     ${class.name}
134                                 <#else>
135                                     ${class.name}s
136                                 </#if>
137                             </a>
138                         </li>
139                     </#if>
140                 </#list>
141                 <li><a href="${urls.base}/research" alt="${i18n().view_all_research}">${i18n().view_all}</a></li>
142             </#if>
143         </#list>
144         <#if !foundClassGroup>
145             <p><li>${i18n().no_research_content_found}</li></p>
146         </#if>
147     </ul>
148 </section>
149 </#macro>
```

It would be possible to display a random selection of classes rather than a hard-coded list, the same way that the Departments section displays a randomly selected list of academic departments. To do this, you would have to copy the macros and java script used for the academic departments, and then modify it accordingly so that it displays research classes. Refer to The Departments Section below for more details.

8.3.4.4 The Faculty Section

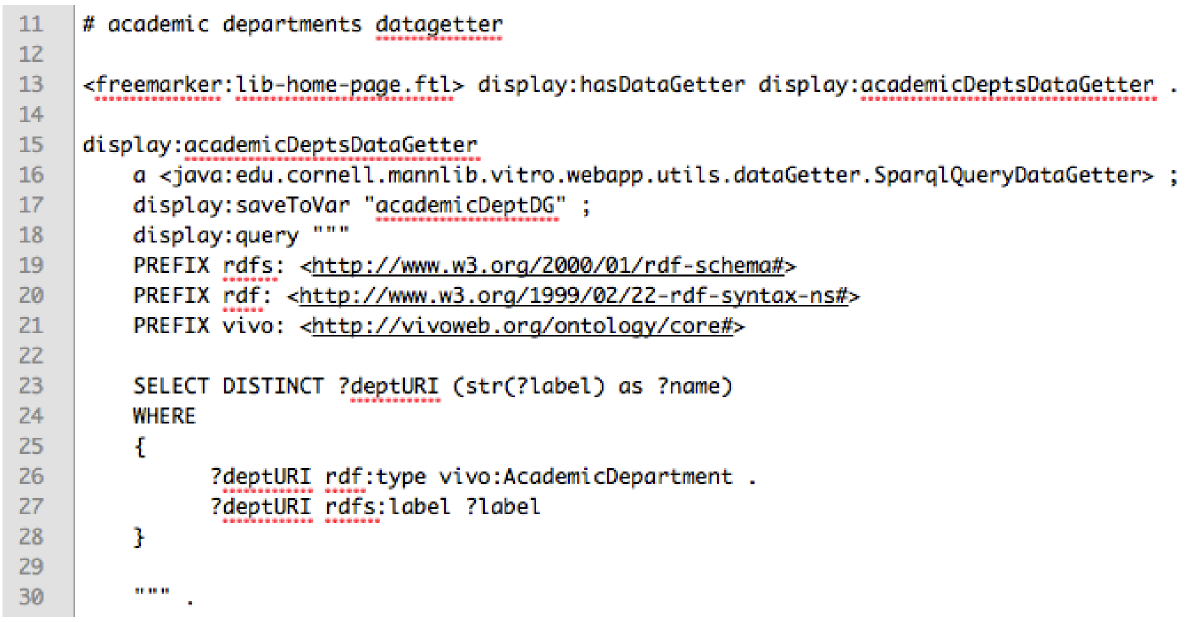
There's very little customization that can be done to the faculty section of the home page, excluding css changes and relocating the section to another part of the home page. The one configurable piece is the number of faculty members that get displayed. This change is made in the homePageUtils.js file. Locate the getFacultyMembers function and modify the pageSize variable (shown in line 29 below).



```
22      function getFacultyMembers() {
23          var individualList = "";
24
25          if ( facultyMemberCount > 0 ) {
26              // determine the row at which to start the solr query
27              var rowStart = Math.floor((Math.random()*facultyMemberCount));
28              var diff;
29              var pageSize = 4; // the number of faculty to display on the home page
30
```

8.3.4.5 The Departments Section

The list of academic departments is a randomly selected list that relies on a data getter as well as two macros in the lib-home-page.ftl file. The data getter is defined in the homePageDataGetters.n3 file. If you want to display something other than academic departments, you need to update the SPARQL query portion of the data getter, shown in lines 18-30 below. Substitute the class you want to display for vivo:AcademicDepartment.



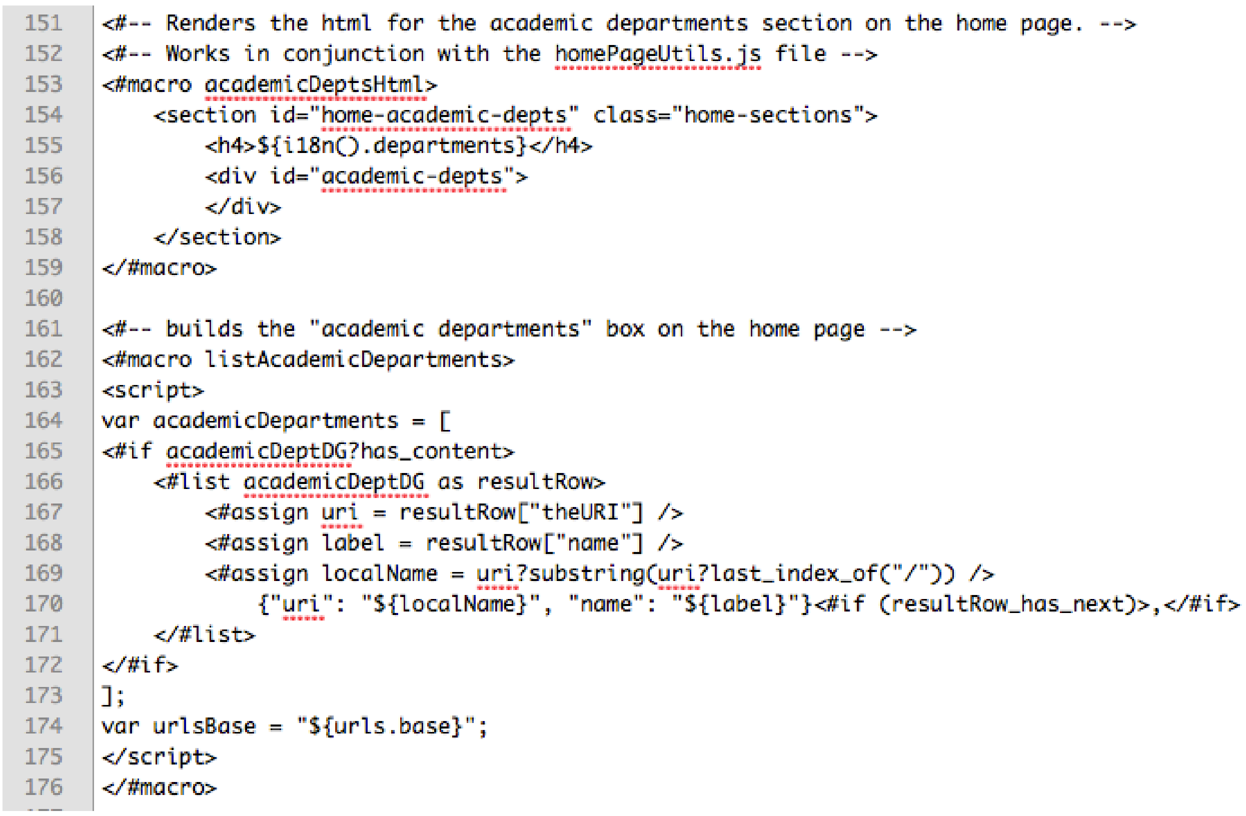
```
11  # academic departments datagetter
12
13  <freemarker:lib-home-page.ftl> display:hasDataGetter display:academicDeptsDataGetter .
14
15  display:academicDeptsDataGetter
16      a <java:edu.cornell.mannlib.vitro.webapp.utils.dataGetter.SparqlQueryDataGetter> ;
17      display:saveToVar "academicDeptDG" ;
18      display:query """
19      PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
20      PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
21      PREFIX vivo: <http://vivoweb.org/ontology/core#>
22
23      SELECT DISTINCT ?deptURI (str(?label) as ?name)
24      WHERE
25      {
26            ?deptURI rdf:type vivo:AcademicDepartment .
27            ?deptURI rdfs:label ?label
28      }
29
30      """ .
```

It is possible to expand the query to include more than one class. To do so without having to make any other macro or template changes, use UNION clauses in your query, as follows:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX vivo: <http://vivoweb.org/ontology/core#>
```

```
SELECT DISTINCT ?theURI (str(?label) as ?name)
WHERE
{{
   ?theURI rdf:type vivo:AcademicDepartment .
   ?theURI rdfs:label ?label .
}
UNION
{
   ?theURI rdf:type vivo:Association .
   ?theURI rdfs:label ?label .
}}
```

The following code snippet shows the two macros used to render the Departments section. If you change the data getter to use a different class, you do not have to change any variable or macro names. The only change you'll need to make is to the heading of this section so that it correctly reflects the class being displayed. Line 155 (below) is where you would make the change. (Note the use of the internationalization variable. As part of your change, you may want to update the i18n/all.properties file to include your new section heading.



```
151 <#-- Renders the html for the academic departments section on the home page. -->
152 <#-- Works in conjunction with the homePageUtils.js file -->
153 <#macro academicDeptsHtml>
154     <section id="home-academic-depts" class="home-sections">
155         <h4>${i18n().departments}</h4>
156         <div id="academic-depts">
157         </div>
158     </section>
159 </#macro>
160 
161 <#-- builds the "academic departments" box on the home page -->
162 <#macro listAcademicDepartments>
163 <script>
164 var academicDepartments = [
165 <#if academicDeptDG?has_content>
166     <#list academicDeptDG as resultRow>
167         <#assign uri = resultRow["theURI"] />
168         <#assign label = resultRow["name"] />
169         <#assign localName = uri?substring(uri?last_index_of("/")) />
170             {"uri": "${localName}", "name": "${label}"}<#if (resultRow_has_next)>,</#if>
171     </#list>
172 </#if>
173 ];
174 var urlsBase = "${urls.base}";
175 </script>
176 </#macro>
```

8.3.4.6 The Geographic Focus Map

The new map on the home page uses circular markers to show the countries and regions that researchers in a VIVO installation have chosen as their areas of geographic focus (vivo:GeographicFocus). Clicking on a marker takes the user to that country or region's profile page, which shows the list of researchers in that location. The map is built using the Leaflet.js java script library, map tiles provided without charge by ESRI, and geographical data stored in a JSON file.

The map is enabled in the runtime.properties file. Include or uncomment the line:

```
homePage.geoFocusMaps=enabled
```

How the Map Works

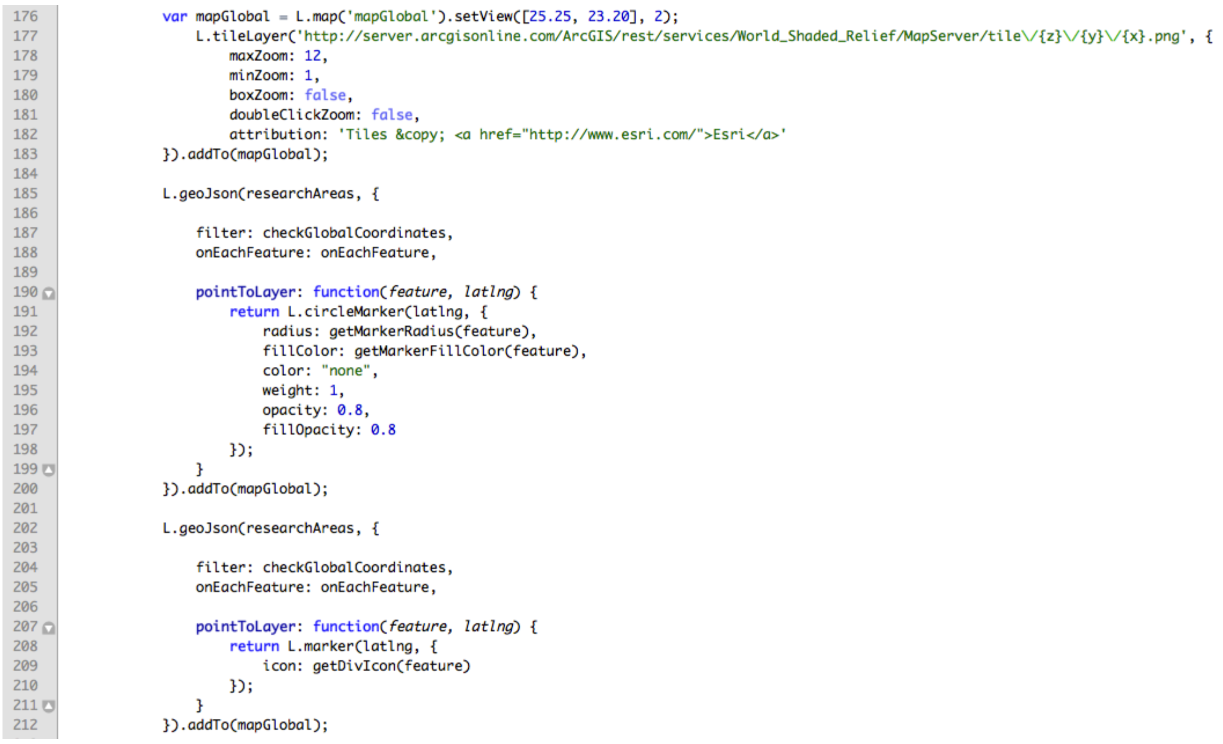
When the home page gets loaded, three java script files relating specifically to the map are sourced in: leaflet.js, latLongJson.js and homePageMaps.js. The first is the java script library that does the actual map rendering, from sourcing in the map tiles to placing the markers on the map. The second file contains a JSON array containing geographic data such as the names of countries and regions, their latitude and longitude, and some additional information that is used to build the GeoJSON object. The last file, homePageMaps.js, contains the functions that serve as the driver for rendering the map. The following outline covers the sequence of those events.

1) The getGeoJsonForMaps() function uses an AJAX request to call the GeoFocusMapLocations.java class. The purpose of this class is to run the SPARQL query that retrieves the names of the countries and regions that researchers have selected as areas of geographic focus as well the number of researchers associated with each area.

2) Once the SPARQL query results are returned to the getGeoJsonForMaps() function, it then parses the results and uses several function calls to build the GeoJSON array that gets used by the Leaflet java script. For example, the getLatLong() function call gets the longitude and latitude of a geographic area from the latLongJson.js file. The GeoJSON array, which is stored in a variable named "researchAreas," takes this format:

```
{ "type": "FeatureCollection",
  "features": [{'geometry': {'type': 'Point', 'coordinates': '-64.0,-34.0'},
                'type': 'Feature',
                'properties': {'mapType': 'global',
                               'popupContent': 'Argentina',
                               'html': '1',
                               'radius': '8,
                               'uri': 'http%3A%2F%2Faims.fao.org%2Faos%2Fgeopolitical.owl%23Argentina'}},
                {'geometry': {'type': 'Point', 'coordinates': '-2.0,54.0'},
                'type': 'Feature',
                'properties': {'mapType': 'global',
                               'popupContent': 'United Kingdom',
                               'html': '6',
                               'radius': '10,
                               'uri':
'http%3A%2F%2Faims.fao.org%2Faos%2Fgeopolitical.owl%23United_Kingdom'}},
                ... ]
}
```

3) Once the researchAreas variable is set, the buildGlobalMap() function is called. The main portion of that function is shown below:



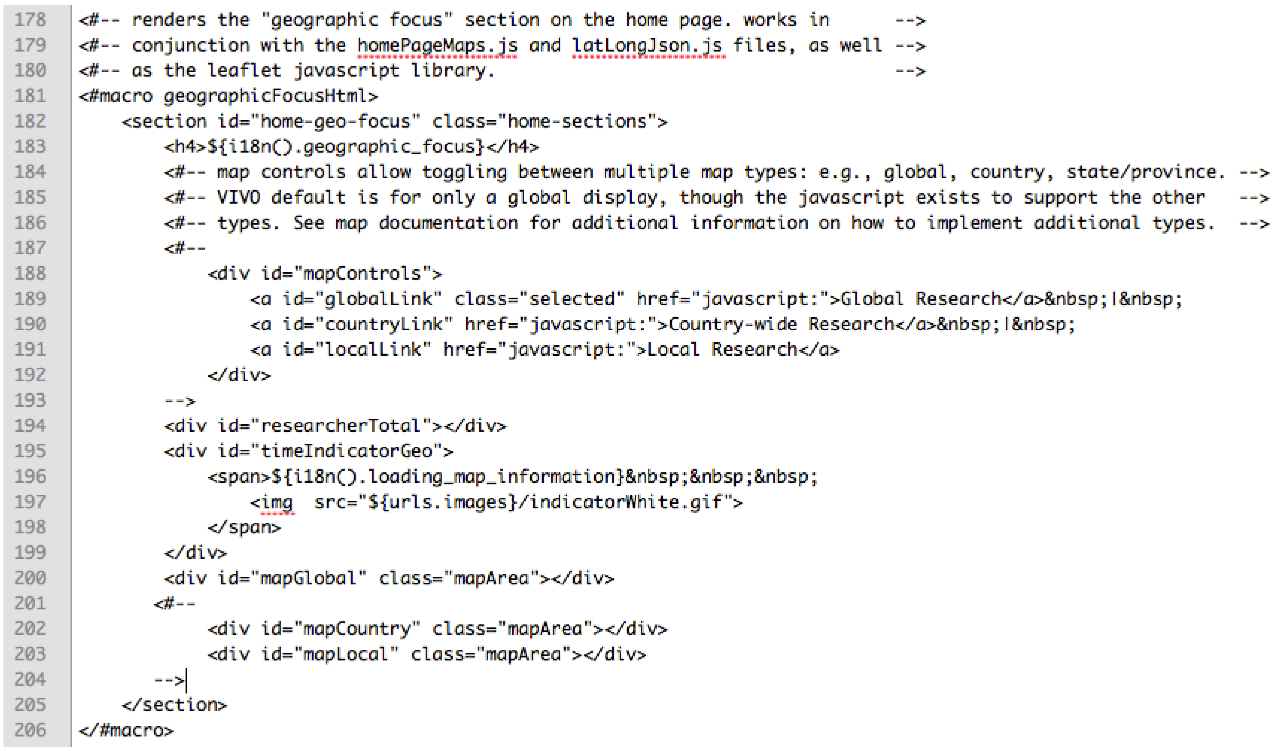
```
176             var mapGlobal = L.map('mapGlobal').setView([25.25, 23.20], 2);
177                 L.tileLayer('http://server.arcgisonline.com/ArcGIS/rest/services/World_Shaded_Relief/MapServer/tile\/{z}\/{y}\/{x}.png', {
178                     maxZoom: 12,
179                     minZoom: 1,
180                     boxZoom: false,
181                     doubleClickZoom: false,
182                     attribution: 'Tiles &copy; <a href="http://www.esri.com/">Esri</a>'
183             }).addTo(mapGlobal);
184
185             L.geoJson(researchAreas, {
186
187                 filter: checkGlobalCoordinates,
188                 onEachFeature: onEachFeature,
189
190                 pointToLayer: function(feature, latlng) {
191                     return L.circleMarker(latlng, {
192                         radius: getMarkerRadius(feature),
193                         fillColor: getMarkerFillColor(feature),
194                         color: "none",
195                         weight: 1,
196                         opacity: 0.8,
197                         fillOpacity: 0.8
198                     });
199                 }
200             }).addTo(mapGlobal);
201
202             L.geoJson(researchAreas, {
203
204                 filter: checkGlobalCoordinates,
205                 onEachFeature: onEachFeature,
206
207                 pointToLayer: function(feature, latlng) {
208                     return L.marker(latlng, {
209                         icon: getDivIcon(feature)
210                     });
211                 }
212             }).addTo(mapGlobal);
```

Here are some key points to note about the previous code:

- The "L." references in the above code are calls to to Leaflet java script library.
- The setView function in line 176 uses latitude and longitude coordinates to center the display of the map.
- Also in line 176, 'mapGlobal' (in L.map('mapGlobal')...) is the name of the <div> element in which Leaflet will render the html for the map.

The geographicFocusHtml Macro

As noted earlier, the lib-home-page.ftl file contains the macros that are used to build the new sections on the home page. Here is the geographicFocusHtml macro:



```
178 <#-- renders the "geographic focus" section on the home page. works in      -->
179 <#-- conjunction with the homePageMaps.js and latLongJson.js files, as well -->
180 <#-- as the leaflet javascript library.                                     -->
181 <#macro geographicFocusHtml>
182     <section id="home-geo-focus" class="home-sections">
183         <h4>${i18n().geographic_focus}</h4>
184         <#-- map controls allow toggling between multiple map types: e.g., global, country, state/province. -->
185         <#-- VIVO default is for only a global display, though the javascript exists to support the other   -->
186         <#-- types. See map documentation for additional information on how to implement additional types.  -->
187         <#--
188             <div id="mapControls">
189                 <a id="globalLink" class="selected" href="javascript:">Global Research</a>&nbsp;|&nbsp;
190                 <a id="countryLink" href="javascript:">Country-wide Research</a>&nbsp;|&nbsp;
191                 <a id="localLink" href="javascript:">Local Research</a>
192             </div>
193         -->
194         <div id="researcherTotal"></div>
195         <div id="timeIndicatorGeo">
196             <span>${i18n().loading_map_information}&nbsp;&nbsp;&nbsp;
197                 <img  src="${urls.images}/indicatorWhite.gif">
198             </span>
199         </div>
200         <div id="mapGlobal" class="mapArea"></div>
201        <#--
202             <div id="mapCountry" class="mapArea"></div>
203             <div id="mapLocal" class="mapArea"></div>
204        -->
205     </section>
206 </#macro>
```

Note line 200: this is the <div> element where Leaflet renders the map.

Customizing the Look of the Map

There are three principal ways to customize the look of the map:

1. Change the source of the map tiles that provide the "atlas"
2. Change the colors of the markers
3. Change the size of the markers

Change the source of the map tiles

This is the most significant modification that you can make. The map currently uses tiles provided by ESRI, which has other map tiles for you to use. Mapquest is another source of free map tiles, as is Google. OpenCloud is a source of map tiles but they charge a small fee.

To change the tiles, you need to update the L.tileLayer definition in the buildGlobalMap() function. This is shown in line 177 above. Simply change the URL to the URL of the service providing your map tiles. (That service may also use a slightly different API.)

Change the colors of the markers

You can change the marker colors in the getMarkerFillColor() function in homePageMaps.js. Note that there are separate colors for countries and regions. If you do change the colors of the markers, you will also have to update the legend that appears in the lower left corner of the map. The circles in this legend are actually image files (map_legend_countries.png and map_legend_regions.png), so you will have to create new image files to match the colors you have chosen for markers.

Change the size of the markers

The size of the markers is the value that is set in the "radius" property in the GeoJSON array. This value is actually calculated in the GeoFocusMapLocations.java class. You can either update this class or add a new function to homePageMaps.js and modify the radius value in that java script file.

Enabling the Country and State/Province Maps

Currently, the home page map section only shows one map view: a global view with markers displayed for regions and countries. However, the code is available to include two additional views, one for a specific country and one for a specific state or province within a country. These are the steps you need to follow to implement the other two map views.

1. Update the geoFocusHtml macro
2. Update the coordinates in the setView() function
3. Update the getResearcherCount() function
4. Update the latLongJson.js file
5. Update the SPARQL query in the GeoFocusMapLocations.java class
6. Update your VIVO data as necessary

Update the geoFocusHtml macro

If you are using multiple map views, than you need to uncomment the mapControls <div> element in the geoFocusHtml macro (`<div id="mapControls">`). If you are only implementing two views (global and country), then you will want to ensure that the "localLink anchor tag is commented out (`<a id="localLink" href="javascript:">`). These anchor tags, along with corresponding java script in the homePageMaps.js file, allow the user to toggle between the implemented map views. (No change to the js file is necessary.)

Next you need to uncomment the <div> elements where the additional map views will be rendered: `<div id="mapCountry" class="mapArea">` and/or `<div id="mapLocal" class="mapArea">`. Again, only uncomment the <div> elements you are implementing.

Update the coordinates in the setView() function

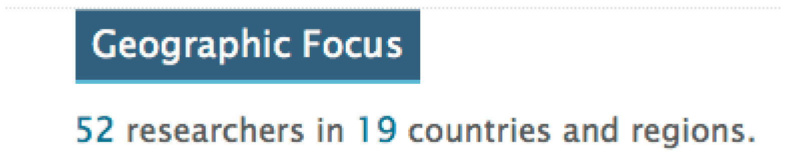
Besides the buildGlobalMap() function (discussed above), the homePageMaps.js file also includes buildCountryMap() and buildLocalMap() functions. These functions are very similar to the buildGlobalMap() function and work in the same way. When you are implementing a country map, you will want the map to be centered on that country.

```
var mapCountry = L.map('mapCountry').setView([46.0, -97.0], 3);
```

The coordinates above, `[46.0, -97.0]`, center the map on the United States. If you want this map to be centered on a different country, you will have to change these coordinates accordingly. The third value in the line of code above, `3`, is the zoom value and sets the default for when the map is loaded. Note that the default zoom value for the global map is `2` and the default for the local map could be any thing from `4` to `8` depending on the location you are displaying.

Update the getResearcherCount() function

For all three types of views, the map includes summary text that shows the total number of researchers and geographical areas in the results, as show below:



Depending on the map views you implement, and the actual country or areas they display, you may want to modify the wording that gets displayed here. This is done in the getResearcherCount() function in of the homePageMaps.js file. (Note that the text here uses internationalization variables, so you may need to update the i18n/all.properties file as well.)

Update the latLongJson.js file

The latLongJson.js file contains data for countries, transnational regions and states within the United States. Therefore, if your installation wants to implement a country map other than the U.S., you will need to update the latLongJson.js file to include the necessary data. For example, if the country to be displayed is Australia, the latLongJson.js file would need to include data on the states and territories of Australia. The JSON array in this file takes data in this format:

```
{"name": "Victoria", "data": {"mapType": "country", "geoClass": "state", "latitude": "-37.4713",
"longitude": "144.7851"}}
```

Note that the mapType corresponds to the map view, in this case "country" as opposed to "global, while the geoClass corresponds (loosely) to the VIVO ontology class. ("Loosely," because the class is actually "StateOrProvince.")

Implementing a state/province map would mean updating the latLongJson.js file to include data for the geographic areas within a state. For U.S. states, examples would include counties, townships and even more general areas such as the Hudson or Mohawk valleys in New York (two areas of geographic focus for Cornell researchers). In this third case the mapType must be set to "local."

Update the SPARQL query in the GeoFocusMapLocations.java class

For performance and practical reasons, the SPARQL query in the GeoFocusLocations.java class excludes states and provinces. (Since only the global map is displayed by default, there is no reason to include state and provinces in the query results.) To update the query to include states and provinces, simply remove this line from the query:

```
FILTER (NOT EXISTS {?location a core:StateOrProvince})
```

If you want to implement a state/province map, you may need to update the query further to ensure that the local geographic areas are included in the query results. Although there are VIVO classes for counties and "populated places," your VIVO installation might have additional refinements to the ontology.

Update your VIVO data as necessary

The SPARQL query in the GeoFocusLocations.java class does not simply return a count for the numbers of researchers that have selected a country (for example) as an area of geographic focus. The query also roll-ups the counts for "child" locations into the "parent" location. For example, if 5 researchers have the U.S. as their area of geographic focus, and another 5 researchers have individual states as their focus, the query will return a count of 10 for the U.S. This is accomplished through the vivo:geographicallyContains object property. Similarly, country counts are rolled up into regional counts through the geo:hasMember object property. *It's possible that you will* *need to curate your VIVO data to ensure that the necessary object property relationships exist in your installation. This is especially true with the local geographical areas.*

8.3.5 Menu and page management

8.3.5.1 Overview

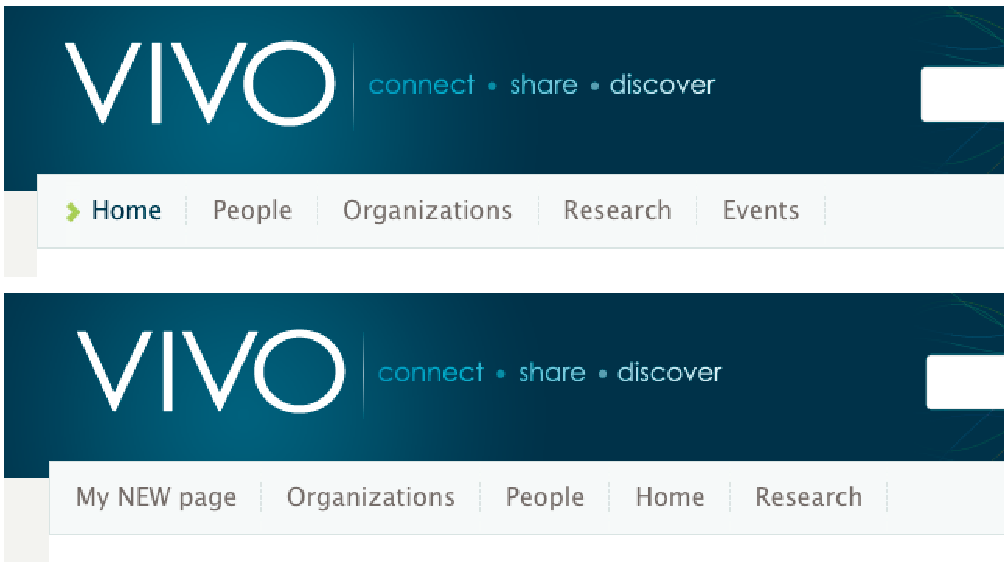
What can it do for you?

- —Create “browse” pages, static pages, or pages that display the results of a query (reports)
- —Remove existing pages
- —Manipulate the page menu

It's easy to surmise that Page Management only allows you to make changes to the menu. And it's true that you can create new menu pages, rearrange the menu, or remove items from it.

But you can also use Page Management to create pages that aren't in the menu. You assign a simple URL to each page, so you can link to them from your other pages. The content of the pages can be simple HTML, the results of a SPARQL query, or a "browse" page for individuals in VIVO.

Before and After



What do you need to know?

- —How to follow the GUI for page management

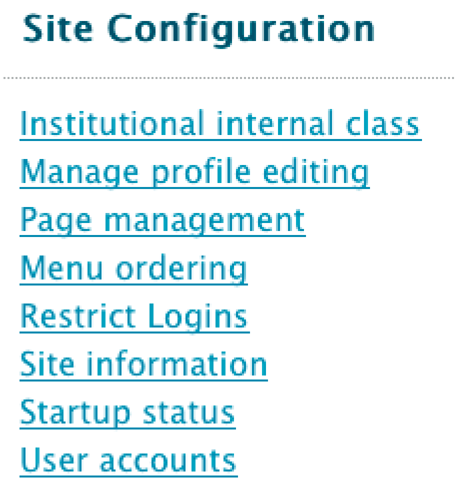
- —Optional – how to write Freemarker templates
- —Optional – how to write SPARQL queries

Getting started

VIVO comes with a set of managed pages, including the ones that you see in the menu on each page.

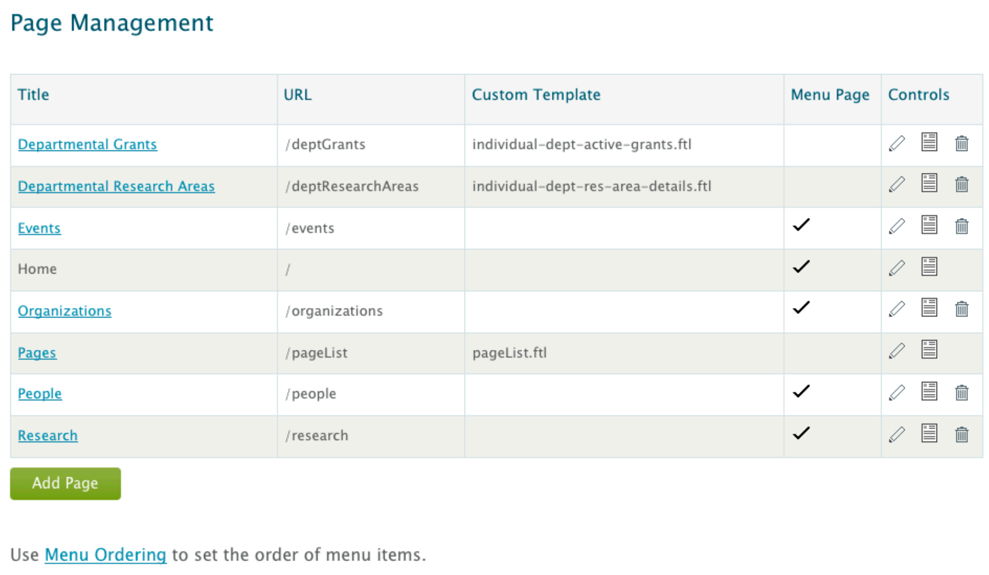
8.3.5.2 What to do

Go to the ***Site Admin*** page, and choose ***Page management***.


Site Configuration

Institutional internal class
Manage profile editing
Page management
Menu ordering
Restrict Logins
Site information
Startup status
User accounts

Use the links provided to create new pages, or edit existing ones.

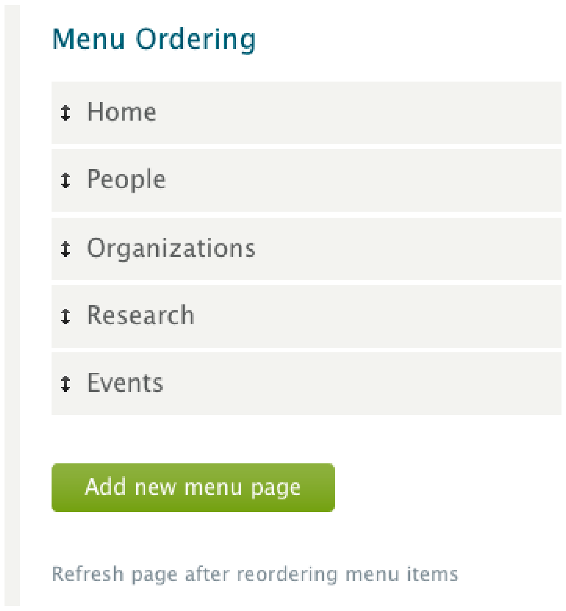

Page Management

Title	URL	Custom Template	Menu Page	Controls
Departmental Grants	/deptGrants	individual-dept-active-grants.ftl		
Departmental Research Areas	/deptResearchAreas	individual-dept-res-area-details.ftl		
Events	/events		✓	
Home	/		✓	
Organizations	/organizations		✓	
Pages	/pageList	pageList.ftl		
People	/people		✓	
Research	/research		✓	

Add Page

Use Menu Ordering to set the order of menu items.

Click on the Menu Ordering link to re-arrange the menu. Drag entries up or down to establish the order you want. When you refresh the page, or go to another, you will see your changes in the menu.



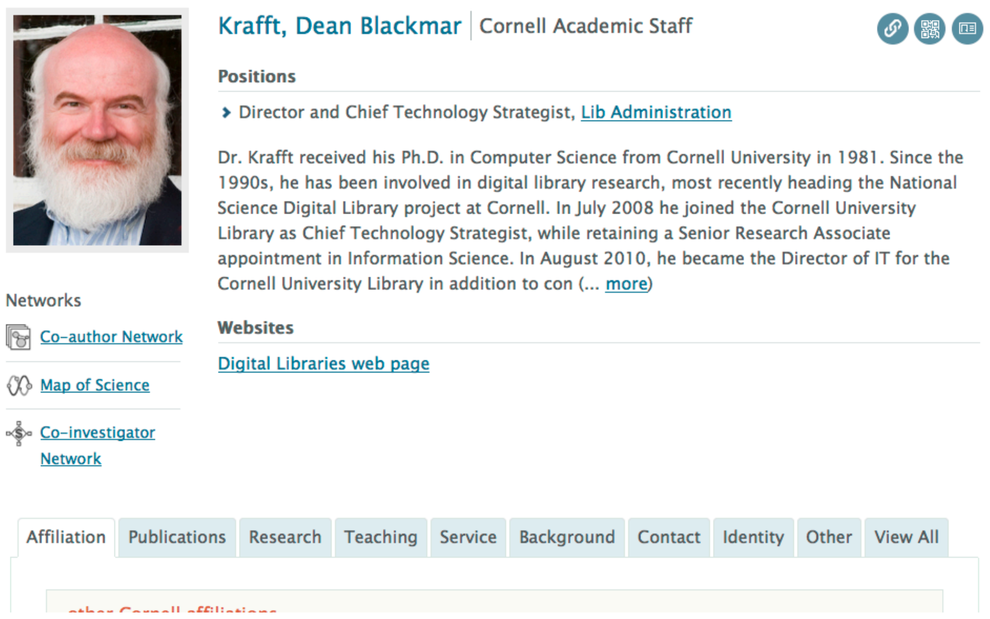
8.3.6 Annotations on the ontology

8.3.6.1 Edit property groups

Overview

Before and After

Rename "Affiliation" to "Allegiances", and change the order of "Publications" and "Research".



Krafft, Dean Blackmar | Cornell Academic Staff

Positions

› Director and Chief Technology Strategist, Lib Administration

Dr. Krafft received his Ph.D. in Computer Science from Cornell University in 1981. Since the 1990s, he has been involved in digital library research, most recently heading the National Science Digital Library project at Cornell. In July 2008 he joined the Cornell University Library as Chief Technology Strategist, while retaining a Senior Research Associate appointment in Information Science. In August 2010, he became the Director of IT for the Cornell University Library in addition to con (... more)

Websites

Digital Libraries web page

Networks

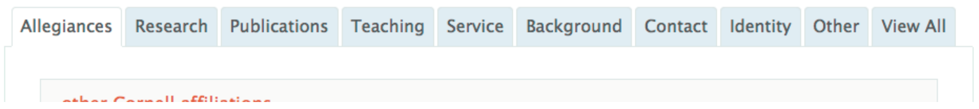
Co-author Network

Map of Science

Co-investigator Network

Affiliation | Publications | Research | Teaching | Service | Background | Contact | Identity | Other | View All

other Cornell affiliations



Allegiances | Research | Publications | Teaching | Service | Background | Contact | Identity | Other | View All

other Cornell affiliations

What do you need to know?

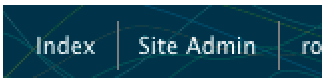
How to follow the GUI for property groups.

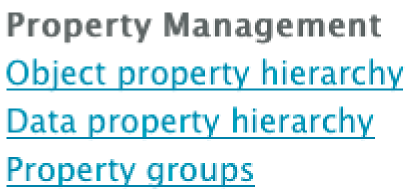
Getting started

VIVO comes with a default set of property groups. You can rename them, reorder them, create new groups or delete existing ones. You can also move properties from one group to another, but that is covered in Edit the appearance of properties (see page 141).

What to do

From the VIVO ***Site Admin*** page, navigate to the ***Property Groups*** page.





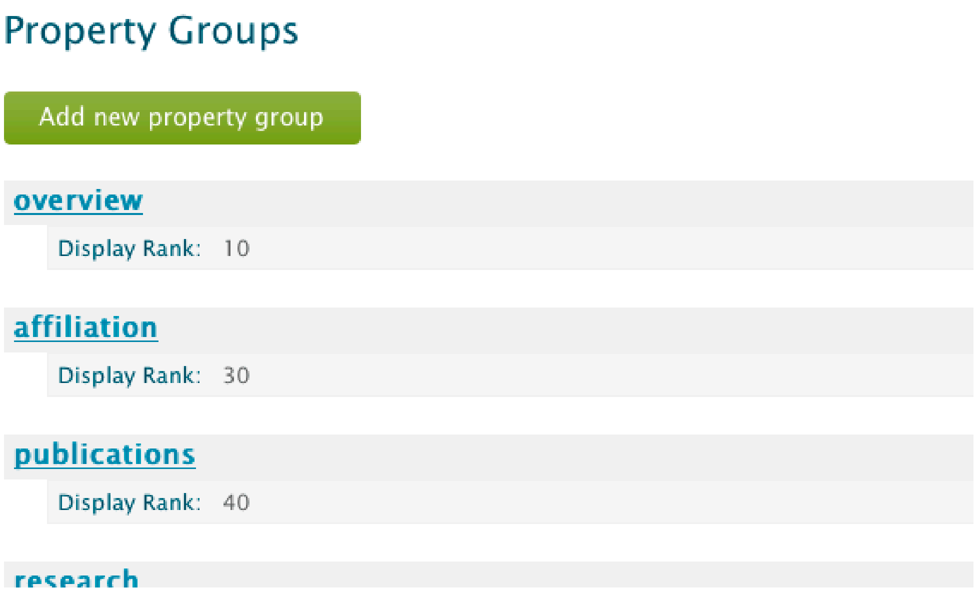
Property Management

Object property hierarchy

Data property hierarchy

Property groups

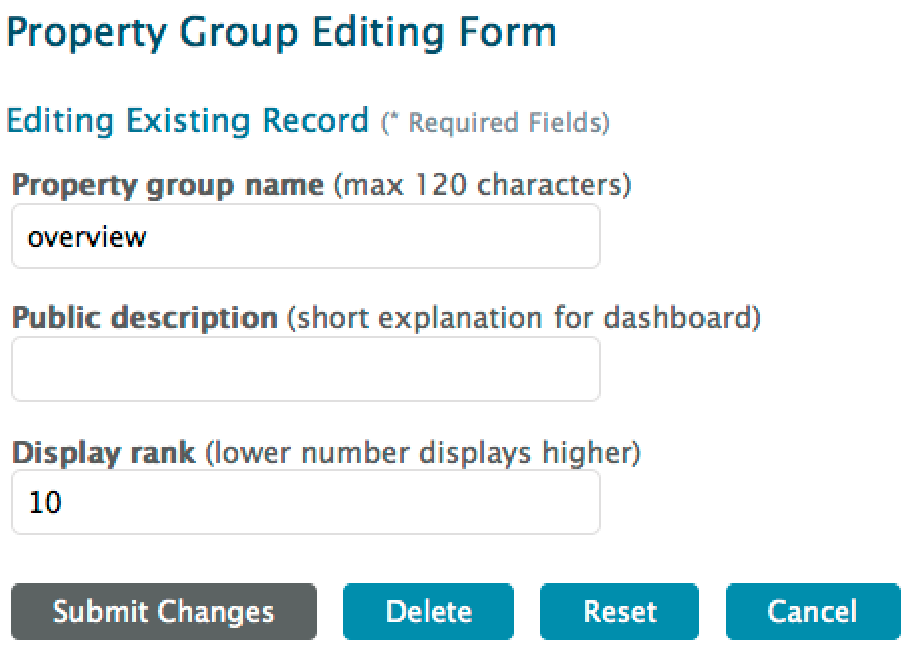
You can add a new property group, or click on the name of an existing group to edit it.


Property Groups

Add new property group

overview	
Display Rank:	10
affiliation	
Display Rank:	30
publications	
Display Rank:	40
research	

You can change the name of a group, change its display rank, or even delete it.


Property Group Editing Form

Editing Existing Record (* Required Fields)

Property group name (max 120 characters)	overview
Public description (short explanation for dashboard)	
Display rank (lower number displays higher)	10

When an profile page is displayed, the property groups are shown in order of ascending display rank.

Properties that aren't included in any of the groups are displayed on the profile page as part of the group named `Other`. This is not an actual property group; it is simply a display convention.

If you delete a property group, the properties that were in it will be displayed in `Other`, until you assign them to new groups.

8.3.6.2 Edit the appearance of properties

Overview

What can it do for you?

Change how VIVO displays the properties of an individual.

For each property, you can change

- the display label
- the public and private descriptions
- which property group it belongs to
- the display rank within the property group
- who can see the values
- who can edit the values
- whether the values will be published in linked open data requests
- whether the display will be collated by sub-properties (object properties only)

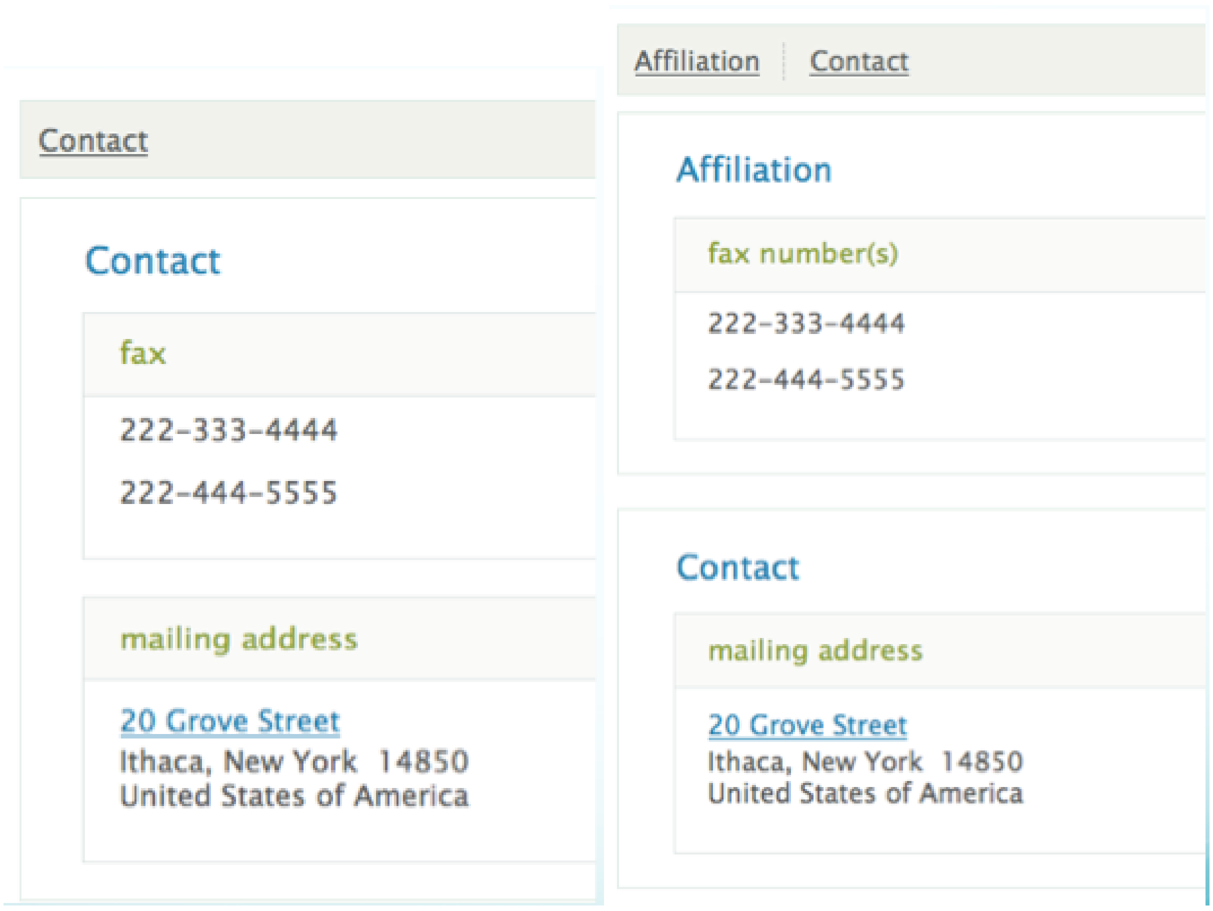
You can also change things like the namespace and parent property, but these are actually changes to the ontology.

Notice that there are necessary differences between the editing options for a data property and those for an object property. Since an object property describes the relationship between two individuals, it is richer than a data property which has only a text value.

The property editing form also allows you to assign a custom entry form to a property, as described in Customize: date entry forms (see page 138)

Before and After

Modify the "fax" property, changing the display label to "fax number(s)", and moving it to the "Affiliation" property group.



Contact

Contact

fax

222-333-4444

222-444-5555

mailing address

20 Grove Street
Ithaca, New York 14850
United States of America

Affiliation | Contact

Affiliation

fax number(s)

222-333-4444

222-444-5555

Contact

mailing address

20 Grove Street
Ithaca, New York 14850
United States of America

What do you need to know?

How to follow the GUI for property editing.—

Getting started

VIVO comes with a default set of properties. You can edit them to suit your display requirements, or make more extensive modifications, by customizing the ontology.

What to do

There are several ways to navigate to the ***Property Editing Form*** for a particular property. Perhaps the most common way is to show the profile page for an individual, turn on ***verbose property display***,

click on the name of the property you want to edit,

from the ***Property Control Panel***, choose to edit the property

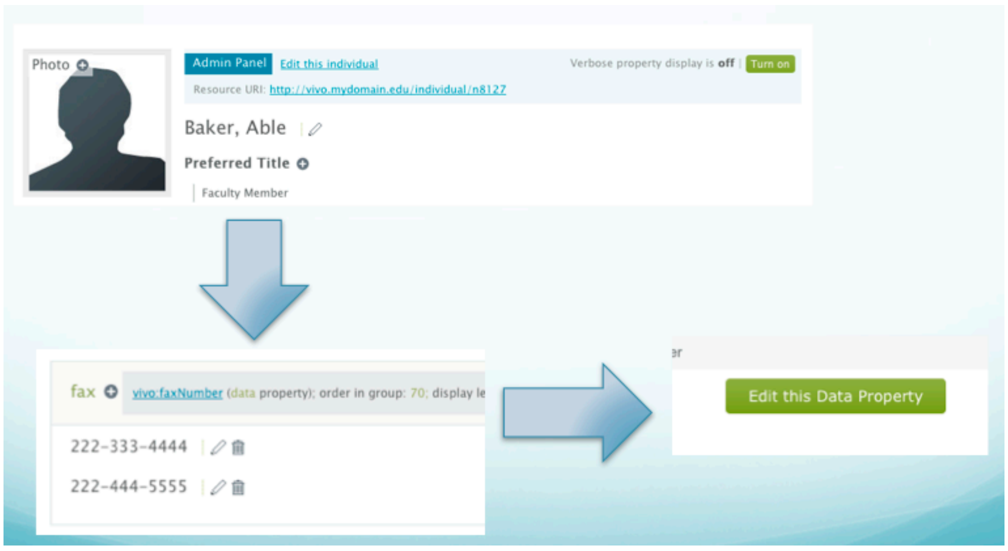
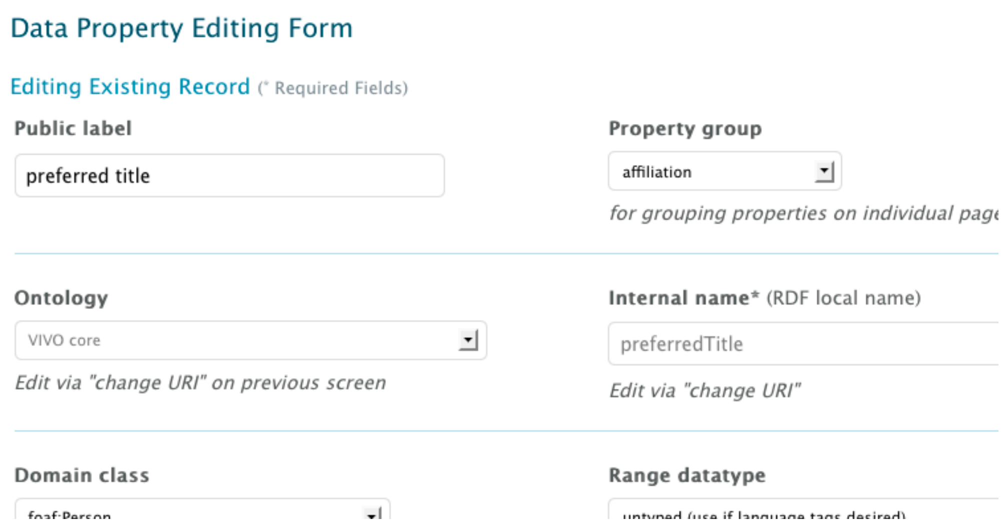


Photo
Admin Panel
Edit this individual
Verbose property display is off | Turn on
Resource URI: http://vivo.mydomain.edu/individual/n8127
Baker, Able
Preferred Title
Faculty Member
fax
vivo:faxNumber (data property); order in group: 70; display le
222-333-4444
222-444-5555
Edit this Data Property

When the property editing form appears, make the changes you want to see, and click on `Submit Changes`. Navigate to a profile page and you will see the effect of your changes immediately.


Data Property Editing Form

Editing Existing Record (* Required Fields)

Public label: preferred title	Property group: affiliation *for grouping properties on individual page*
Ontology: VIVO core *Edit via "change URI" on previous screen*	Internal name* (RDF local name): preferredTitle *Edit via "change URI"*
Domain class: foaf:Person	Range datatype: untyped (use if language tags desired)

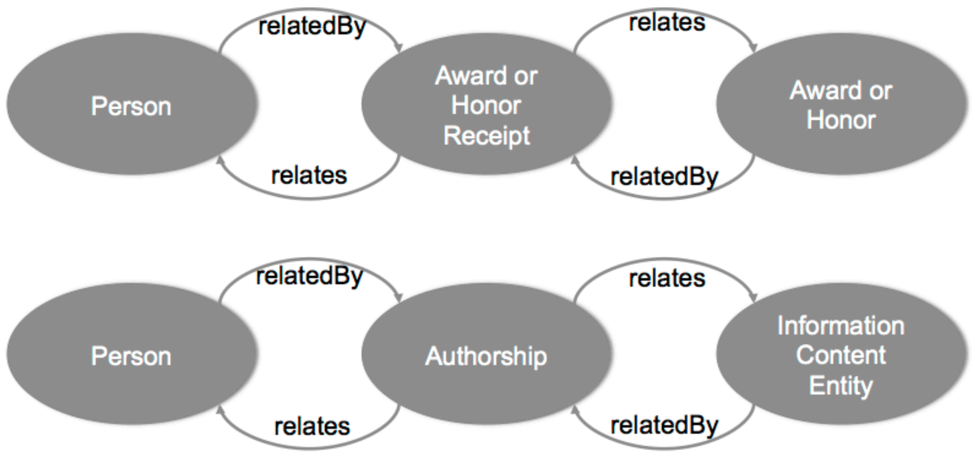
Notes

Properties may appear differently depending to someone who is authorized to edit them. Try logging out to see how your changes will appear to the general user.

8.3.6.3 Create and edit faux properties

Overview

The emphasis in ontology design has been fewer properties connecting more classes. For example, we see that the relationship between a Person, an Authorship, and an Article is very similar to the relationship between a Person, an Award Receipt and an Award.



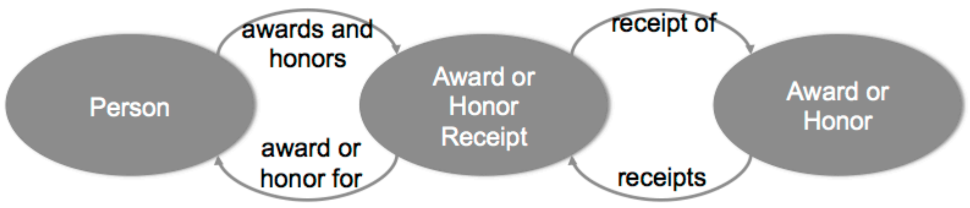
The profile pages in VIVO have been organized by properties. This re-use of properties makes it difficult to organize information on the page. Not only do we want to see at a glance the difference between an authorship and a received award; we may also want to display them in different areas of the page, using different custom views, etc.

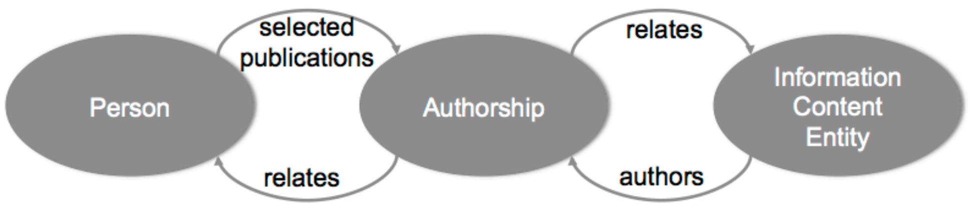
VIVO allows us to create "faux" properties, as restrictions on object properties. The faux property has the same property URI as its base property. It has a domain and a range that are restrictions of the domain and range of the base property. Once we have established these criteria, we can assign display properties to the faux property, just as if it were its own object property with its own URI.

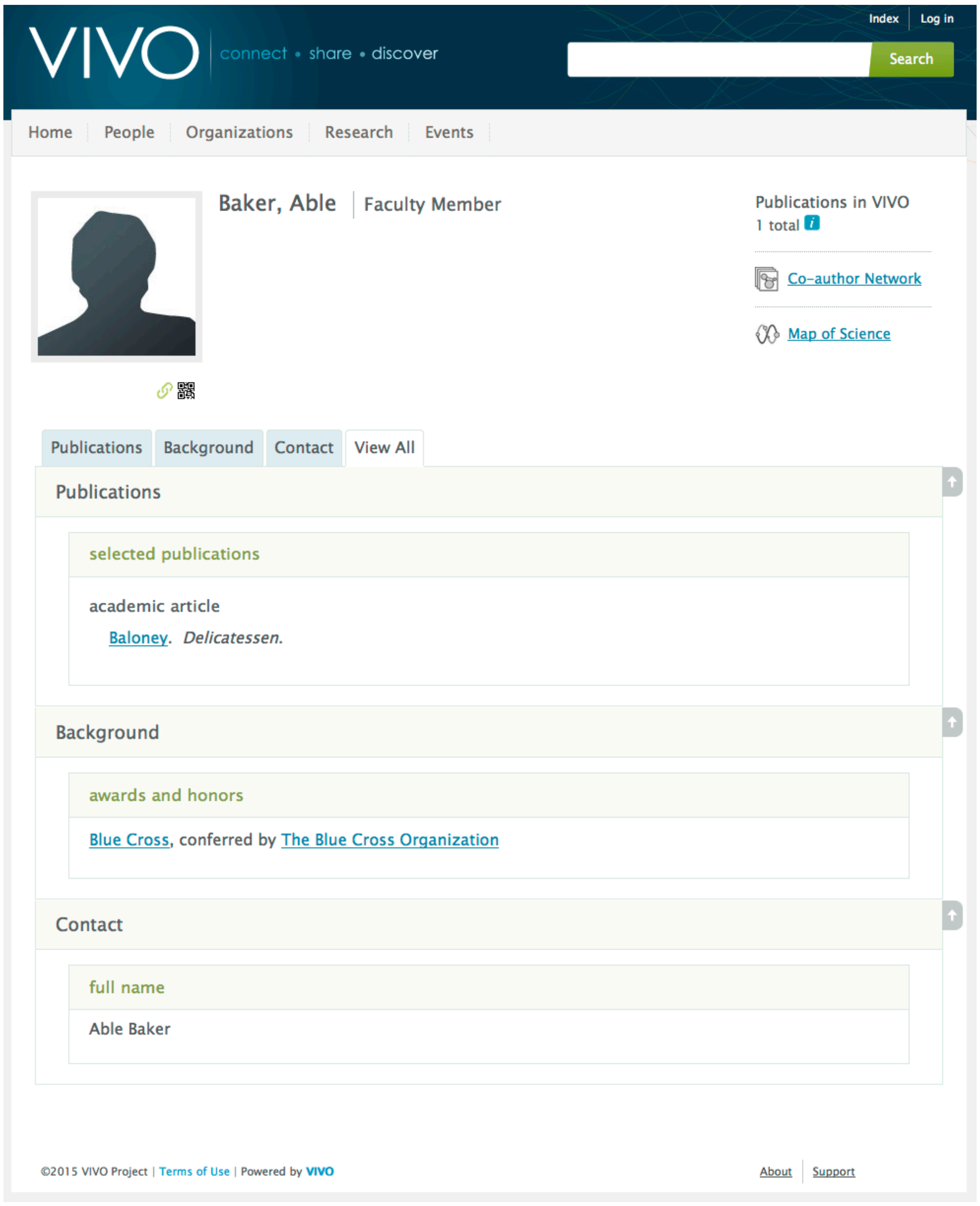
In this way, we can define faux properties as follows:

URI	Domain	Range	label	property group
relatedBy	Person	Award or Honor Receipt	awards and honors	Background
relatedBy	Person	Authorship	selected publications	Publications
relatedBy	Award or Honor	Award or Honor Receipt	receipts	Overview
relatedBy	Information Content Entity	Authorship	authors	Overview
relates	Award or Honor Receipt	Person	award or honor for	Overview
relates	Award or Honor	Award or Honor Receipt	receipt of	Overview

Now, these same relationships display quite differently:


Person
awards and honors
award or honor for
Award or Honor Receipt
receipt of
receipts
Award or Honor


Person
selected publications
relates
Authorship
relates
authors
Information Content Entity



Baker, Able | Faculty Member

Publications in VIVO
1 total

Publications

selected publications

academic article

Baloney. *Delicatessen.*

Background

awards and honors

Blue Cross, conferred by The Blue Cross Organization

Contact

full name

Able Baker

©2015 VIVO Project | Terms of Use | Powered by VIVO   About | Support

In general, it makes sense to partition all of a property into faux properties. Then the base property is set to be invisible (except to the root user), and the faux properties display the desired information.

What do you need to know?

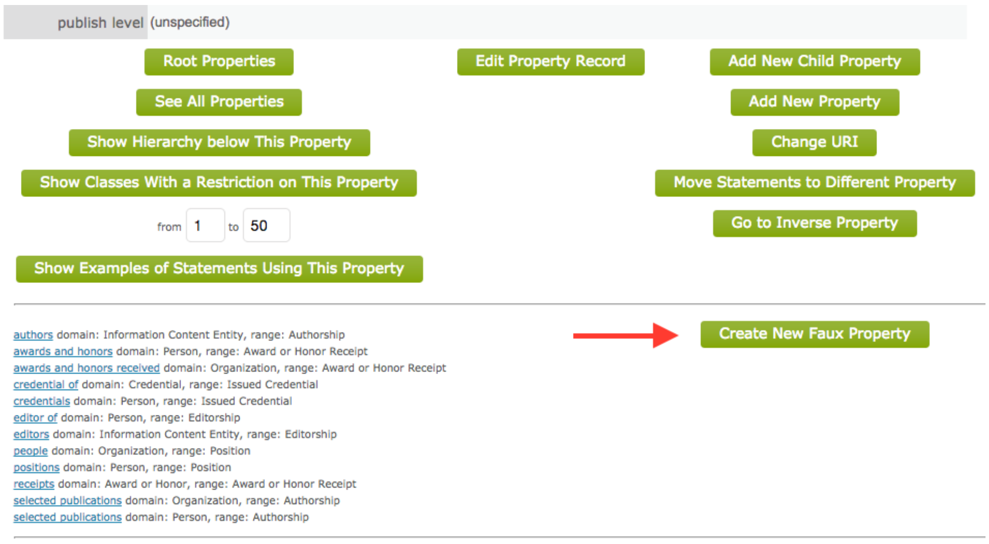
How to follow the GUI for faux properties.

Getting started

VIVO comes with a default set of faux properties. You can edit them to suit your display requirements, or make more extensive modifications.

Creating a faux property

First, navigate to the control panel for the object property you want to create a faux property for. The control panel can be accessed by clicking **Object Property Hierarchy** on the Site Administration Page, then clicking on an object property, i.e. related by. The Create New Faux Property button can be found next to a list of that property's existing faux properties.


publish level (unspecified)
Root Properties
Edit Property Record
Add New Child Property
See All Properties
Add New Property
Show Hierarchy below This Property
Change URI
Show Classes With a Restriction on This Property
Move Statements to Different Property
from 1 to 50
Go to Inverse Property
Show Examples of Statements Using This Property
Create New Faux Property

authors domain: Information Content Entity, range: Authorship
awards and honors domain: Person, range: Award or Honor Receipt
awards and honors received domain: Organization, range: Award or Honor Receipt
credential of domain: Credential, range: Issued Credential
credentials domain: Person, range: Issued Credential
editor of domain: Person, range: Editorship
editors domain: Information Content Entity, range: Editorship
people domain: Organization, range: Position
positions domain: Person, range: Position
receipts domain: Award or Honor, range: Award or Honor Receipt
selected publications domain: Organization, range: Authorship
selected publications domain: Person, range: Authorship

Editing a faux property

There are several ways to navigate to the Faux Property Editing Form. Perhaps the most common way is to show the profile page for an individual, and turn on **verbose property display**.

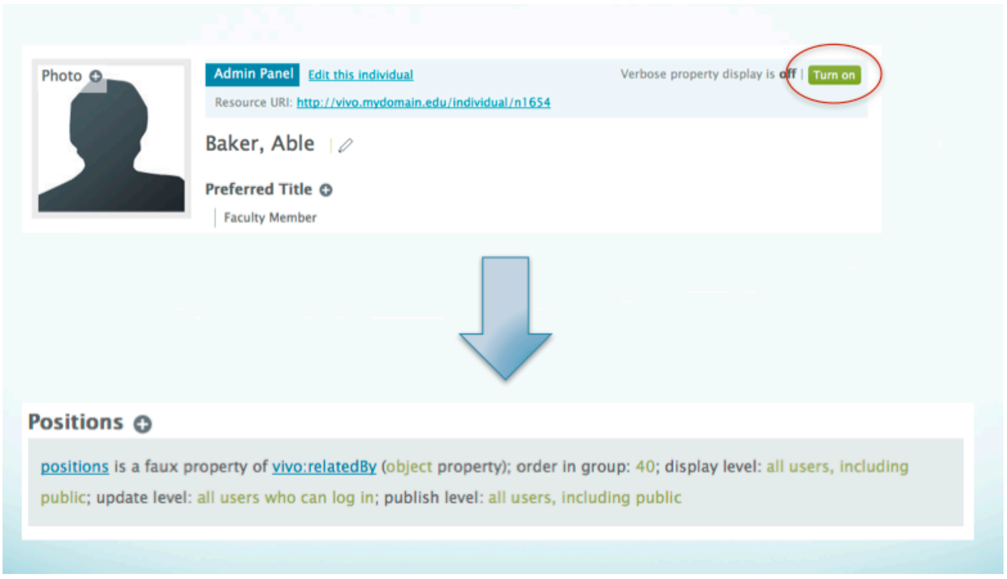
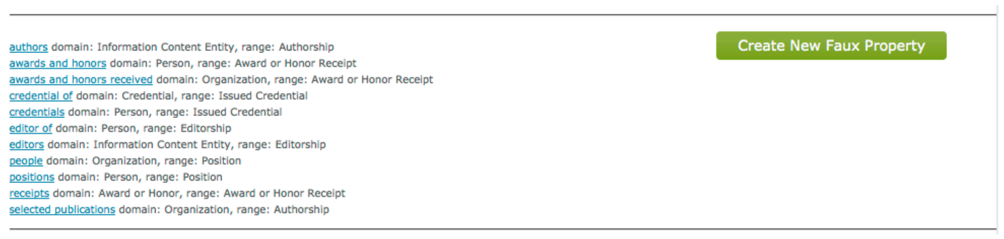


Photo
Admin Panel
Edit this individual
Verbose property display is off | Turn on
Resource URI: http://vivo.mydomain.edu/individual/n1654
Baker, Able
Preferred Title
Faculty Member
Positions

positions is a faux property of vivo:relatedBy (object property); order in group: 40; display level: all users, including public; update level: all users who can log in; publish level: all users, including public

In this example, we see that `positions` is a faux property of `vivo:relatedBy`. If you click on the link for `positions`, you will see the Faux Property Editing Form for `positions`. On the other hand, if you click the link for `vivo:relatedBy`, you will see the Object Property Editing Form for `vivo:relatedBy`. In addition to the parameters for `vivo:relatedBy`, you will also see links to each of the faux properties that are based on it:



authors domain: Information Content Entity, range: Authorship
awards and honors domain: Person, range: Award or Honor Receipt
awards and honors received domain: Organization, range: Award or Honor Receipt
credential of domain: Credential, range: Issued Credential
credentials domain: Person, range: Issued Credential
editor of domain: Person, range: Editorship
editors domain: Information Content Entity, range: Editorship
people domain: Organization, range: Position
positions domain: Person, range: Position
receipts domain: Award or Honor, range: Award or Honor Receipt
selected publications domain: Organization, range: Authorship

From each of these pages, you can modify or delete the faux property that is displayed.

Alternatively, the Faux Property Editing form can be accessed by navigating to the Site Administration Page and clicking **Faux Property Listing**. This will display the complete list of faux properties. Click the title to access the editing form for that faux property.

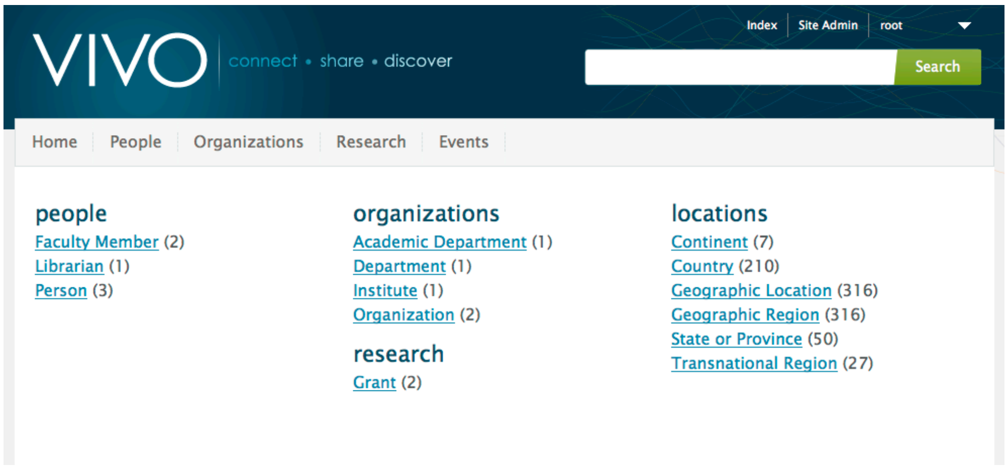
Note: Faux property custom list views are configured in the Faux Property Editing form, rather than as documented in Custom List View Configuration[130] for ontology properties.

8.3.6.4 Edit class groups

Overview

Before and After

Rename "people" to "personnel", and move it to the end of the display order.



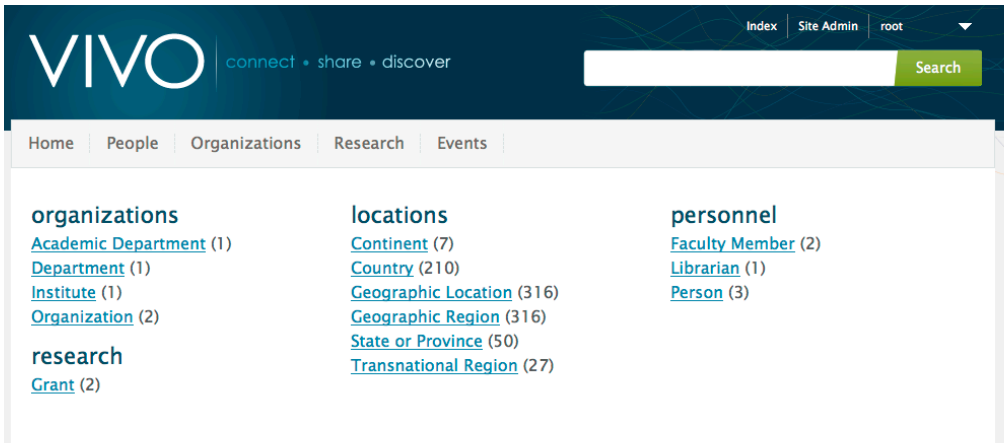
people
Faculty Member (2)
Librarian (1)
Person (3)

organizations
Academic Department (1)
Department (1)
Institute (1)
Organization (2)

research
Grant (2)

locations
Continent (7)
Country (210)
Geographic Location (316)
Geographic Region (316)
State or Province (50)
Transnational Region (27)

130 https://wiki.duraspace.org/display/VIVO/Custom+List+View+Configurations



organizations
Academic Department (1)
Department (1)
Institute (1)
Organization (2)

research
Grant (2)

locations
Continent (7)
Country (210)
Geographic Location (316)
Geographic Region (316)
State or Province (50)
Transnational Region (27)

personnel
Faculty Member (2)
Librarian (1)
Person (3)

What do you need to know?

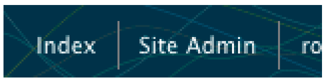
How to follow the GUI for class groups.

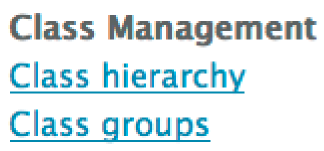
Getting started

VIVO comes with a default set of class groups. You can rename them, reorder them, create new groups or delete existing ones. You can also move classes from one group to another.

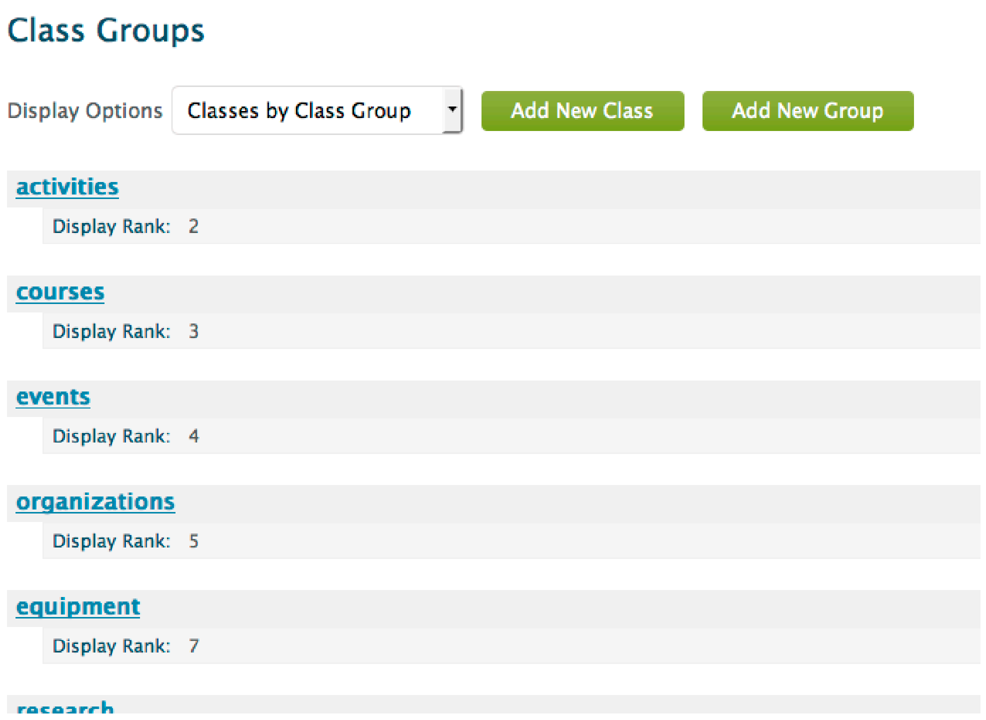
What to do

From the VIVO ***Site Admin*** page, navigate to the ***Class Groups*** page.





You can add a new class group, or click on the name of an existing group to edit it. You can also create a new class, but that involves editing the ontology.


Class Groups

Display Options: Classes by Class Group

activities
Display Rank: 2

courses
Display Rank: 3

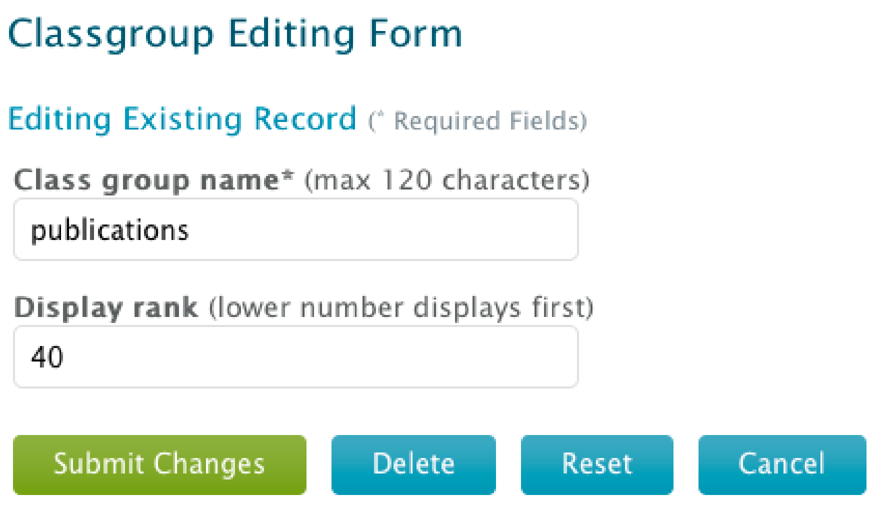
events
Display Rank: 4

organizations
Display Rank: 5

equipment
Display Rank: 7

research

You can change the name of a group, change its display rank, or delete it.


Classgroup Editing Form

Editing Existing Record (* Required Fields)

Class group name* (max 120 characters): publications

Display rank (lower number displays first): 40

When the index page is displayed, the class groups are shown in order of ascending display rank.

Classes that aren't included in any of the groups are not displayed on the index page. If you delete a class group, the classes that were in it will not be displayedon the index page unless you assign them to new groups.

Class groups and their membership can also affect the contents of managed pages. "Browse"-style pages display the contents of some or all of the classes in a single group, so the structure of your class groups will affect how these pages can be configured. Find more information in the section on Menu and page management (see page 136).

Class groups are also used to provide facets in search results.

8.3.6.5 Edit the appearance of classes

Overview

What can it do for you?

Change how VIVO displays a class of individuals.

For each class, you can change

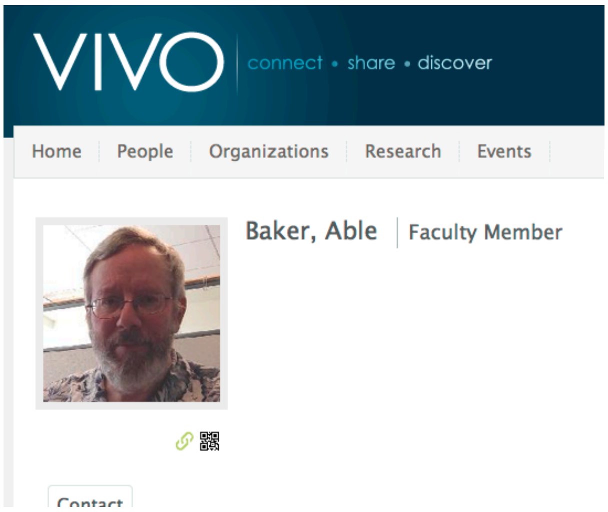
- the display label
- the public and private descriptions
- which class group it belongs to
- who can see the values
- who can edit the values
- whether the values will be published in linked open data requests

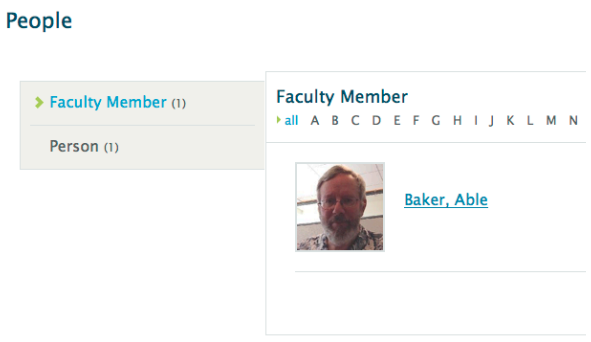
You can also change things like the namespace and parent class, but these are changes to the ontology.

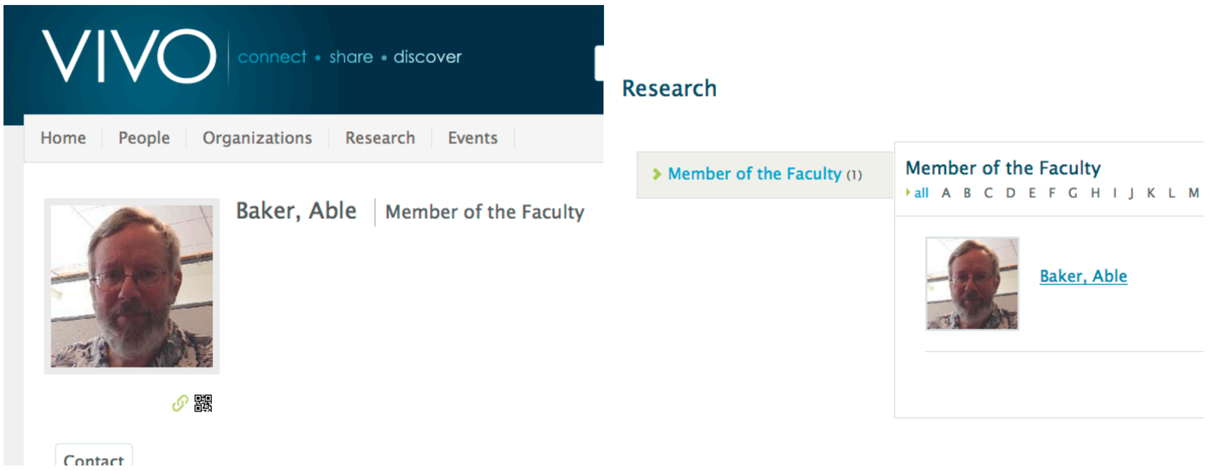
The class editing form also allows you to assign a custom entry form to a class, as described in Custom entry forms (see page 186)

Before and After

Rename "Faculty Member" to "Member of the Faculty", and move it from the "People" class group to the "Research" class group.







What do you need to know?

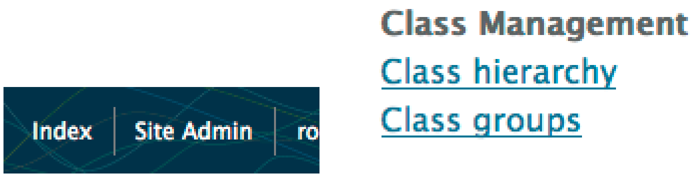
How to follow the GUI for class editing.—

Getting started

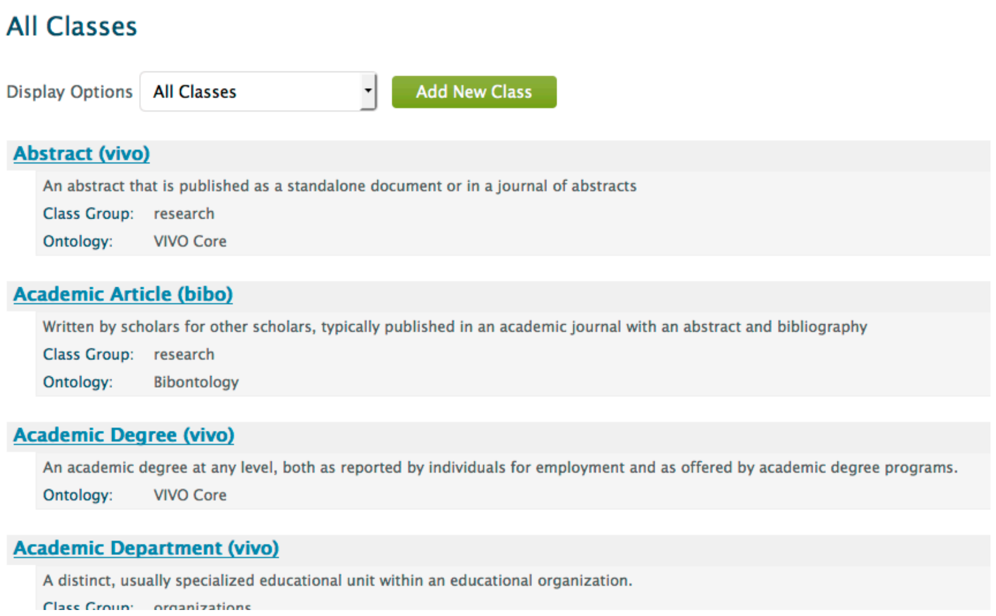
VIVO comes with a default set of classes. You can edit them to suit your display requirements, or make more extensive modifications, by customizing the ontology.

What to do

From the VIVO ***Site Admin*** page, navigate to the ***Class Hierarchy*** page.



If you know the ancestry of the class you want to change, you can navigate to it through the hierarchy. Otherwise, you may want to set the display options to show `All Classes`, and scroll directly to the class you want.


All Classes

Display Options All Classes Add New Class

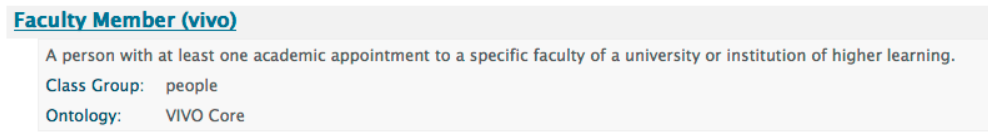
Abstract (vivo)
An abstract that is published as a standalone document or in a journal of abstracts
Class Group: research
Ontology: VIVO Core

Academic Article (bibo)
Written by scholars for other scholars, typically published in an academic journal with an abstract and bibliography
Class Group: research
Ontology: Bibontology

Academic Degree (vivo)
An academic degree at any level, both as reported by individuals for employment and as offered by academic degree programs.
Ontology: VIVO Core

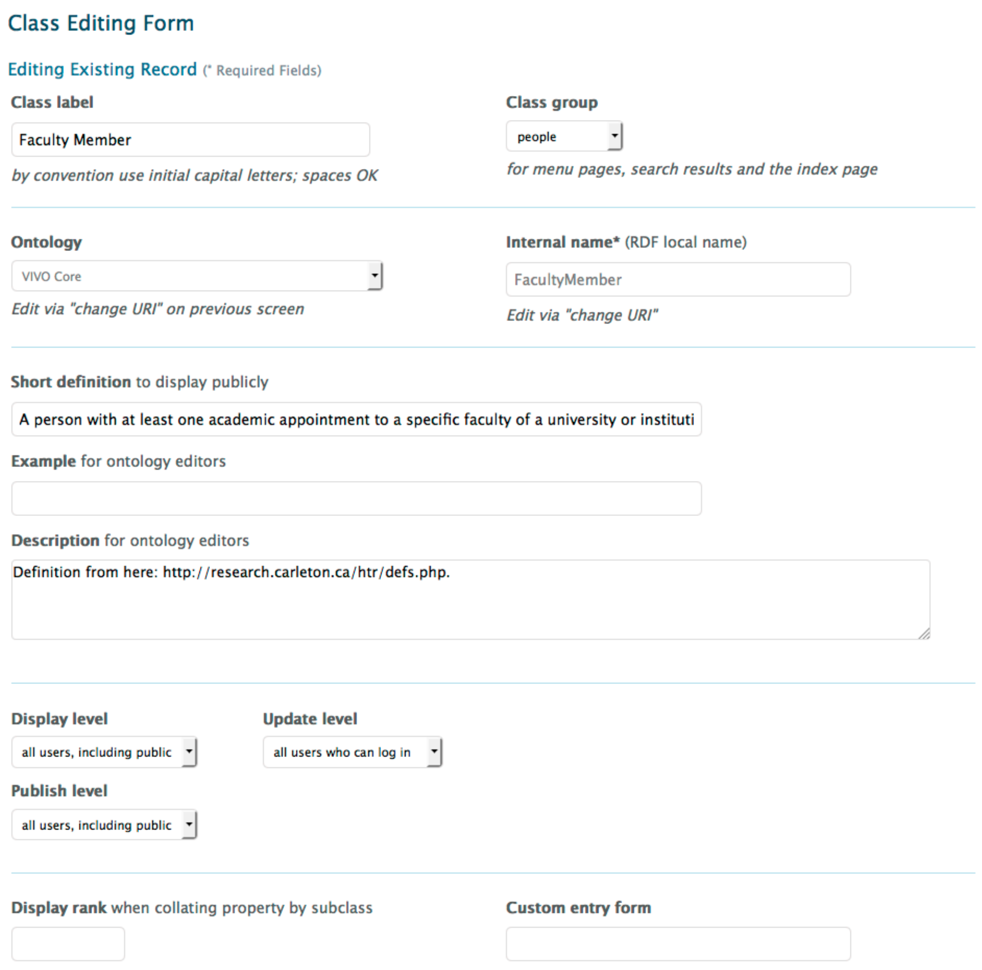
Academic Department (vivo)
A distinct, usually specialized educational unit within an educational organization.
Class Group: organizations

Click on the name of the class you want to edit.


Faculty Member (vivo)
A person with at least one academic appointment to a specific faculty of a university or institution of higher learning.
Class Group: people
Ontology: VIVO Core

This shows you the ***Class Control Panel***. Click on the `Edit Class` button, and you will see the ***Class Editing* Form**.

You can change the class label, or the membership in a class group. You can also change the definition and description of the class.


Class Editing Form

Editing Existing Record (* Required Fields)

Class label: Faculty Member — *by convention use initial capital letters; spaces OK*	**Class group**: people — *for menu pages, search results and the index page*
Ontology: VIVO Core — *Edit via "change URI" on previous screen*	**Internal name*** (RDF local name): FacultyMember — *Edit via "change URI"*

Short definition to display publicly: A person with at least one academic appointment to a specific faculty of a university or instituti

Example for ontology editors:

Description for ontology editors: Definition from here: http://research.carleton.ca/htr/defs.php.

Display level: all users, including public

Update level: all users who can log in

Publish level: all users, including public

Display rank when collating property by subclass:

Custom entry form:

When you have made the changes, click on `Submit Changes`, and navigate to a page where you can see the results.

Notes

There is no need to restart or rebuild VIVO to see the effects of your changes.

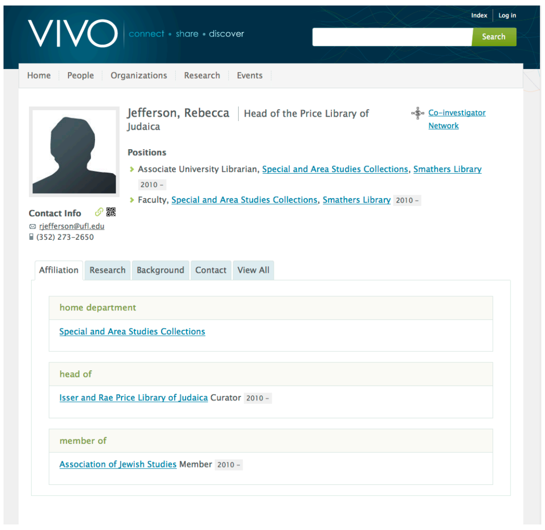
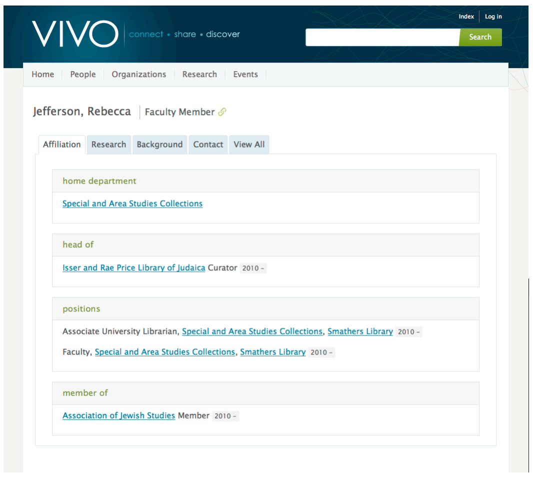
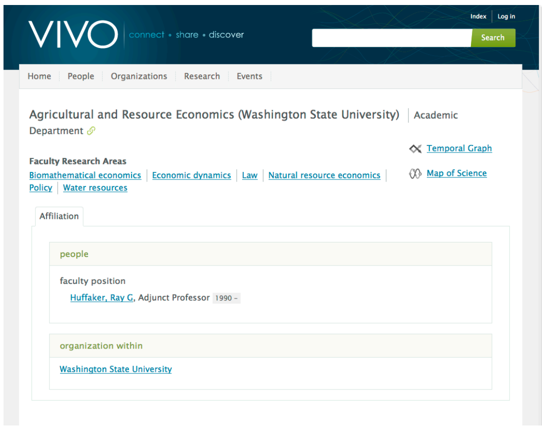
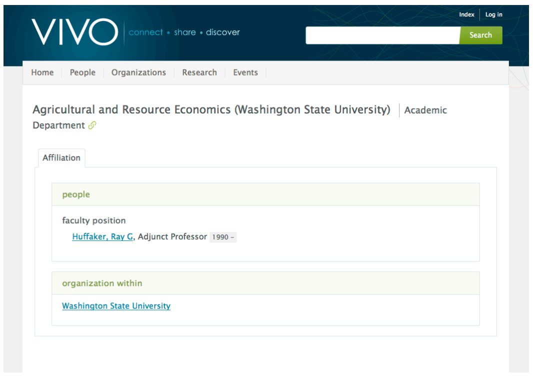
8.3.7 Class-specific templates for profile pages

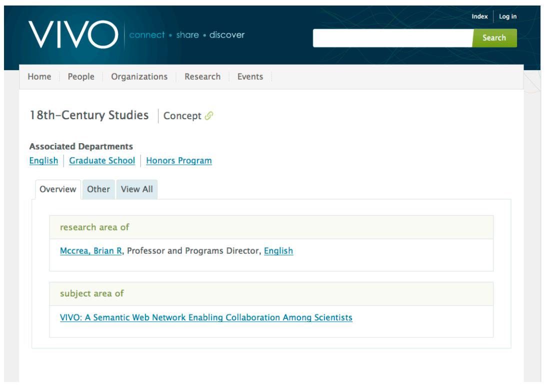
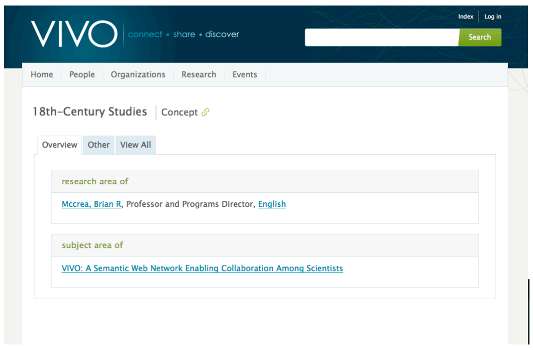
8.3.7.1 Overview

When the profile page for an individual is created, it includes the standard header and footer, but most of the content is built by the Freemarker template `individual.ftl`.

Sometimes you would prefer for a particular class of individuals to have a particular style of profile page.For example, you want the contact information for a `foaf:Person` to appear right below their picture. Or you want to see a link to their co-investigator network near the top of the page.

VIVO lets you specify different templates for different classes of individuals. The standard distribution includes three of those specifications. Here are examples of profile pages for a Person, and Organization, and a Concept, with and without specified templates.

	specified template	**default template**
Person	 Jefferson, Rebecca | Head of the Price Library of Judaica Co-investigator Network **Positions** Associate University Librarian, Special and Area Studies Collections, Smathers Library 2010 - Faculty, Special and Area Studies Collections, Smathers Library 2010 - Contact Info rjefferson@ufl.edu (352) 273-2650 Affiliation | Research | Background | Contact | View All home department Special and Area Studies Collections head of Isser and Rae Price Library of Judaica Curator 2010 - member of Association of Jewish Studies Member 2010 -	 Jefferson, Rebecca | Faculty Member Affiliation | Research | Background | Contact | View All home department Special and Area Studies Collections head of Isser and Rae Price Library of Judaica Curator 2010 - positions Associate University Librarian, Special and Area Studies Collections, Smathers Library 2010 - Faculty, Special and Area Studies Collections, Smathers Library 2010 - member of Association of Jewish Studies Member 2010 -
Organization	 Agricultural and Resource Economics (Washington State University) | Academic Department Temporal Graph Map of Science **Faculty Research Areas** Biomathematical economics | Economic dynamics | Law | Natural resource economics | Policy | Water resources Affiliation people faculty position Huffaker, Ray G, Adjunct Professor 1990 - organization within Washington State University	 Agricultural and Resource Economics (Washington State University) | Academic Department Affiliation people faculty position Huffaker, Ray G, Adjunct Professor 1990 - organization within Washington State University

Concept	 Index Log in connect • share • discover Search Home People Organizations Research Events Overview Other View All 18th-Century Studies Concept Associated Departments English | Graduate School | Honors Program research area of Mccrea, Brian R, Professor and Programs Director, English subject area of VIVO: A Semantic Web Network Enabling Collaboration Among Scientists	 Index Log in connect • share • discover Search Home People Organizations Research Events Overview Other View All 18th-Century Studies Concept research area of Mccrea, Brian R, Professor and Programs Director, English subject area of VIVO: A Semantic Web Network Enabling Collaboration Among Scientists

A specified profile template applies to individuals of the specified class, and to individuals of all sub-classes. So a page specified for `foaf:Person` will also apply to its sub-classes, like `vivo:FacultyMember`.

8.3.7.2 How to do it

There is no page in VIVO that will allow you to set or change these template specifications. Here are two ways to set it up.

Changes on an empty data model

When you start an empty VIVO instance for the first time, it will load the files in the `rdf/tbox/firsttime` directory into the `asserted-tbox` model.

The file `initialTBoxAnnotations.n3`, in this directory, contains the triples that specify these profile templates. They are scattered in the file, and mingled with other triples, but if you look, you can find these statements:

```
foaf:Organization
      vitro:customDisplayViewAnnot
              "individual--foaf-organization.ftl"^^xsd:string .
foaf:Person
      vitro:customDisplayViewAnnot
              "individual--foaf-person.ftl"^^xsd:string .
skos:Concept
      vitro:customDisplayViewAnnot
              "individual--skos-concept.ftl"^^xsd:string .
```

You can remove triples from this file prior to the first startup of VIVO, or add triples to create other template specifications.

Changes to an existing data model

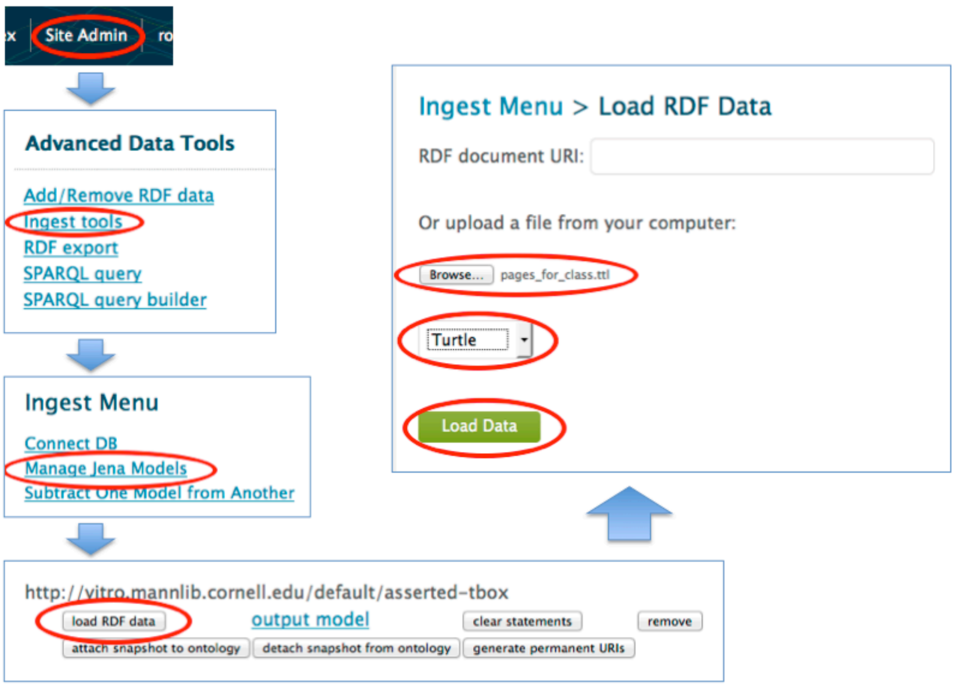
If VIVO has already been started, the files in `rdf/tbox/firsttime` will not be read again. You can use the advanced data tools on the Site Administrator's page to make changes.

Adding specifications

- Create a specialized Freemarker template.
- Prepare an RDF file containing the triples you want to add. Here is an example, in Turtle format:

```
@prefix bibo:    <http://purl.org/ontology/bibo/> .
@prefix vitro:   <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .

bibo:Article
      vitro:customDisplayViewAnnot
            "individual--bibo-article.ftl"^^xsd:string .
```

- Login to VIVO as an admin user
- Follow the header link to the `Site Admin` page
- On the `Site Admin` page, under `Advanced Data Tools`, choose `Ingest tools`.
    - The link to `Add/Remove RDF data` is tempting, but it does not allow us to load into a specific model.
- On the `Ingest Menu` page, choose `Manage Jena Models`.
- In the list of available models, locate the controls for `http://vitro.mannlib.cornell.edu/default/asserted-tbox`, and choose `load RDF data`.
- On the `Load RDF Data` page, use the `Browse` control to locate your RDF file, and select the type of RDF format you used. Click the `Load Data` button.



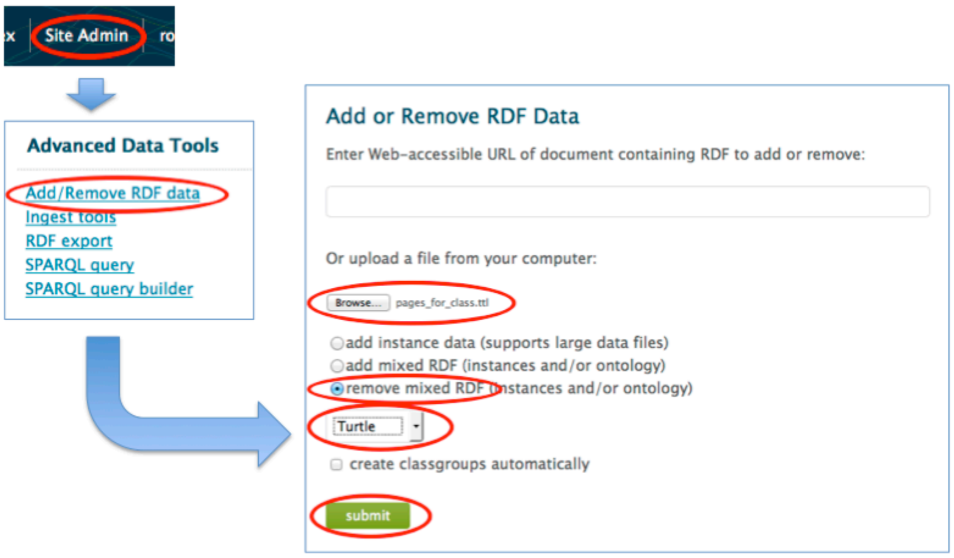
If there is a problem with the load, you will see a screen that shows an error message. Unfortunately, if the load is successful, you will see no indication. You will simply be returned to the list of available models.

You must restart VIVO to see the effect of your changes.

Removing specifications

- Create an RDF file containing the triples you want to remove. In this example, we will remove the triple that was added above, so we will use the same file.
- Login to VIVO as an admin user.
- Follow the header link to the `Site Admin` page.

- On the `Site Admin` page, under `Advanced Data Tools`, choose `Add/Remove RDF data`.
- On the `Add or Remove RDF Data` page, use the `Browse` control to locate your RDF file, choose to `remove mixed RDF`, and select the type of RDF format you used. Click the `submit` button.



Add or Remove RDF Data

Enter Web-accessible URL of document containing RDF to add or remove:

Or upload a file from your computer:

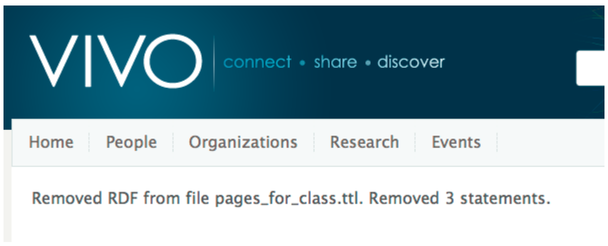
pages_for_class.ttl

( ) add instance data (supports large data files)
( ) add mixed RDF (instances and/or ontology)
(●) remove mixed RDF (instances and/or ontology)

Turtle

☐ create classgroups automatically

If there is a problem with your data file, you will see a screen showing an error message. If the removal is successful, you will see a message like the following:



Removed RDF from file pages_for_class.ttl. Removed 3 statements.

Note that the message tells you how many triples you asked to have removed, without regard to whether they actually existed in the data model.

You must restart VIVO to see the effect of your changes.

Changing specifications

There is no direct way to replace triples in the data model. Use the preceding steps to remove the triples you don't want, and to add new triples to replace them.

Other mechanisms

Expert VIVO users will be aware of many other ways of adding or removing triples.

Remember that VIVO must be restarted, since the list of specific templates is created at startup.

8.3.8 Excluding Classes from the Search

8.3.8.1

8.3.8.2 Overview

A VIVO/Vitro instance can be customized to exclude a class of individuals from the search index. All instances of a class can be excluded from the search index by adding a vitroDisplay:excludeClass[131] property between vitroDisplay:SearchIndex and the URI of the class that you would like to exclude. This will have the effect of not displaying any individual with this class in search results, on the index page, in browse pages and as options for entry in some forms. The search exclusions are controlled by RDF statements in the display model.

8.3.8.3 Steps to create a new search exclusion

1. Create a file with your new exclusion
2. In your deploy directories, place that file in either `vivo/rdf/display/everytime` or `vitro/webapp/rdf/display/everytime`
3. Stop Tomcat, deploy, and restart Tomcat
4. Login to VIVO as an admin and rebuild the search index.

8.3.8.4 Example on the contents of an RDF file to define exclusions

```
@prefix vitroDisplay: <http://vitro.mannlib.cornell.edu/ontologies/display/1.1#> .

vitroDisplay:SearchIndex
    vitroDisplay:excludeClass <http://example.org/ns/ex/Hat> .
```

8.3.9 Custom List View Configurations

131 http://vitroDisplayexcludeClass

8.3.9.1 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.

8.3.9.2 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. (see page 143)

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[132] 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.

132 http://corehasRole

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.

8.3.9.3 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>
```

8.3.10 Creating short views of individuals

8.3.10.1 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.

8.3.10.2 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                   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
       """ .
```

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 `?individualUrl` 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.

8.3.10.3 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.

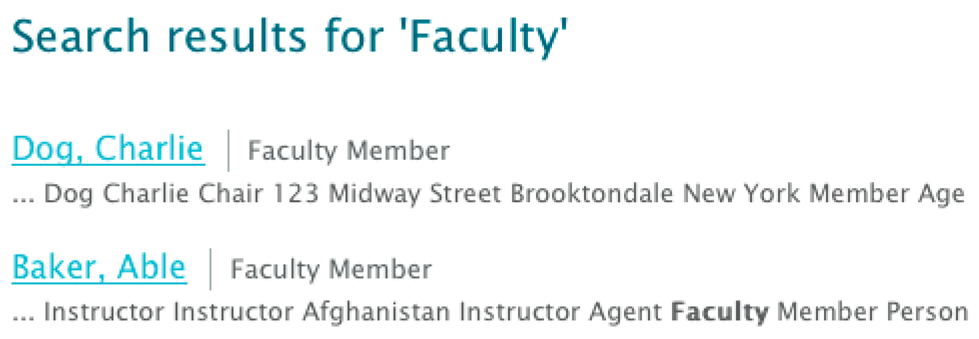
If the FacultyMember is not a member of a department, the short view should just omit the name, without causing an error.

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:

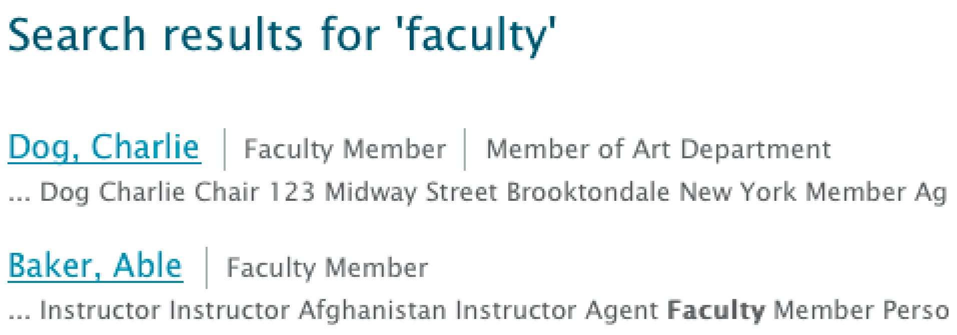
```
vivo:FacultyMember
    display:hasCustomView   mydomain:facultySearchView .

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

mydomain:facultyDepartmentDG
    a                   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
        """ .
```

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

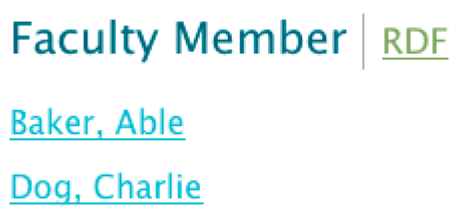
INDEX example

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:



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
       """ .
```

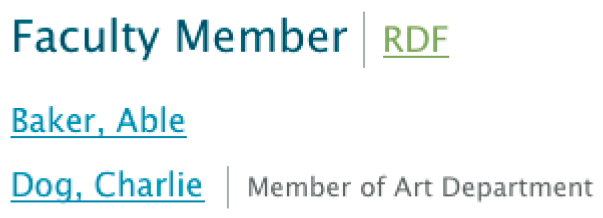
Note that the DataGetter is the same as in the previous example. If two custom short views want to use the same DataGetter, there is no need to code it twice. If both of these examples are tried at the same time, the two custom short views would refer to the same DataGetter.

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

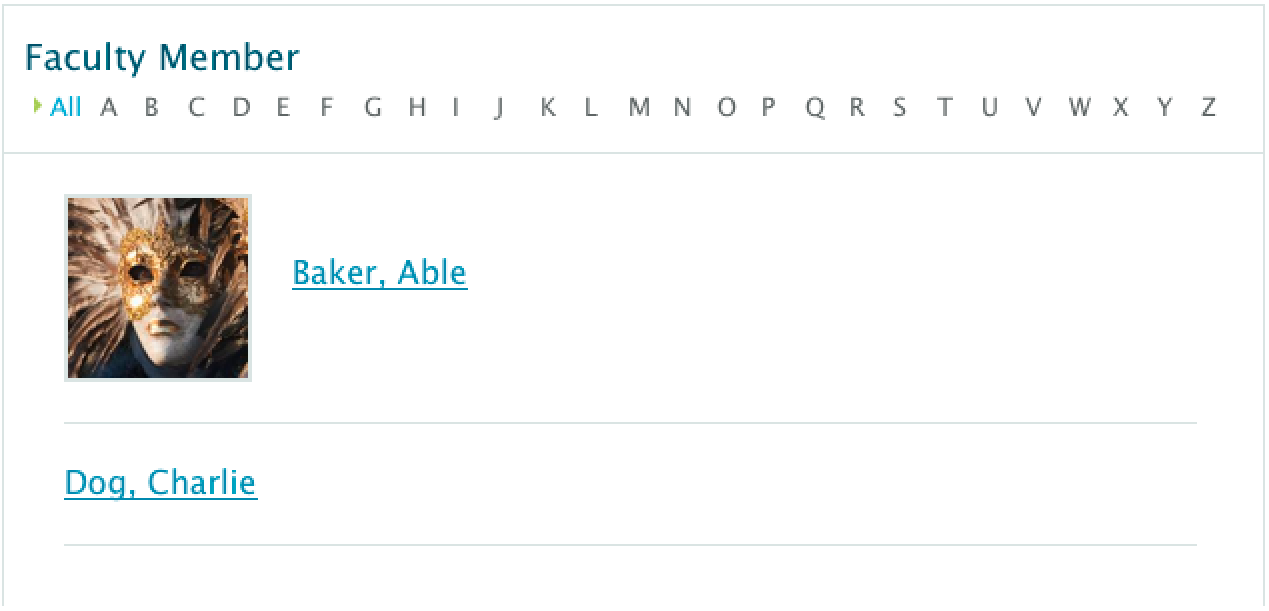
The default short view for the `BROWSE` context looks like this:

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

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

<#if (individual.thumbUrl)??>
    <img src="${individual.thumbUrl}" width="90" alt="${individual.name}" />
    <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>
```

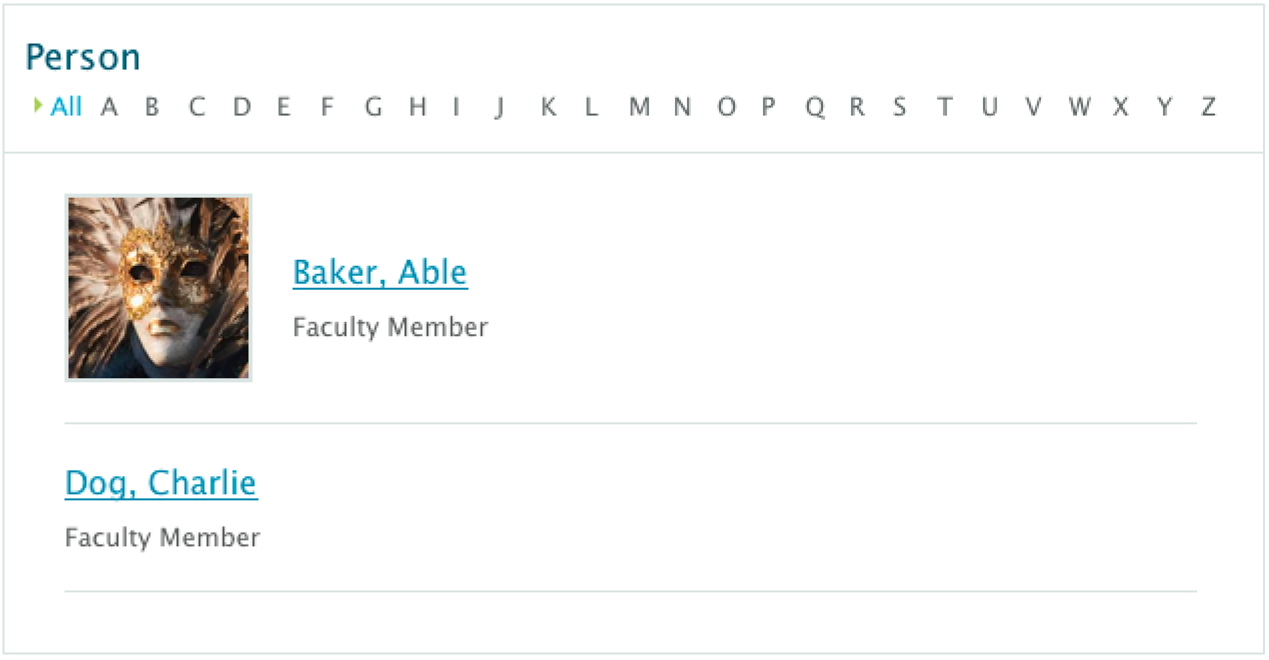
```
</li>
```

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:



Or this:



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                   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
       """ .

mydomain:facultyPreferredTitleDG
    a                  datagetters: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.

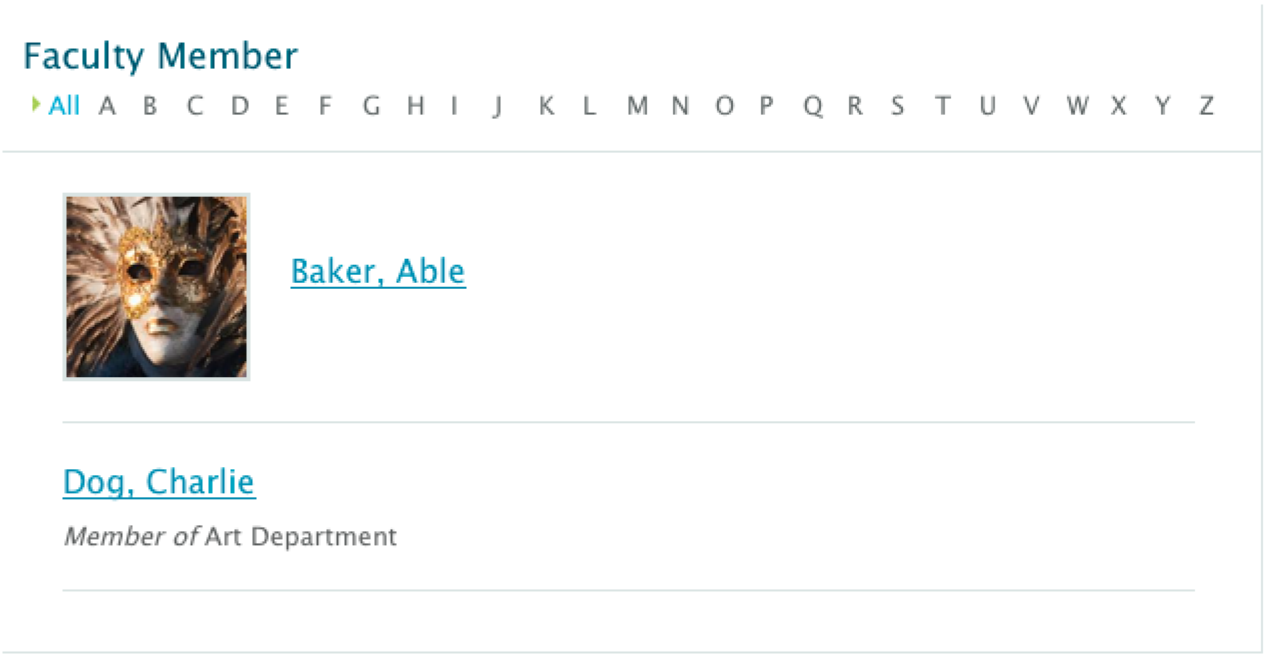
Also note that this short view uses two DataGetters. One stores its results in "details" and the other stores its results in "extra", so the freemarker template will have access to both sets of results.

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)??>
    <img src="${individual.thumbUrl}" width="90" alt="${individual.name}" />
    <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

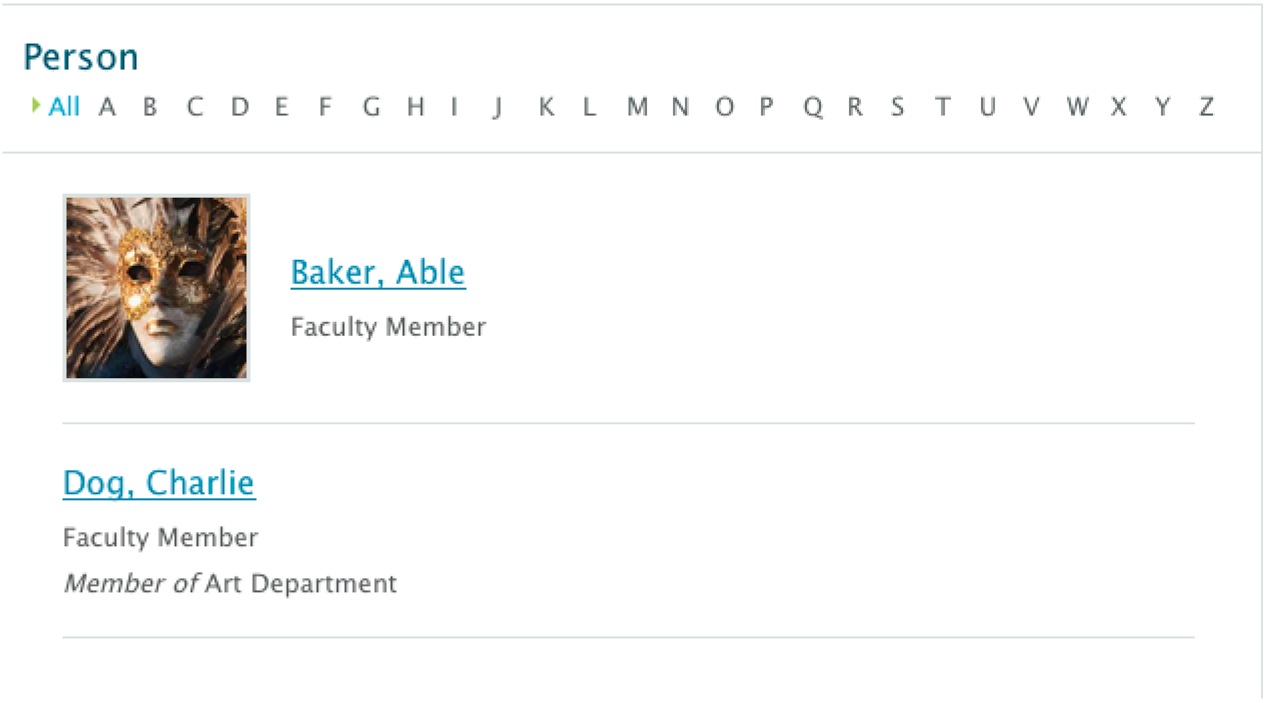
All A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

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

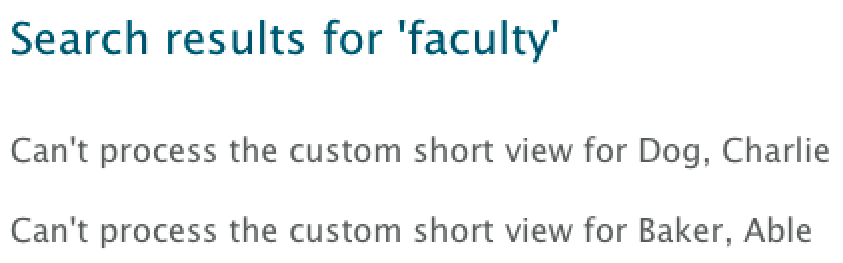
Member of Art Department

8.3.10.4 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:

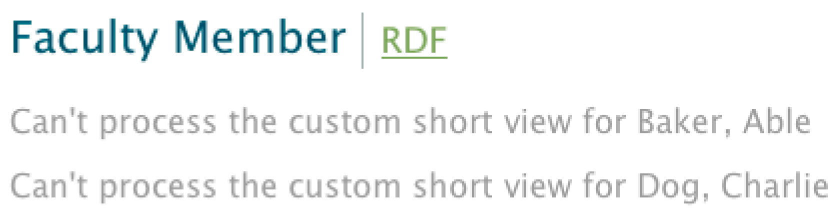


Search results for 'faculty'

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

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

In an index page:

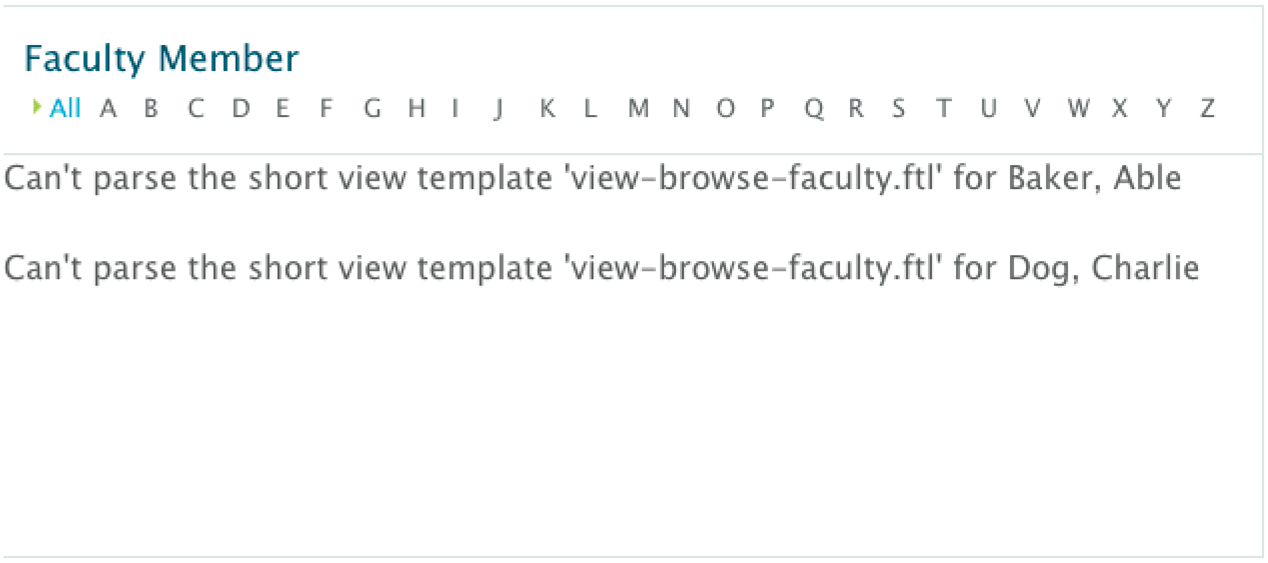


Faculty Member | RDF

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

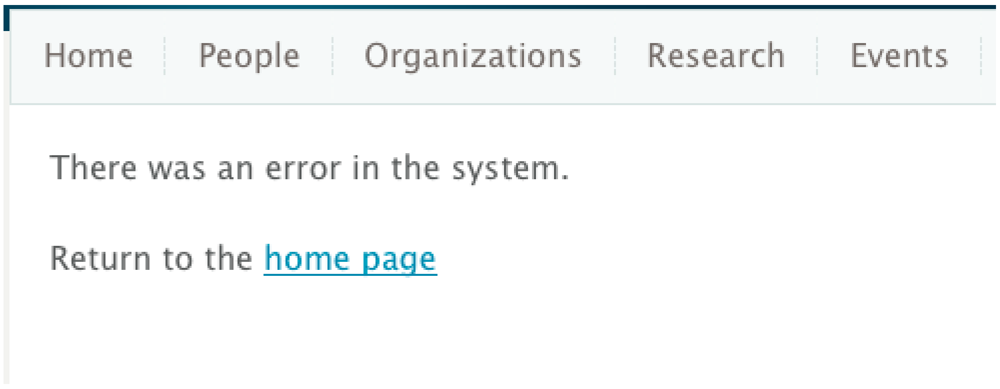
Errors in the Query?

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:



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`

8.3.10.5 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 Indvidual 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`

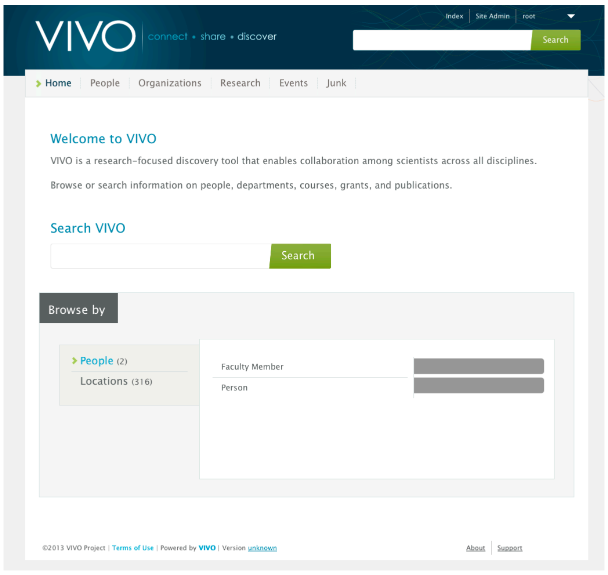
8.3.11 Creating a custom theme

8.3.11.1 Overview

What can it do for you?

Change the "look and feel" of your VIVO installation. Change the styling, the images, the layout, the text, and more. Modify the header and footer on all pages.

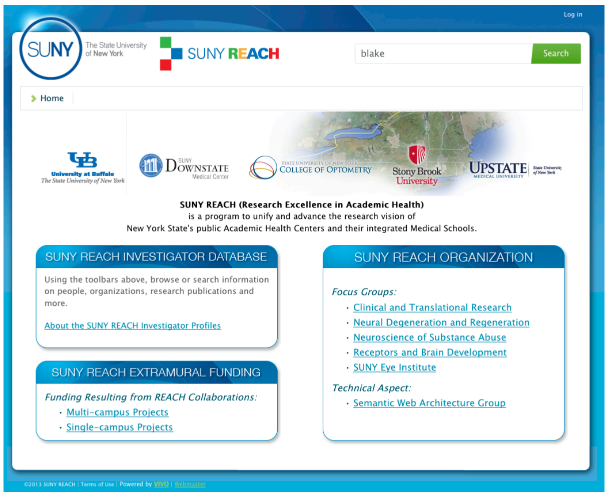
Before and After



VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

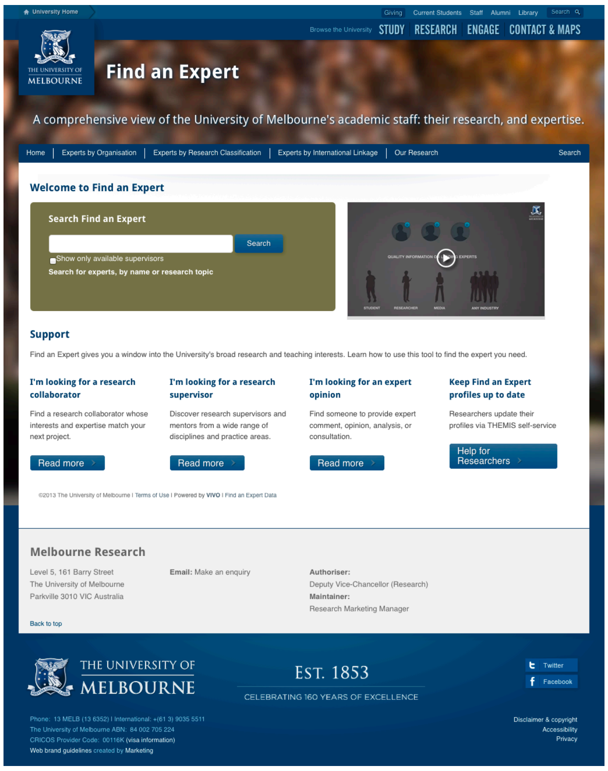
©2013 VIVO Project | Terms of Use | Powered by VIVO | Version unknown


Log in
SUNY The State University of New York
SUNY REACH
blake
Search
Home
University at Buffalo
Downstate Medical Center
College of Optometry
Stony Brook University
Upstate Medical University
SUNY REACH INVESTIGATOR DATABASE
About the SUNY REACH Investigator Profiles
SUNY REACH ORGANIZATION
Focus Groups:
Clinical and Translational Research
Neural Degeneration and Regeneration
Neuroscience of Substance Abuse
Receptors and Brain Development
SUNY Eye Institute
Technical Aspect:
Semantic Web Architecture Group
SUNY REACH EXTRAMURAL FUNDING
Funding Resulting from REACH Collaborations:
Multi-campus Projects
Single-campus Projects

SUNY REACH (Research Excellence in Academic Health) is a program to unify and advance the research vision of New York State's public Academic Health Centers and their integrated Medical Schools.

Using the toolbars above, browse or search information on people, organizations, research publications and more.

©2013 SUNY REACH | Terms of Use | Powered by VIVO | Webmaster


University Home
Giving | Current Students | Staff | Alumni | Library | Search
Browse the University STUDY RESEARCH ENGAGE CONTACT & MAPS
THE UNIVERSITY OF MELBOURNE
Find an Expert
Home | Experts by Organisation | Experts by Research Classification | Experts by International Linkage | Our Research | Search
Welcome to Find an Expert
Search Find an Expert
Search
Show only available supervisors
Search for experts, by name or research topic
STUDENT RESEARCHER MEDIA ANY INDUSTRY
Support
Read more
Help for Researchers
Back to top
THE UNIVERSITY OF MELBOURNE
EST. 1853
CELEBRATING 160 YEARS OF EXCELLENCE
Twitter
Facebook
Disclaimer & copyright
Accessibility
Privacy

A comprehensive view of the University of Melbourne's academic staff: their research, and expertise.

Find an Expert gives you a window into the University's broad research and teaching interests. Learn how to use this tool to find the expert you need.

I'm looking for a research collaborator	I'm looking for a research supervisor	I'm looking for an expert opinion	Keep Find an Expert profiles up to date
Find a research collaborator whose interests and expertise match your next project.	Discover research supervisors and mentors from a wide range of disciplines and practice areas.	Find someone to provide expert comment, opinion, analysis, or consultation.	Researchers update their profiles via THEMIS self-service

©2013 The University of Melbourne | Terms of Use | Powered by VIVO | Find an Expert Data

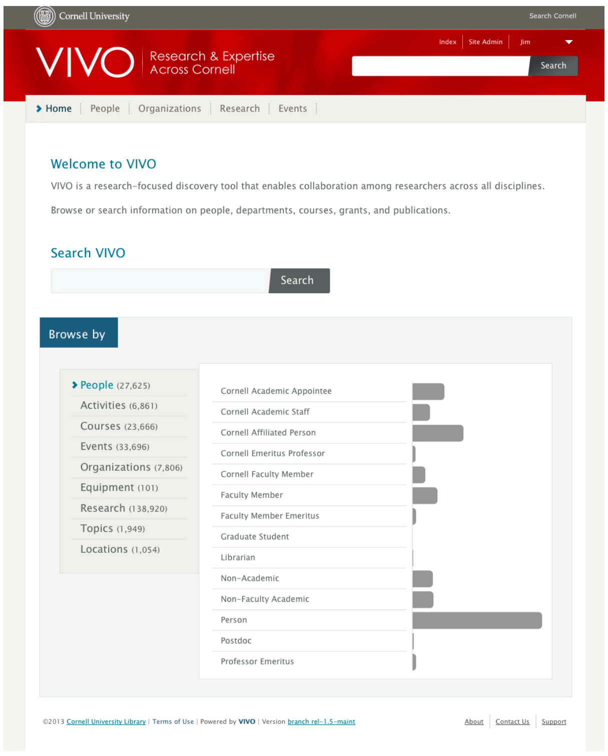
Melbourne Research

Level 5, 161 Barry Street
The University of Melbourne
Parkville 3010 VIC Australia

Email: Make an enquiry

Authoriser:
Deputy Vice-Chancellor (Research)
Maintainer:
Research Marketing Manager

Phone: 13 MELB (13 6352) | International: +(61 3) 9035 5511
The University of Melbourne ABN: 84 002 705 224
CRICOS Provider Code: 00116K (visa information)
Web brand guidelines created by Marketing


Cornell University
Search Cornell
Index | Site Admin | jim
VIVO Research & Expertise Across Cornell
Search
Home | People | Organizations | Research | Events
Welcome to VIVO
Search VIVO
Search
Browse by
About | Contact Us | Support

VIVO is a research-focused discovery tool that enables collaboration among researchers across all disciplines.

Browse or search information on people, departments, courses, grants, and publications.

People (27,625)	Cornell Academic Appointee
Activities (6,861)	Cornell Academic Staff
Courses (23,666)	Cornell Affiliated Person
Events (33,696)	Cornell Emeritus Professor
Organizations (7,806)	Cornell Faculty Member
Equipment (101)	Faculty Member
Research (138,920)	Faculty Member Emeritus
Topics (1,949)	Graduate Student
Locations (1,054)	Librarian
	Non-Academic
	Non-Faculty Academic
	Person
	Postdoc
	Professor Emeritus

©2013 Cornell University Library | Terms of Use | Powered by VIVO | Version branch rel-1.5-maint

What do you need to know?

- Standard web-site technologies: HTML, CSS and maybe JavaScript.
- Something about the Freemarker[133] template engine.
- Where the theme files are stored in VIVO, and how to reference them.

133 http://freemarker.org/

Getting started

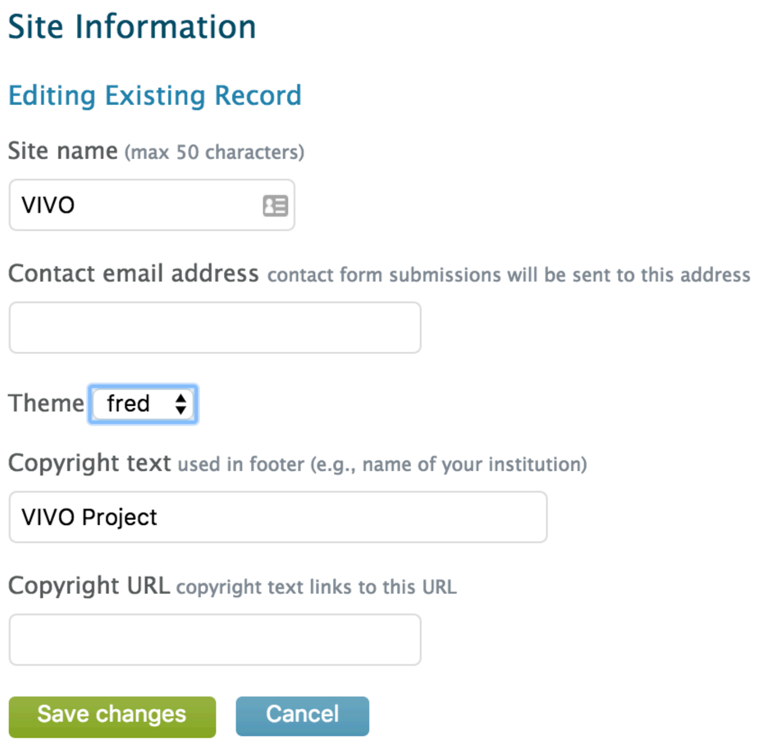
VIVO comes with a standard theme, called `wilma.` `wilma` is a folder in `vivo/installer/webapp/target/vivo/themes`.

To create a new theme, choose a name for your new theme. In these examples below we we will call the new theme `fred`.

Copy the wilma directory and its contents to a new directory called `fred`. `fred` must also be in `vivo/installer/webapp/target/vivo/themes.`

Your new theme will contain CSS files, image files, and Freemarker[134] templates.

Run the Maven install to deploy your new theme to the Tomcat container. Restart the VIVO Tomcat process. You can then go to the ***Site Admin*** page and choose ***Site Information***, to select your theme as the current one.


Site Information

Editing Existing Record

Site name (max 50 characters)

VIVO

Contact email address contact form submissions will be sent to this address

Theme fred

Copyright text used in footer (e.g., name of your institution)

VIVO Project

Copyright URL copyright text links to this URL

8.3.11.2 The structure of pages in VIVO

The pages in VIVO are built around three different frameworks. Each of these uses the same header and footer, to provide consistency. In addition to including the header and footer, the pages frequently include smaller templates to provide detail.

These are the basic frameworks:

The home page	As the point of entry for VIVO, the home page is special. It is based on the Freemarker template `page-home.ftl`

134 http://freemarker.org/

All other public pages	Based on the Freemarker template `page.ftl`
"back-end" pages	Pages used for editing the ontology, or manipulating the raw data of VIVO are based on a JSP named `basicPage.jsp`

8.3.11.3 Some significant templates

page.ftl

`page.ftl` is the default base template. The rest of the theme templates listed are components of `page.ftl` (included either directly or indirectly). Closer inspection of `page.ftl` reveals a stripped down file that declares minimal markup itself and instead reads as a list of includes for the component templates.

On the VIVO home page, page-home.ftl is used instead of page.ftl. It serves much the same purpose, but allows you to create a different layout for your home page than for the other pages in VIVO.

For consistency, It is critical that the following components be maintained:

head.ftl

This component template is responsible for everything within the `<head>` element. Note that the open and closing tags for the `<head>` element are defined in `page.ftl` and wrap the include for `head.ftl`. There are several includes within `head.ftl` that should be carried over to any new theme to maintain expected functionality:

- `<#include "stylesheets.ftl">` - ensures that the necessary stylesheets called by templates downstream will be added to the page via <link> elements
- `<#include "headscripts.ftl">` - ensures that the scripts called by templates which must be in the <head> will be added to the page via <script> elements

identity.ftl

This component template is responsible for rendering the VIVO logo, secondary navigation and search input field at the top of the page. There are no mandatory includes from `identity.ftl` that need to be carried over but there are 2 template variables that are of particular interest (`${user}` and `${urls}`).

menu.ftl

This component template is responsible for rendering the primary navigational menu for the site. In wilma, it also happens to declare the open tag for the main content container. There are no mandatory includes from `menu.ftl`. The `${menu}` template variable is crucial since it contains an array of menu items needed to build the primary navigational menu.

footer.ftl

This component template is responsible for rendering the copyright notice, revision information, secondary navigation, and link for the contact form. There is a single include that should be maintained:

- `<#include "scripts.ftl">` - ensures that the non head scripts (those that don't need to be placed in the `<head>`) called by the templates will be added to the page via `<script>` elements

Several template variables of interest include `${copyright}`, `${user}`, and `${version}`.

googleAnalytics.ftl

This component template is included by `footer.ftl`. Simply uncomment the `<script>` element and provide your Google Analytics Tracking Code.

Adjust the markup as necessary in `page.ftl`, and these component templates to achieve the desired content structure, and modify the stylesheets to meet layout needs and style your site. Remember that changes should be made in the source directory and that you will need to redeploy the project before the changes are reflected in the live website.

You can find more information about the structure of the VIVO theme in How VIVO creates a page (see page 215).

8.3.11.4 Making changes

Modify files in the theme

You can edit the Freemarker templates and the CSS files in the theme with any text editor. You can replace the image files with images that you choose.

Add files to the theme

Add CSS, JavaScript, or image files

As you modify the templates, you may want to use additional images, CSS files, or JavaScript files. When your templates refer to these files, they will use the Freemarker variable `urls.theme`, as shown in these examples:

```
<!-- an image file -->
<img src="${urls.theme}/images/arrow-green.gif"/>

<!-- a CSS file -->
<link rel="stylesheet" href="${urls.theme}/css/screen.css" />

<!-- a JavaScript file (create a js directory in your theme) -->
<script type="text/javascript" src="${urls.theme}/js/my.js"></script>
```

Add Freemarker templates

If your modifications use new Freemarker templates, you can refer to them more simply. Freemarker already knows where your theme directory is located.

```
<#include "my-new-template.ftl">
```

Override files that are not in the theme directory

In order to keep the theme directory uncluttered, VIVO keeps most of the front-end files in a separate location. Changes to the theme usually involve the files in the theme directory, but you can override other files as well.

Override CSS, JavaScript or image files that are not in the theme directory

You may notice that templates refer to files that are not in the theme directory. They use references based on the Freemarker variable `urls.base` instead of `urls.theme`, like this:

```
<!-- an image file -->
```

```
<img src="${urls.base}/images/arrowIcon.gif"/>

<!-- a CSS file -->
<link rel="stylesheet" href="${urls.base}/css/login.css" />

<!-- a JavaScript file -->
<script type="text/javascript" src="${urls.base}/js/browserUtils.js"></script>
```

These refer to files in the `vivo/installer/webapp/target/vivo` directory. If you look, you will see that this directory contains some files also used in the construction of thr VIVO interface.

Override Freemarker templates that are not in the theme directory

To override templates not in the theme directory, simply modify freemarker templates in `vivo/installer/webapp/target/vivo.` These changes will apply to all your themes.

VIVO treats all available Freemarker templates as belonging to the same flat namespace, whether they are in the theme directory or in the `templates/freemarker` directory, or one of its sub-directories. A file in `vivo/installer/webapp/target/vivo` can be overridden by a corresponding file in the theme directory.

Working on the theme

When you make changes to VIVO, you should make the changes in your VIVO distribution directory, run Maven install, restart Tomcat, and test the changes. If you are doing full customizing of VIVO, this cycle might be best.

If you are only working on the theme, you can speed things up.

- Tell the build script to skip the unit tests: they don't test the theme
  - `mvn install -Dskiptests=true`
- Don't restart Tomcat
  - VIVO always serves the most recent version of CSS files, image files, and JavaScript files. You don't need to restart Tomcat to make that happen.
  - However, your browser may cache these files so you won't see the most recent version. Here are some suggestions for bypassing your browser cache[135].
- Tell VIVO to reload Freemarker templates each time they are requested. See Tips for Interface Developers (see page 226).

Some developers prefer to make theme changes inside the `tomcat/webapp/vivo` directory. This eliminates the need to run the build script, but opens the threat of having the changes over-written the next time the build script runs.

When to restart Tomcat

If you make changes to any of the source files in the theme, including images, CSS, JavaScript or Freemarker templates, you must run the build script, but you do not need to restart Tomcat.

8.3.12 Creating custom entry forms

135 http://en.wikipedia.org/wiki/Wikipedia:Bypass_your_cache

8.3.12.1 Overview

Custom entry forms allow VIVO to transcend the general-purpose, utilitarian editing scheme of Vitro. Without custom entry forms, VIVO users must edit each RDF triple individually. With a custom entry form, users can edit a complex data structure on a single page.

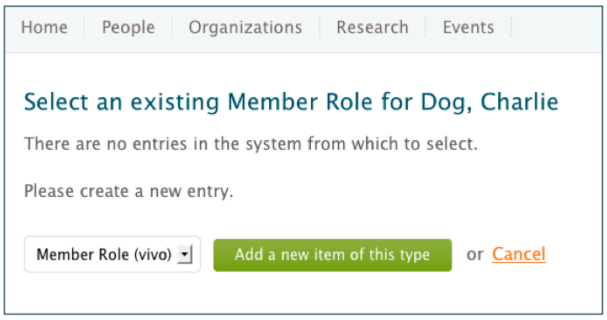
VIVO is distributed with a dozens of custom entry form generators. You may want to modify these form generators, or add more of your own.

8.3.12.2 An example

Say you wish to establish that a particular person is a member of a particular academic department. This relationship can be expressed as a member role. See Membership Model (see page 336)

But what if the academic department doesn't exist in VIVO yet? You will want to create that department, and assign a name to it. You may also want to record the member role in that department, when their membership began, and when it ended (if it is not ongoing).

Without a custom entry form, you would need to record each piece of data individually.

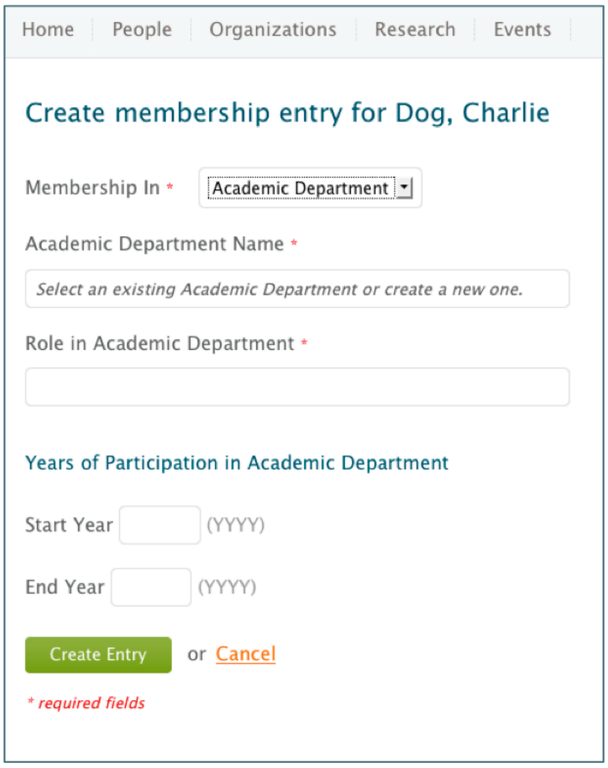


Select an existing Member Role for Dog, Charlie

There are no entries in the system from which to select.

Please create a new entry.

VIVO includes a custom form generator for this relationship. The custom entry form looks like this:



Create membership entry for Dog, Charlie

Membership In * Academic Department

Academic Department Name *

Select an existing Academic Department or create a new one.

Role in Academic Department *

Years of Participation in Academic Department

Start Year (YYYY)

End Year (YYYY)

* required fields

8.3.12.3 How is it created?

The creation of custom entry forms is an arcane and eldritch art, for which little documentation is available.

Each form requires a Java class known as a `EditConfigurationGenerator`. The generator describes the data structure being created, lists the SPARQL queries used, and includes a reference to the Freemarker template that will render the form.

You can start by examining the existing generators in this directory
`[VIVO]/src/edu/cornell/mannlib/vitro/webapp/edit/n3editing/configuration/generators`

and the Freemarker templates found here
`[VIVO]/productMods/templates/freemarker/edit/forms`

- Note: The directory structure has changed in version 1.9+.
  [VIVO]/src/... is now [VIVO]/api/src/main/java/...
  [VIVO]/productMods/... is now [VIVO]/webapp/src/main/webapp/...

There is also a short page of technical description called Implementing custom forms using N3 editing (see page 196).

—

8.3.12.4 Accessing VIVO Data Models

Accessing the models

There is an incredible variety of ways to access all of these models. Some of this variety is because the models are accessed in different ways for different purposes. Additional variety stems from the evolution of VIVO in which new mechanisms were introduced without taking the time and effort to phase out older mechanisms.

Here are some of the ways for accessing data models:

Attributes on Context, Session, or Request

Previously, it was common to assign a model to the ServletContext, to the HTTP Session, or to the HttpSessionRequest like this:

```
OntModel ontModel = (OntModel) getServletContext().getAttribute("jenaOntModel");

Object sessionOntModel = request.getSession().getAttribute("jenaOntModel");

ctx.setAttribute("jenaOntModel", masterUnion);
```

Occasionally, conditional code was inserted, to retrieve a model from the Request if available, and to fall back to the Session or the Context as necessary. Such code was sporadic, and inconsistent. This sort of model juggling also involved inversions of logic, with some code acting so a model in the Request would override one in the Session, while other code would prioritize the Session model over the one in the Request. For example:

```
public OntModel getDisplayModel(){
    if( _req.getAttribute("displayOntModel") != null ){
        return (OntModel) _req.getAttribute(DISPLAY_ONT_MODEL);
    } else {
        HttpSession session = _req.getSession(false);
        if( session != null ){
            if( session.getAttribute(DISPLAY_ONT_MODEL) != null ){
                return (OntModel) session.getAttribute(DISPLAY_ONT_MODEL);
            }else{
                if( session.getServletContext().getAttribute(DISPLAY_ONT_MODEL) != null){
                    return (OntModel)session.getServletContext().getAttribute(DISPLAY_ONT_MODEL);
                }
            }
        }
    }
    log.error("No display model could be found.");
    return null;
}
```

This mechanism has been removed in 1.6, being subsumed into the `ModelAccess` class (see below). Now, the `ModelAccess` attributes on Request, Session and Context are managed using code that is private to `ModelAccess` itself. Similarly, the code which gives priority to a Request model over a Session model is uniformly implemented across the models.

It remains to be seen whether this uniformity can satisfy the various needs of the application. If not, at least the changes can all be made within a single point of access.

The DAO layer

This mechanism is pervasive through the code, and remains quite useful. In it, a `WebappDaoFactory` is created, with access to particular data models. This factory then can be used to create DAO objects which satisfy interfaces like `IndividualDao`, `OntologyDAO`, or `UserAccountsDAO`. Each of these object implements a collection of convenience methods which are used to manipulate the backing data models.

Because the factory and each of the DAOs is an interface, alternative implementations can be written which provide

- Optimization for Jena RDB models
- Optimization for Jena SDB models
- Filtering of restricted data
- and more...

Initially, the `WebappDaoFactory` may have been used only with the full Union model. But what if you want to use these DAOs only against asserted triples? Or only against the ABox? This led to the `OntModelSelector`.

OntModelSelectors

An `OntModelSelector` provides a way to collect a group of Models and construct a `WebappDaoFactory`. With slots for ABox, TBox, and Full model, an `OntModelSelector` could provide a consistent view on assertions, or on inferences, or on the union. The `OntModelSelector` also holds references to a display model, an application metadata model, and a user accounts model, but these are more for convenience than flexibility.

Prior to release 1.6, `OntModelSelectors`, like `OntModels`, were stored in attributes of the Context, Session, and Request. They have been subsumed into the `ModelAccess` class.

Further, the semantics of the "standard" `OntModelSelectors` have changed, so they only act as facades before the Models store in `ModelAccess`. In this way, if we make this call:

```
ModelAccess.on(session).setOntModel(ModelID.BASE_ABOX, someWeirdModel)
```

Then both of the following calls would return the same model:

```
ModelAccess.on(session).getOntModel(ModelID.BASE_ABOX);
ModelAccess.on(session).getBaseOntModelSelector().getABoxModel();
```

Again, this is a change in the semantics of OntModelSelectors. It insures a consistent representation of `OntModels` across `OntModelSelectors`, but it is certainly possible that existing code relies on an inconsistent model instead.

The RDF Service

Model makers and Model sources

The ModelAccess class

⚠ TBD - Show how it represents all of these distinctions. Describe the scope searching and masking, wrt set and get. Include the OntModelSelectors and WADFs.

Initializing the Models

When VIVO starts up, `OntModel` objects are created to represent the various data models. The configuration models are created from the datasource connection, usually to a MySQL database. The content models are created using the new RDFService layer. By default this also uses the datasource connection, but it can be configured to use any SPARQL endpoint for its data.

Some of the smaller models are "memory-mapped" for faster access. This means that they are loaded entirely into memory at startup. Any changes made to the memory image will be replicated in the original model.

The data in each model persists in the application datasource (usually a MySQL database), or in the RDFService. Also, data from disk files may be loaded into the models. This may occur:

- the first time that VIVO starts up,
- if a model is found to be empty,
- every time that VIVO starts up.

depending on the particular model.

Where are the RDF files?

In the distribution, the RDF files appear in `[vivo]/rdf` and in `[vitro]/webapp/rdf`. These directories are merged during the build process in the usual way, with files in VIVO preferred over files in Vitro.

During the build process, the RDF files are copied to the VIVO home directory, and at runtime VIVO will read them from there.

The "first time"

For purposes of initialization, "first time" RDF files are loaded if the relevant data model contains no statements. Content models may also load "first time" files if the RDFService detects that its SDB-based datastore has not been initialized.

Initializing Configuration models

Application metadata
Function: Describes the configuration of VIVO at this site. Many of the configuration options are obsolete.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-applicationMetadata

Source: the application Datasource (MySQL database) (memory-mapped)

If this is the first startup, read the files in rdf/applicationMetadata/firsttime.

- In Vitro, there are none
- In VIVO, initialSiteConfig.rdf, classgroups.rdf and propertygroups.rdf

User Accounts
Contains login credentials and assigned roles for VIVO users.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-userAccounts

Source: the application Datasource (MySQL database) (memory-mapped)

If this model is empty, read the files in rdf/auth/firsttime.

- In Vitro, there are none (except during Selenium testing)
- In VIVO, there are none.

Every time, read the files in rdf/auth/everytime

- In Vitro, permissions_config.n3
- In VIVO, there are none.

The Display model
This is the ABox for the display model, and contains the RDF statements that define managed pages, custom short views, and other items.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadata

Source: the application Datasource (MySQL database) (memory-mapped)

If this model is empty, read the files in rdf/display/firsttime

- In Vitro, application.owl, menu.n3, profilePageType.n3
- VIVO contains its own copy of menu.n3, which overrides the one in Vitro

Every time, read the files in rdf/display/everytime

- in Vitro, displayModelListViews.rdf
- In VIVO, homePageDataGetters.n3, localeSelectionGUI.n3, vivoDepartmentQueries.n3, vivoListViewConfig.rdf, vivoSearchProhibited.n3

Display TBox
The TBox for the display model.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadataTBOX

Source: the application Datasource (MySQL database) (memory-mapped)

Every time, read the files in rdf/displayTbox/everytime.

- In Vitro, displayTBOX.n3
- In VIVO, there are none

DisplayDisplay
Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadata-displayModel

Source: the application Datasource (MySQL database) (memory-mapped)

Every time, read the files in rdf/displayDisplay/everytime

- In Vitro, displayDisplay.n3
- In VIVO, there are none.

Initializing Content models

base ABox
Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-2

Source: named graph from the RDFService

If first setup, read the files in rdf/abox/firsttime

- In Vitro, there are none
- In VIVO, geopolitical.ver1.1-11-18-11.individual-labels.rdf

Every restart, read the files in rdf/abox/filegraph, and create named models in the RDFService. Add them as sub-models to the base ABox. If these files are changed or deleted, update the RDFService accordingly.

- In Vitro, there are none
- In Vivo, geopolitical.abox.ver1.1-11-18-11.owl, academicDegree.rdf, continents.n3
  us-states.rdf, dateTimeValuePrecision.owl, validation.n3, documentStatus.owl, vocabularySource.n3

base TBox
Name: http://vitro.mannlib.cornell.edu/default/asserted-tbox

Source: named graph from the RDFService (memory-mapped)

If first setup, read the files in rdf/tbox/firsttime (without subdirectories)

- In Vitro, there are none
- In VIVO, additionalHiding.n3 initialTBoxAnnotations.n3

Every restart, read the files in rdf/tbox/filegraph, and create named models in the RDFService. Add them as sub-models to the base TBox. If these files are changed or deleted, update the RDFService accordingly.

- In Vitro, vitro-0.7.owl, vitroPublic.owl
- In VIVO, 44 files:

/usr/local/vivo/home/rdf/tbox/filegraph

```
README.md                                    education.owl                         personTypes.n3
agent.owl                                    event.owl                                 process.owl
appControls-temp.n3                        geo-political.owl                  publication.owl
bfo-bridge.owl                                  grant.owl                                 relationship.owl
bfo.owl                                              linkSuppression.n3                    relationshipAxioms.n3
classes-additional.owl           location.owl                          research-resource-iao.owl
clinical.owl                                   object-properties.owl        research-resource.owl
```

```
contact-vcard.owl                    object-properties2.owl       research.owl
contact.owl                                object-properties3.owl       role.owl
data-properties.owl                    objectDomains.rdf            sameAs.n3
dataDomains.rdf                            objectRanges.rdf             service.owl
dataset.owl                                ontologies.owl                     skos-vivo.owl
date-time.owl                            orcid-interface.n3               teaching.owl
dateTimeValuePrecision.owl       other.owl                             vitro-0.7.owl
documentStatus.owl                   outreach.owl                         vitroPublic.owl
```

base Full
Source: a combination of base ABox and base TBox

inference ABox
Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-inf

Source: named graph from the RDFService

inference TBox
Name: http://vitro.mannlib.cornell.edu/default/inferred-tbox

Source: named graph from the RDFService (memory-mapped)

inference Full
Source: a combination of inference ABox and inference TBox

union ABox
Source: a combination of base ABox and inference ABox

union TBox
Source: a combination of base TBox and inference TBox

union Full
Source: a combination of union ABox and union TBox

Transition from previous methods

⚠ TBD - What are we transitioning from? Check out VIVO-82.

- Semantics have changed: saves code, but may alter some uses.
  - Always searches the stack
  - OMS are facades with no internal state
    - There is no way to set an OMS - set the models instead
    - Keeps consistent

	prior to ModelAccess	**using ModelAccess**
User Accounts Model	ctx.getAttribute("userAccountsOntModel")	ModelAccess.on(ctx).getUserAccountsModel()
	ctx.setAttribute("userAccountsOntModel", model)	ModelAccess.on(ctx).setUserAccountsModel(model)

	prior to ModelAccess	**using ModelAccess**
DisplayModel	req.getAttribute("displayOntModel")	ModelAccess.on(req).getDisplayModel()
	session.getAttribute("displayOntModel")	ModelAccess.on(session).getDisplayModel()

	ctx.getAttribute("displayOntModel")
	ModelContext.getDisplayModel(ctx)
		ModelAccess.on(ctx).getDisplayModel()

	ctx.setAttribute("displayOntModel", model)
	ModelContext.setDisplayModel(model, ctx)
		ModelAccess.on(ctx).getDisplayModel()

	req.setAttribute("displayOntModel", model)	ModelAccess.on(req).setDisplayModel(model)
"jenaOntModel"	ctx.getAttribute("jenaOntModel")	ModelAccess.on(ctx).getJenaOntModel()
	session.getAttribute("jenaOntModel")	ModelAccess.on(session).getJenaOntModel()
	req.getAttribute("jenaOntModel")	ModelAccess.on(req).getJenaOntModel()
	ctx.setAttribute("jenaOntModel", model)	ModelAccess.on(ctx).setOntModel(ModelID.UNION_FULL, model)

	req.setAttribute("jenaOntModel", model)	ModelAccess.on(req).setOntModel(ModelID.UNION_FULL, model)
		ModelAccess.on(req).setJenaOntModel(model)

"baseOntModel"
"assertionsModel"
Base Full Model
	ModelContext.getBaseOntModel(ctx)
	ctx.getAttribute("baseOntModel")
	session.getAttribute("baseOntModel")
		ModelAccess.on(ctx).getOntModel(ModelID.BASE_FULL)
		ModelAccess.on(ctx).getBaseOntModel()

	ModelContext.setBaseOntModel(model, ctx)	

	prior to ModelAccess	using ModelAccess
"inferenceModel " Inference Full Model	ctx.getAttribute("inferenceOntModel")	ModelAccess.on(ctx).getInferenceOntModel()

Notes:

- "jenaOntModel" is a previous term for the Union Full model. The convenience methods `getJenaOntModel()` and `setJenaOntModel(m)`support this use.
- "baseOntModel" and "assertionsModel" are both previous terms for the Base Full model. The convenience methods `getBaseOntModel()` and `setBaseOntModel(m)`support this use.

	prior to ModelAccess	using ModelAccess

ontModelSelector
unionOntModelSelector
	ModelContext.setOntModelSelector(model, ctx)
	ModelContext.getUnionOntModelSelector(ctx)
	ctx.getAttribute("ontModelSelector")
	ctx.getAttribute("unionOntModelSelector")
		no mutator methods
		ModelAccess.on(ctx).getOntModelSelector()
		ModelAccess.on(ctx).getUnionOntModelSelector()

baseOntModelSelector	ctx.getAttribute("baseOntModelSelector")	ModelAccess.on(ctx).getBaseOntModelSelector()
inferenceOntModelSelector	ctx.getAttribute("inferenceOntModelSelector")	ModelAccess.on(ctx).getInferenceOntModelSelector()

- The default WebappDaoFactory is the one backed by the unionOntModelSelector. On the request level, this is also known as the "fullWebappDaoFactory". The convenience methods`getWebappDaoFactory()` and `setWebappDaoFactory(wdf)` support this use.
- "baseWebappDaoFactory" and "assertionsWebappDaoFactory" are both previous terms for the WebappDaoFactory backed by the baseOntModelSelector. The convenience methods `getBaseWebappDaoFactory()` and `setBaseWebappDaoFactory(wdf)` support this use.
- Nobody was using the "deductionsWebappDaoFactory", so we got rid of it.

8.3.12.5 Implementing custom forms using N3 editing

Overview

The Vitro/VIVO system comes with basic RDF editing capabilities to add object and datatype statements to individuals. Frequently, people deploying Vitro/VIVO desire a web form which allows editing of multiple properties and individuals on the same form. A contact information form would be an example of a feature that would be implemented with a custom form in Vitro/VIVO.

The creation of a custom forms in Vitro/VIVO is done in two parts. The first is an implementation of the java interface EditConfigurationGenerator and the second is a FreeMarker template for the presentation. The EditConfigurationGenerator creates a EditConfiguration that controls how the values from the form will be used in the editing of the RDF, server side validation, which template to use, and other aspects of the edit. The FreeMarker template controls the HTML and Javascript for the form.

The EditConfigurationGenerator classes can be associated with an RDF property so that they are used from an individual's profile page or by a direct URL.

The main concept of custom forms is that the values submitted by the HTTP request will be substituted into placeholders in RDF N3 strings. These strings are then parsed to Jena RDF Model objects and that RDF is added to the system. For the modification of an existing value, a second set of strings is created and parsed which become the RDF statements to remove for the edit. This substitution is why the editing system is frequently called "N3 Editing". In practice, the N3 strings use only the turtle subset of the N3 syntax.

Steps of an Edit

Step 1. Getting the link to the edit

When a user is logged in, individual profile pages have edit links next to the listed properties. These links will take the user to a page with an edit form. The links on the individual profile page are routed to the EditRequestDispatchController which will determine which EditConfigurationGenerator to use based on which property is being edited. The VIVO/Vitro system can be configured to associate a EditConfigurationGenerator with a property so that the edit links will use a custom EditConfigurationGenerator. If no custom form is specified then the default object or data property EditConfigurationGenerator will be used.

A property can be associated with a custom form in one of two ways:
A) if you go to the site admin -object property hierarchy - the property you want associated with the form, click on the property then edit property record, you can put in the Java class name of the generator in the custom entry form field.
E.g.edu.cornell.mannlib.vitro.webapp.edit.n3editing.configuration.generators.AddDistributionGenerator. This will allow you to associate the custom form while the system is running.

B) if you will be deploying the system for the first time and starting with an empty database, you would update vivo-core-1.5-annotations.rdf to specify that the property has a custom form using the vitro:customEntryFormAnnot[136] property.

Step 2. Generating the EditConfiguration

When the user clicks the link, the client browser requests the URL of the edit link which will be to the EditRequestDispatchControl. That servlet will set up all that is needed in the session for the edit and respond with the HTML form. A custom form is setup by the EditConfigurationGenerator which has the sole purpose of making

136 http://vitrocustomEntryFormAnnot

an EditConfiguration object. The EditRequestDispatchController will run getEditConfiguration() on the EditConfigurationGenerator to create the EditConfiguration. The EditConfiguration object has properties to define the characteristics of the edit. The EditConfiguration will specify the FreeMarker template for the form, and the server side instructions for validating the submitted result and instructions for processing the edit. When authoring a custom form, a central task is the coding of the EditConfigurationGenerator to produce an EditConfiguration that encodes logic of how you desire the edit to happen. The EditConfigurationGenerator is just a java class that creates an EditConfiguration.

When generating the EditConfiguration at runtime, an edit key will be created and the completed EditConfiguration will be associated with that key in the server side user session. This edit key is used to handle parallel editing and back button complexity. The EditConfiguration object for an edit is in a one to one relation with the HTML form for an edit. If the user goes to edit a street address and then goes to edit that street address a second time, the first edit will have an EditConfiguration object in the session and an HTML form with one edit key, and the second will have a different EditConfiguration in the session and an HTML form with a different edit key. An HTML form created for a edit will have an “edit key“ to associate that specific instance of the HTML form with an object stored in the user’s session.

The EditConfiguration can specify SPARQL queries for existing values for fields of the form. These are executed as part of the generation of the EditConfiguration.

Step 3. HTML creation by FreeMarker

Once the EditRequestDispatchController has the EditConfiguration and put it in the session, it will set up some standard values for the template and pass them and the EditConfiguration to the FreeMarker template specified in EditConfiguration.getTemplate(). The HTML form is then generated using the normal FreeMarker process. The HTML form must contain a field with the EditKey so associate the edit with an EditConfiguration in the session.

Step 4. Response From Client

The form will be submitted by the client’s browser to ProcessRdfFormController. This will get the EditConfiguration based on the edit key from the submitted values. It will run validation and then substitute the values from the form into the N3 templates and parse the N3 to RDF. The N3 that gets created will be then added to the VIVO/Vitro models. If the edit is a change of an existing value, then the RDF for the statements to remove will be created and removed form the VIVO/Vitro models.

8.3.12.6 Servlet Lifecycle Management

Description

Like most Java Enterprise applications, Vitro servlets rely on the ServletContext to hold object that they will need to use when servicing requests. These objects are created by ServletContextListeners, which are run by the StartupManager.

The StartupManager creates instances of the listeners and runs them, accumulating information about their running in the StartupStatus.

As each listener runs, it may add messages to the StartupStatus. Each message will have a severity level associated with it:

- FATAL – The listener encountered a problem. Perhaps the application was configured incorrectly, or perhaps the system utilities are not performing as intended. The problem is severe enough that the application will not run. The message describes the problem, with suggestions on how to fix it.
- WARNING – The listener encountered a problem, but the problem will not prevent the application from running. The message describes the problem, and tells what parts of the application will be affected, with suggestions on how to fix the problem.
- INFO – No problem is indicated. The message contains information that may be helpful in monitoring the application.

If a FATAL status is recorded, the StartupManager will not execute any additional listeners. Access to the application will be blocked, and any attempt to access the application will display the StartupStatus in an error page.

If a WARNING status is recorded, the StartupManager continues as normal. Access to the application will be blocked one time, to display the StartupStatus. In subsequent requests, the application will respond normally.

When logged in, an administrator may view the StartupStatus from a link on the Site Admin page.

Specifying context listeners

In any Java Enterprise application, developers can specify context listeners in the deployment descriptor (web.xml). These listeners that will be activated when the application starts and when it shuts down.

In Vitro, the only listener in web.xml is the StartupManager. Here is the relevant section of Vitro's web.xml:

```
<!--
   StartupManager instantiates and runs the listeners from startup_listeners.txt
   All ServletContextListeners should be listed there, not here.
-->
<listener>
  <listener-class>edu.cornell.mannlib.vitro.webapp.startup.StartupManager</listener-class>
</listener>
```

Vitro contains a list of startup listeners in a file at Vitro (see page 363)/webapp/config/startup_listeners.txt. This file is simple text with each line containing the fully-qualified class name of a startup listener. Blank lines are ignored, as are comment lines – lines that begin with a "hash" character. Here is a portion of that file:

```
#
# ServletContextListeners for Vitro,
# to be instantiated and run by the StartupManager.
#

edu.cornell.mannlib.vitro.webapp.config.ConfigurationPropertiesSetup

edu.cornell.mannlib.vitro.webapp.config.RevisionInfoSetup

edu.cornell.mannlib.vitro.webapp.email.FreemarkerEmailFactory$Setup

# DefaultThemeSetup needs to run before the JenaDataSourceSetup to allow creation
# of default portal and tab
edu.cornell.mannlib.vitro.webapp.servlet.setup.DefaultThemeSetup
```

Writing context listeners

Each listener must implement the ServletContextListener interface, and must have a zero-argument constructor.

When Vitro starts, the StartupManager will call contextInitialized() in each listener, in the order that they appear in the file. The listener can call methods on StartupStatus to record messages. If the listener is successful, it should record one or more INFO messages that provide a brief description of what it has done. If a problem is detected, the listener may record WARNING messages or ERROR messages, depending on the severity of the problem. The listener may also throw a RuntimeException from contextInitialized(), which the StartupManager will treat like an ERROR.

Here is an example of a basic listener. When contextInitialized() is called, the listener will perform some setup. If there is no problem, a call to StartupStatus.info() reports some basic information about the listener's actions. If a problem is found, a call to StartupStatus.warning() describes the nature of the problem (by reporting the exception) and how this problem will affect the application.

```java
public static class Setup implements ServletContextListener {
   @Override
   public void contextInitialized(ServletContextEvent sce) {
       ServletContext ctx = sce.getServletContext();
       StartupStatus ss = StartupStatus.getBean(ctx);

       try {
           FreemarkerEmailFactory factory = new FreemarkerEmailFactory(ctx);
           ctx.setAttribute(ATTRIBUTE_NAME, factory);

           if (factory.isConfigured()) {
               ss.info(this, "The system is configured to "
                       + "send mail to users.");
           } else {
               ss.info(this, "Configuration parameters are missing: "
                       + "the system will not send mail to users.");
           }
       } catch (Exception e) {
           ss.warning(this,
                   "Failed to initialize FreemarkerEmailFactory. "
                           + "The system will not be able to send email "
                           + "to users.", e);
       }
   }

   @Override
   public void contextDestroyed(ServletContextEvent sce) {
       sce.getServletContext().removeAttribute(ATTRIBUTE_NAME);
   }
}
```

Note that the StartupManager treats ServletContextListeners just like you would expect from reading the Servlet 2.4 specification:

- Only one instance of the listener is created per JVM.
- The contextInitialized() method is called once when the system is starting.
- The contextDestroyed() method is called on that same instance when the system shuts down.

8.3.13 Enhancing Freemarker templates with DataGetters

8.3.13.1 Overview

It is possible for a Freemarker template to display data that is not normally provided to it.

You can create an RDF file that describes a custom `DataGetter` object, and associates it with the desired template. Each time that template is used, the `DataGetter` will be executed, and the data will be stored in a variable, so the template can display it.

This does not require changes to the Java code. You create the RDF file in your VIVO distribution directory and modify the template in your theme.

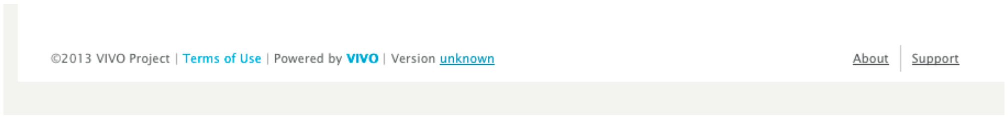
8.3.13.2 An example

Let's assume that we need to display information about the most recent data ingest operation. We want to display the name of the Person who supervised the ingest. We would like to display this on every page.

As part of the ingest process, we can load statements like this into the data model:

```
<http://vivo.mydomain.edu/individual/n5242>
    <http://vivo.mydomain.edu/individual/isMostRecentUpdater>
    "true" .
<http://vivo.mydomain.edu/individual/n5242>
    <http://www.w3.org/2000/01/rdf-schema#label>
    "Baker, Able" .
```

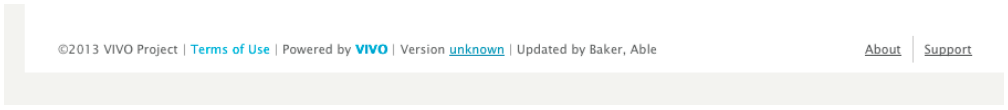
We would like for VIVO to display the name of this individual on every page, so the footer will change from this:



©2013 VIVO Project | Terms of Use | Powered by VIVO | Version unknown

About | Support

to this:



©2013 VIVO Project | Terms of Use | Powered by VIVO | Version unknown | Updated by Baker, Able

About | Support

8.3.13.3 Creating the DataGetter

VIVO allows you to define and use `DataGetter` objects in several contexts. `DataGetters` come in many flavors, but the most commonly used is the `SparqlQueryDataGetter`, which lets you define a SPARQL query, and store the results of that query for your Freemarker template to display.

By adding statements to your data model, you can define a `SparqlQueryDataGetter` object, and associate it with a Freemarker template. Here is the definition that is used in this example:

```
@prefix display: <http://vitro.mannlib.cornell.edu/ontologies/display/1.1#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<freemarker:footer.ftl> display:hasDataGetter display:updatedInfoDataGetter .

display:updatedInfoDataGetter
    a <java:edu.cornell.mannlib.vitro.webapp.utils.dataGetter.SparqlQueryDataGetter> ;
    display:saveToVar "updatedInfo" ;
    display:query """
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX local: <http://vivo.mydomain.edu/individual/>

    SELECT (str(?rawLabel) AS ?updater)
    WHERE {
        ?uri local:isMostRecentUpdater ?o ;
             rdfs:label ?rawLabel .
    }
    LIMIT 1
     """ .
```

These statements can be added to your data model in any of several ways. For this example, I stored these lines in a file called `data_getter_for_example.n3` and placed it in the VIVO distribution directory under `rdf/display/everytime`. Files placed in this directory are loaded when VIVO starts, but are not persisted when VIVO stops. This allows you to edit or remove the file without leaving residual statements in your data model.

Notice that:

- The first statement says that the Freemarker template `footer.ftl` has a `DataGetter`, defined in subsequent lines.
- The definition of the `DataGetter` states:
    - the type of the data getter,
    - the SPARQL query that will be executed
    - the Freemarker variable that will hold the results.

The results of the SPARQL query are stored in a Freemarker variable, in this case `updatedInfo`. The variable will contain a Sequence of Hashes, where each Hash represents one line of the SPARQL result. Within each Hash, result values are specified as key/value pairs.

For more information on Sequences and Hashes, consult the Freemarker manual:

- Retrieving data from a Sequence[137]
- Retrieving data from a Hash[138]

8.3.13.4 Modifying the template

Here is the standard `footer.ftl` template:

137 http://freemarker.sourceforge.net/docs/dgui_template_exp.html#dgui_template_exp_var_sequence
138 http://freemarker.sourceforge.net/docs/dgui_template_exp.html#dgui_template_exp_var_hash

footer.ftl

```
<#-- $This file is distributed under the terms of the license in /doc/license.txt$ -->
</div> <!-- #wrapper-content -->
<footer role="contentinfo">
    <p class="copyright">
        <#if copyright??>
            <small>&copy;${copyright.year?c}
            <#if copyright.url??>
                <a href="${copyright.url}" title="${i18n().menu_copyright}">${copyright.text}</a>
            <#else>
                ${copyright.text}
            </#if>
             | <a class="terms" href="${urls.termsOfUse}" title="${i18n().menu_termuse}">$
{i18n().menu_termuse}</a></small>
        </#if>
        ${i18n().menu_powered} <a class="powered-by-vivo" href="http://vivoweb.org" target="_blank"
 title="${i18n().menu_powered} VIVO"><strong>VIVO</strong></a>
        <#if user.hasRevisionInfoAccess>
             | ${i18n().menu_version} <a href="${version.moreInfoUrl}" title="${i18n().menu_version}">$
{version.label}</a>
        </#if>
    </p>
    <nav role="navigation">
        <ul id="footer-nav" role="list">
            <li role="listitem"><a href="${urls.about}" title="${i18n().menu_about}">${i18n().menu_about}</
a></li>
            <#if urls.contact??>
                <li role="listitem"><a href="${urls.contact}" title="${i18n().menu_contactus}">$
{i18n().menu_contactus}</a></li>
            </#if>
            <li role="listitem"><a href="http://www.vivoweb.org/support" target="blank" title="$
{i18n().menu_support}">${i18n().menu_support}</a></li>
        </ul>
    </nav>
</footer>
<#include "scripts.ftl">
```

Insert these lines between lines 17 and 18:

```
        <#if (updatedInfo?first.updater)??>
            | Updated by ${updatedInfo?first.updater}
        </#if>
```

The SPARQL result is obtained and stored into the Freemarker variable `updatedInfo` each time the `footer.ftl` template is loaded for display. The name we want is in the first row of the SPARQL result, keyed to the name `updater`.

8.3.13.5 Summary

Enhancing Freemarker templates is one more way to use the VIVO `DataGetter` mechanism. When you associate a `DataGetter` with a Freemarker template, that `DataGetter` will be run each time the template is invoked. This is true whether the template is specified by the controller, or included in another template. You can modify the template to display the data from the `DataGetter`, but it is prudent to include an `<#if>` tag, so your template won't fail if the data is not found.

8.3.14 Enriching profile pages using SPARQL query DataGetters

8.3.14.1 Introduction

VIVO 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 possible to extend the faculty information to display all of the faculty members' research areas.

8.3.14.2 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[139] functionality.

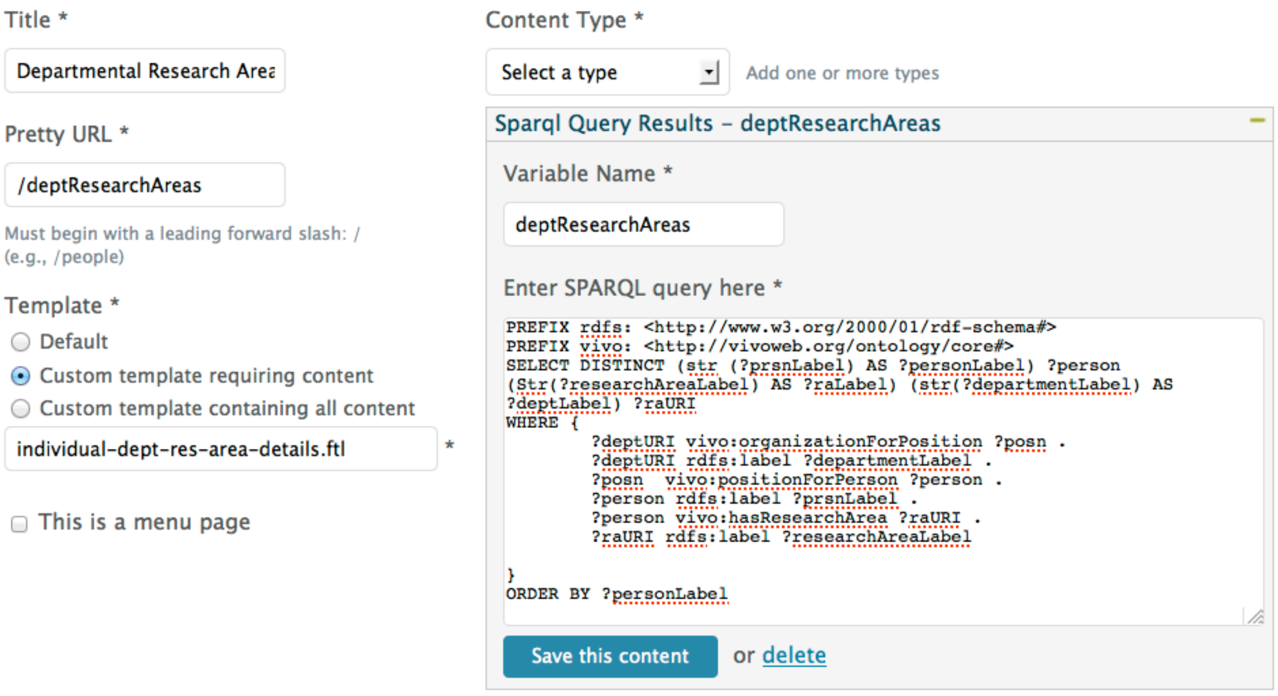
As noted in Step 4, the anchor tags in the list of research areas include an `href` attribute that takes this format:

```
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.

139 https://wiki.duraspace.org/display/VIVO/Customize%3A+Page+management


Title *
Departmental Research Area
Pretty URL *
/deptResearchAreas
Must begin with a leading forward slash: / (e.g., /people)
Template *
Default
[✓ Custom template requiring content]
Custom template containing all content
individual-dept-res-area-details.ftl
This is a menu page
Content Type *
Select a type
Add one or more types
Sparql Query Results – deptResearchAreas
Variable Name *
deptResearchAreas
Enter SPARQL query here *
Save this content
or delete

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX vivo: <http://vivoweb.org/ontology/core#>
SELECT DISTINCT (str (?prsnLabel) AS ?personLabel) ?person
(Str(?researchAreaLabel) AS ?raLabel) (str(?departmentLabel) AS
?deptLabel) ?raURI
WHERE {
        ?deptURI vivo:organizationForPosition ?posn .
        ?deptURI rdfs:label ?departmentLabel .
        ?posn  vivo:positionForPerson ?person .
        ?person rdfs:label ?prsnLabel .
        ?person vivo:hasResearchArea ?raURI .
        ?raURI rdfs:label ?researchAreaLabel

}
ORDER BY ?personLabel
```

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.

8.3.15 Multiple profile types for foaf:Person

8.3.15.1 Introduction

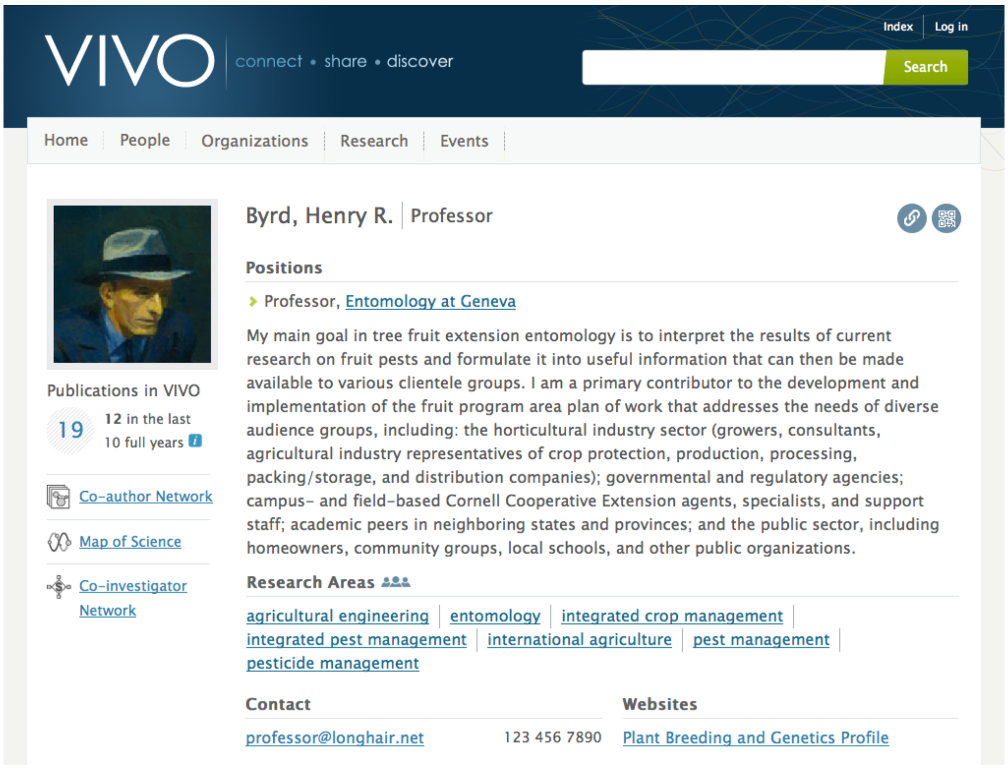
VIVO now supports multiple profile pages for foaf:Persons. This feature, which is optional so installations can continue to use just the individual--foaf-person.ftl template, currently consists of two profile page types: a standard view, which is a redesigned version of the foaf:Person template in previous releases; and a quick view, which emphasizes the individual's own web page presence while providing summary VIVO information, such as current positions and research areas. The profile quick view requires the use of a web service that captures images of web pages. This web service is not included with the VIVO software, so an installation will either have to develop their own service or use a third-party service, usually for a small fee depending on the number of images served. Examples of these services include WebShotsPro, Thumbalizr and Websnapr.

8.3.15.2 The Profile Page Types

As noted above, there are currently two supported profile page types. Here are examples of those two views

The Standard View

The standard view is similar to the default foaf:Person template except that the information displayed at the top of the page is divided into only two primary columns instead of three. The actual template name for this page type is individual--foaf-person-2column.ftl.



Byrd, Henry R. | Professor

Positions

› Professor, Entomology at Geneva

My main goal in tree fruit extension entomology is to interpret the results of current research on fruit pests and formulate it into useful information that can then be made available to various clientele groups. I am a primary contributor to the development and implementation of the fruit program area plan of work that addresses the needs of diverse audience groups, including: the horticultural industry sector (growers, consultants, agricultural industry representatives of crop protection, production, processing, packing/storage, and distribution companies); governmental and regulatory agencies; campus- and field-based Cornell Cooperative Extension agents, specialists, and support staff; academic peers in neighboring states and provinces; and the public sector, including homeowners, community groups, local schools, and other public organizations.

Research Areas

agricultural engineering | entomology | integrated crop management | integrated pest management | international agriculture | pest management | pesticide management

Contact

professor@longhair.net 123 456 7890

Websites

Plant Breeding and Genetics Profile

Publications in VIVO
19 — 12 in the last 10 full years

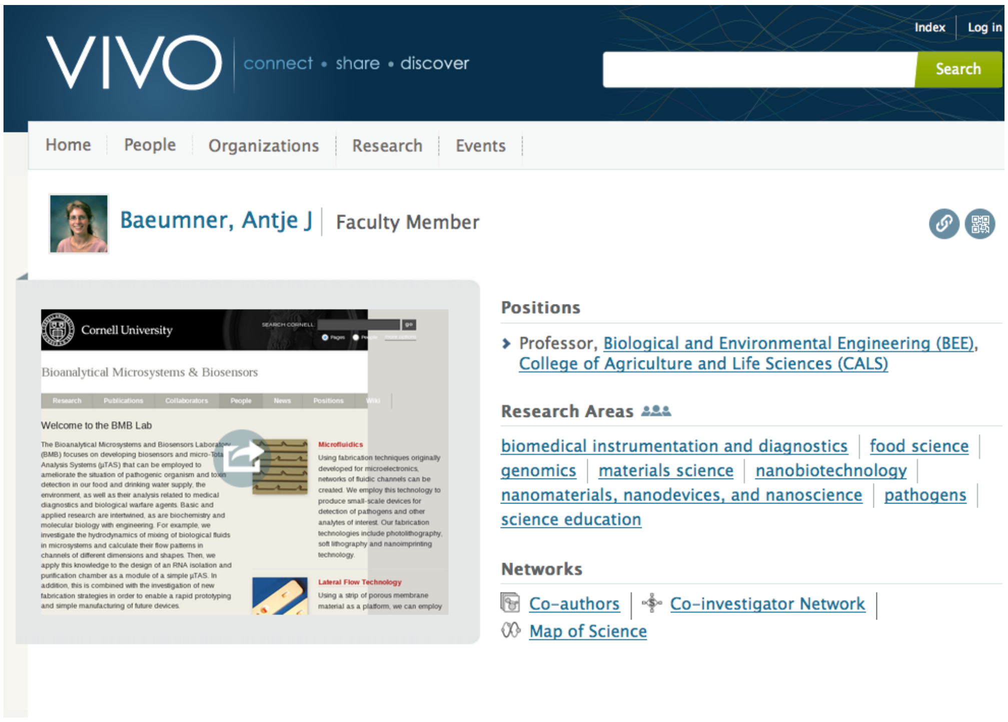
Co-author Network

Map of Science

Co-investigator Network

The Quick View

As illustrated below, the quick view puts a visual emphasis on the individual's own web presence. In this case, the person only has one web page displayed. When there is more than one, the primary web page is displayed as shown and any additional web pages are displayed as thumbnails beneath the primary one.



Baeumner, Antje J | Faculty Member

Positions

› Professor, Biological and Environmental Engineering (BEE), College of Agriculture and Life Sciences (CALS)

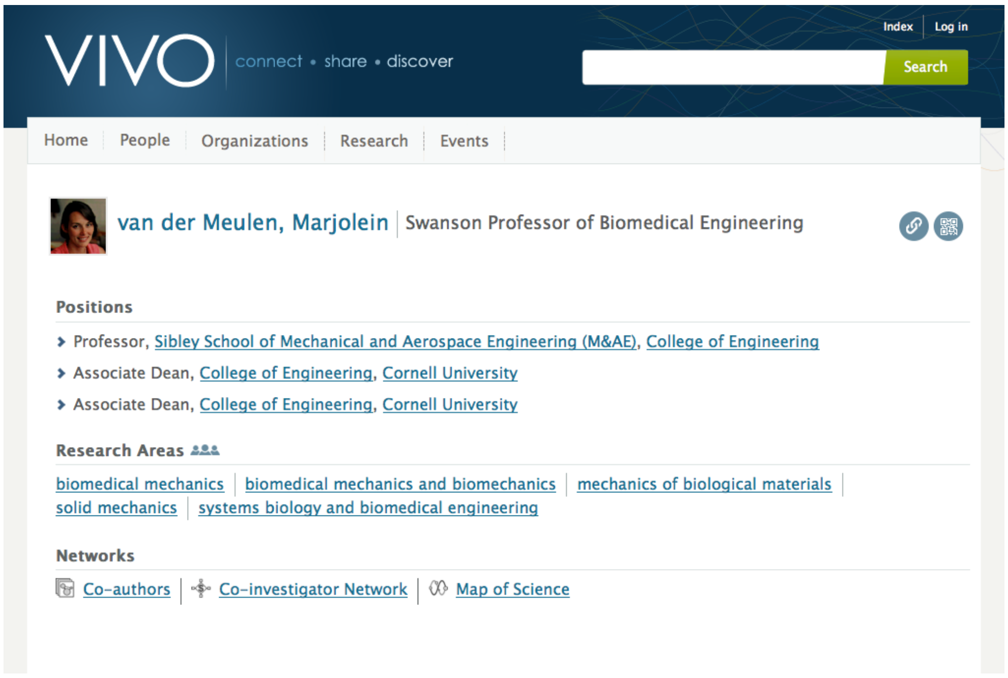
Research Areas

biomedical instrumentation and diagnostics | food science | genomics | materials science | nanobiotechnology | nanomaterials, nanodevices, and nanoscience | pathogens | science education

Networks

Co-authors | Co-investigator Network | Map of Science

It's possible that there will be some individuals who do not have a web page to display. In that situation the quick view will display as follows.



van der Meulen, Marjolein | Swanson Professor of Biomedical Engineering

Positions

- Professor, Sibley School of Mechanical and Aerospace Engineering (M&AE), College of Engineering
- Associate Dean, College of Engineering, Cornell University
- Associate Dean, College of Engineering, Cornell University

Research Areas

biomedical mechanics | biomedical mechanics and biomechanics | mechanics of biological materials | solid mechanics | systems biology and biomedical engineering

Networks

Co-authors | Co-investigator Network | Map of Science

8.3.15.3 Implementing Multiple Profile Pages

Here are the steps required to implement the multiple profile pages feature.

1. Develop or a website image capture service
2. Update the runtime.properties file
3. Override the default foaf:Person template
4. Update the webpage quick view template
5. Set the Profile Page Type for your foaf:Persons

Step 1. Develop or a Website Image Capture Service

Since there are currently only two page views, and one of those emphasizes the individual's own web site, to implement the multiple profile pages feature requires that an installation either develop its own web service for capturing images of web sites or select a third-party service for this purpose. As noted in the introduction, these services include WebShotsPro, Thumbalizr and Websnapr.

A third option, however, would be to modify the quick view template (individual--foaf-person-quickview.ftl) so that it does not display a web page image (as in the third screen shot above). This template file is located in the `productMods/templates/freemarker/body/individual` directory.

Step 2. Update the runtime.properties File

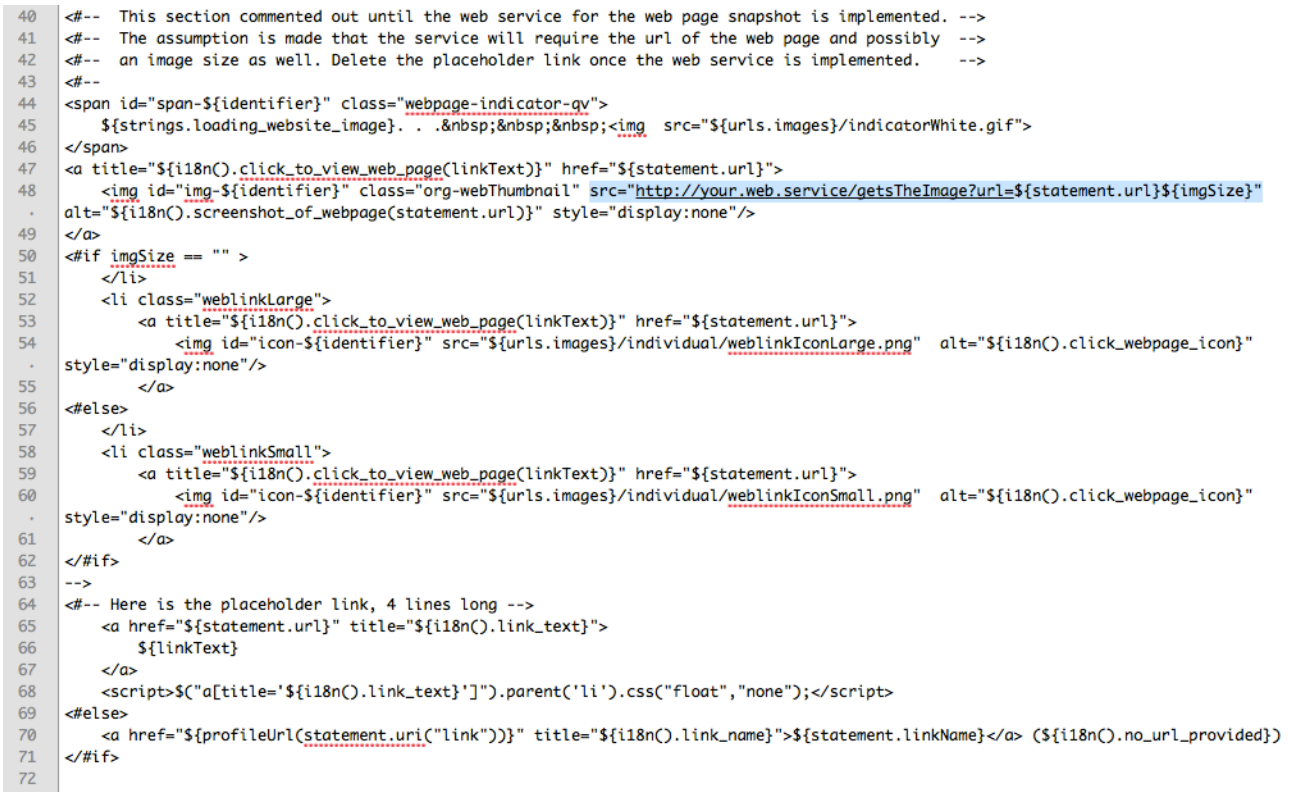
Set the multiViews.profilePageTypes to "enabled" and ensure that it is not commented out.

Step 3. Override the Default foaf:Person Template

There are two ways to override the default individual--foaf-person.ftl template, which is located in the `themes/wilma/templates` directory: (1) rename the file, or (2) remove it from that directory.

Step 4. Update the Webpage Quick View Template

The template that displays the web page image in the quick view is named propStatement-webpage-quickview.ftl. As delivered, this template uses a placeholder link (or links) to display the individual's web page (or pages), while the code that calls the web service is currently commented out. Here is that section of the template:

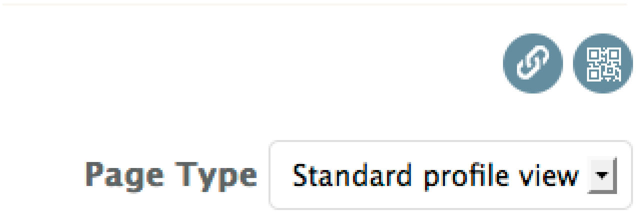


```
40  <#--  This section commented out until the web service for the web page snapshot is implemented. -->
41  <#--  The assumption is made that the service will require the url of the web page and possibly  -->
42  <#--  an image size as well. Delete the placeholder link once the web service is implemented.    -->
43  <#--
44  <span id="span-${identifier}" class="webpage-indicator-qv">
45      ${strings.loading_website_image}. . .&nbsp;&nbsp;&nbsp;<img  src="${urls.images}/indicatorWhite.gif">
46  </span>
47  <a title="${i18n().click_to_view_web_page(linkText)}" href="${statement.url}">
48      <img id="img-${identifier}" class="org-webThumbnail" src="http://your.web.service/getsTheImage?url=${statement.url}${imgSize}"
 ·  alt="${i18n().screenshot_of_webpage(statement.url)}" style="display:none"/>
49  </a>
50  <#if imgSize == "" >
51      </li>
52      <li class="weblinkLarge">
53          <a title="${i18n().click_to_view_web_page(linkText)}" href="${statement.url}">
54              <img id="icon-${identifier}" src="${urls.images}/individual/weblinkIconLarge.png"  alt="${i18n().click_webpage_icon}"
 ·  style="display:none"/>
55          </a>
56  <#else>
57      </li>
58      <li class="weblinkSmall">
59          <a title="${i18n().click_to_view_web_page(linkText)}" href="${statement.url}">
60              <img id="icon-${identifier}" src="${urls.images}/individual/weblinkIconSmall.png"  alt="${i18n().click_webpage_icon}"
 ·  style="display:none"/>
61          </a>
62  </#if>
63  -->
64  <#-- Here is the placeholder link, 4 lines long -->
65      <a href="${statement.url}" title="${i18n().link_text}">
66          ${linkText}
67      </a>
68      <script>$("a[title='${i18n().link_text}']").parent('li').css("float","none");</script>
69  <#else>
70      <a href="${profileUrl(statement.uri("link"))}" title="${i18n().link_name}">${statement.linkName}</a> (${i18n().no_url_provided})
71  </#if>
72
```

Note the highlighted text on line 48. The URL in the src attribute is where you call either the web service you developed or the third-party service. The APIs for these services are fairly standard. Besides the URL of the web site that will be the source of the screen shot, the code in this template assumes that the API also takes an image size. For example, some services can provide small, medium and large images; others may only provide a large image and a thumbnail image. Once you've updated line 48 to call your web service, remember to comment out or remove the placeholder link, lines 65-68.

Step 5. Set the Profile Page Type for your foaf:Persons

When multiple profile pages are implemented, the default view is the standard profile view. You can change an individual's profile page type through the GUI by accessing the Page Type drop down:



Page Type Standard profile view

You can also set the profile page type by ingesting RDF. An N3 triple, for example would consist of the following parts:

- the subject would be the URI of the individual, such as

  `<http://localhost:8080/individual/n7829>;`

- the predicate would be the hasDefaultProfilePageType object property,

  `<http://vitro.mannlib.cornell.edu/ontologies/display/1.1#hasDefaultProfilePageType>;`

- and the object would be the type of profile,

  `<http://vitro.mannlib.cornell.edu/ontologies/display/1.1#quickView>` (or `#standard`).

The ProfilePageType class is defined in the display model. Refer to the profilePageType.n3 file for details.

8.3.15.4 Using the Standard View Without Implementing Multiple Profile Pages

It's possible that an installation may want to use the standard view instead of the default foaf:Person template, but does not want to implement multiple profile pages. This can be done by simply (1) overriding the default foaf:Person template (just as in Step 3 above) and (2) ensuring that the multiViews.profilePageTypes properties in the runtime.properties file is either commented out or set to "disabled."

8.3.16 Using OpenSocial Gadgets

8.3.16.1 Overview

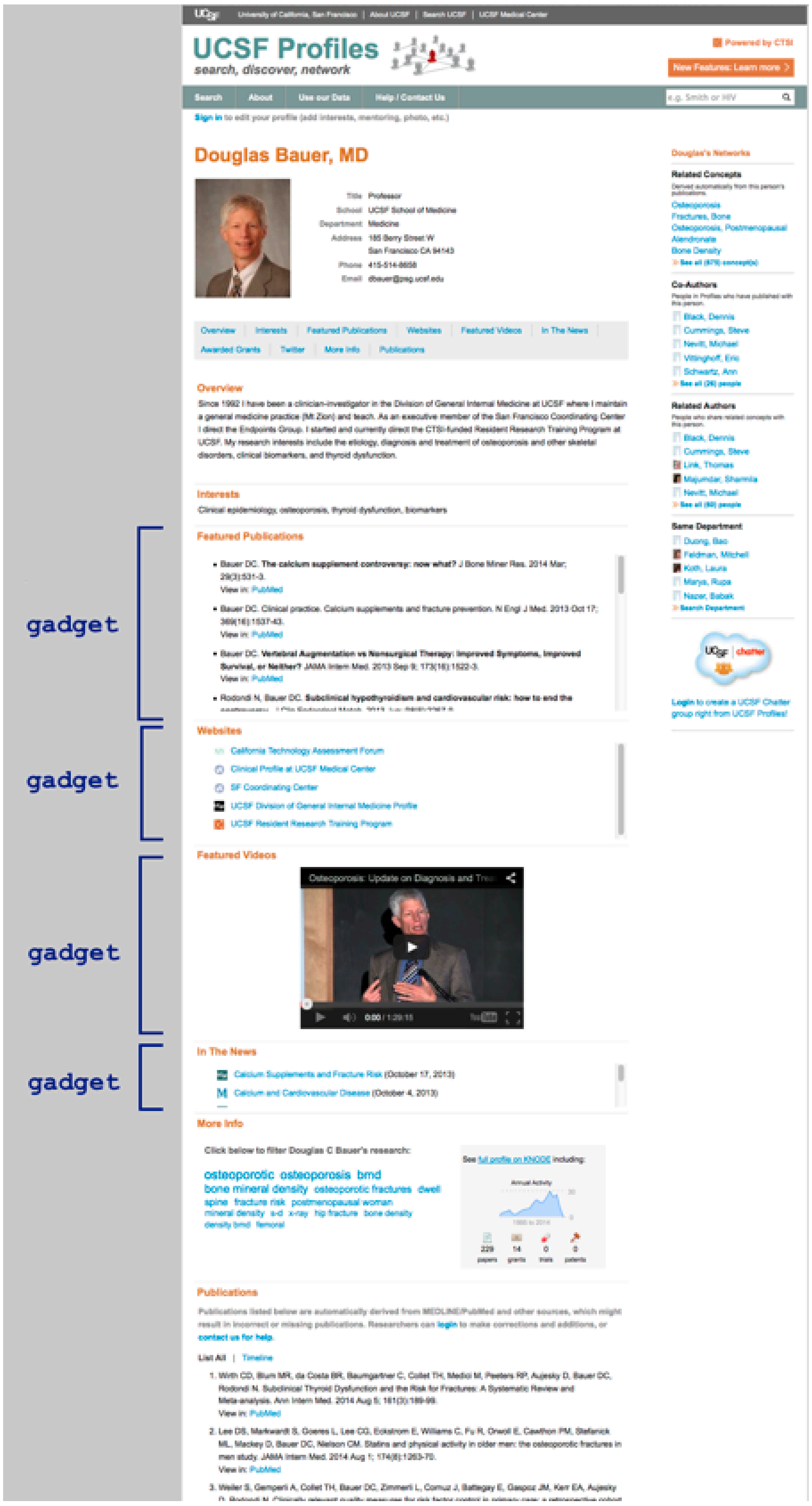
What can you do?

Your site administrators can configure a collection of "gadgets" for your VIVO installation. From that collection, each faculty member can decide which gadgets he will show on his profile page, and how they should be configured.

Perhaps it would be better to describe the gadgets as "page sections", because you can use CSS styling to make the gadget seamlessly match your theme. The result is profile pages that still look unified, but are at least partially configurable by the individual faculty member.

An example

Here is a portion of a profile page from UCSF Profiles. Each gadget is there because the page owner selected it and configured it.



Sign in to edit your profile (add interests, mentoring, photo, etc.)

Douglas Bauer, MD

Title	Professor
School	UCSF School of Medicine
Department	Medicine
Address	185 Berry Street W San Francisco CA 94143
Phone	415-514-8658
Email	dbauer@psg.ucsf.edu

Overview | Interests | Featured Publications | Websites | Featured Videos | In The News | Awarded Grants | Twitter | More Info | Publications

Overview

Since 1992 I have been a clinician-investigator in the Division of General Internal Medicine at UCSF where I maintain a general medicine practice (Mt Zion) and teach. As an executive member of the San Francisco Coordinating Center I direct the Endpoints Group. I started and currently direct the CTSI-funded Resident Research Training Program at UCSF. My research interests include the etiology, diagnosis and treatment of osteoporosis and other skeletal disorders, clinical biomarkers, and thyroid dysfunction.

Interests

Clinical epidemiology, osteoporosis, thyroid dysfunction, biomarkers

Featured Publications

- Bauer DC. **The calcium supplement controversy: now what?** J Bone Miner Res. 2014 Mar; 29(3):531-3.
  View in: PubMed
- Bauer DC. Clinical practice. Calcium supplements and fracture prevention. N Engl J Med. 2013 Oct 17; 369(16):1537-43.
  View in: PubMed
- Bauer DC. **Vertebral Augmentation vs Nonsurgical Therapy: Improved Symptoms, Improved Survival, or Neither?** JAMA Intern Med. 2013 Sep 9; 173(16):1522-3.
  View in: PubMed
- Rodondi N, Bauer DC. **Subclinical hypothyroidism and cardiovascular risk: how to end the**

Websites

- California Technology Assessment Forum
- Clinical Profile at UCSF Medical Center
- SF Coordinating Center
- UCSF Division of General Internal Medicine Profile
- UCSF Resident Research Training Program

Featured Videos

Osteoporosis: Update on Diagnosis and Trea

0:00 / 1:20:15

In The News

- Calcium Supplements and Fracture Risk (October 17, 2013)
- Calcium and Cardiovascular Disease (October 4, 2013)

More Info

Click below to filter Douglas C Bauer's research:

osteoporotic osteoporosis bmd bone mineral density osteoporotic fractures dwell spine fracture risk postmenopausal woman mineral density s-d x-ray hip fracture bone density density bmd femoral

See full profile on KNODE including:

Annual Activity

1990 to 2014

229 papers | 14 grants | 0 trials | 0 patents

Publications

Publications listed below are automatically derived from MEDLINE/PubMed and other sources, which might result in incorrect or missing publications. Researchers can **login** to make corrections and additions, or **contact us for help.**

List All | Timeline

1. Wirth CD, Blum MR, da Costa BR, Baumgartner C, Collet TH, Medici M, Peeters RP, Aujesky D, Bauer DC, Rodondi N. Subclinical Thyroid Dysfunction and the Risk for Fractures: A Systematic Review and Meta-analysis. Ann Intern Med. 2014 Aug 5; 161(3):189-99.
   View in: PubMed
2. Lee DS, Markwardt S, Goeres L, Lee CG, Eckstrom E, Williams C, Fu R, Orwoll E, Cawthon PM, Stefanick ML, Mackey D, Bauer DC, Nielson CM. Statins and physical activity in older men: the osteoporotic fractures in men study. JAMA Intern Med. 2014 Aug 1; 174(8):1263-70.
   View in: PubMed
3. Weiler S, Gemperli A, Collet TH, Bauer DC, Zimmerli L, Cornuz J, Battegay E, Gaspoz JM, Kerr EA, Aujesky

Douglas's Networks

Related Concepts
Derived automatically from this person's publications.
- Osteoporosis
- Fractures, Bone
- Osteoporosis, Postmenopausal
- Alendronate
- Bone Density
- See all (879) concept(s)

Co-Authors
People in Profiles who have published with this person.
- Black, Dennis
- Cummings, Steve
- Nevitt, Michael
- Vittinghoff, Eric
- Schwartz, Ann
- See all (26) people

Related Authors
People who share related concepts with this person.
- Black, Dennis
- Cummings, Steve
- Link, Thomas
- Majumdar, Sharmila
- Nevitt, Michael
- See all (60) people

Same Department
- Duong, Bao
- Feldman, Mitchell
- Koth, Laura
- Marya, Rupa
- Nazar, Babak
- Search Department

UCSF | chatter

Login to create a UCSF Chatter group right from UCSF Profiles!

OpenSocial

The OpenSocial standard was developed to make it easy for developers to add functionality to social networking systems like Google and MySpace. OpenSocial has lost popularity in social networking, but is becoming more favored in enterprise systems.

The Clinical and Translational Science Institute at UCSF created a project to host OpenSocial gadgets in the Harvard Profiles system. In keeping with the cross-platform origins of OpenSocial, the CTSI team decided to adapt their gadgets for use in VIVO as well.

ORNG

Social networking systems provide very little information about their participants. The group at CTSI wanted to combine the display tools of OpenSocial with the data structure of VIVO RDF. They accomplished this through an extension to the standard, which they called Open Research Networking Gadgets, or ORNG.

8.3.16.2 Adding gadgets to VIVO

The gadgets used at UCSF are provided in a library. Some are written specifically for UCSF, or specifically for the Harvard Profiles platform. However, many are available for use in VIVO.

You can also create your own gadgets. The gadgets are written in JavaScript, and you can use the existing gadgets as coding examples.

Under your control

The VIVO administrators select which gadgets will be available for the site. They also decide where the gadgets will appear on a profile page, if enabled.

Under control of your faculty

Each page owner may choose to enable individual gadgets for their page. A gadget may be written to accept settings that allow further configuration of its content and appearance.

8.3.16.3 Getting started

The VIVO Installation Instructions contain a section on how to add OpenSocial gadgets to a VIVO site. This will require some setup, and re-deploying VIVO. Once those steps are completed, your gadget library is configured by settings in a MySQL database table, and the gadget appearance is controlled by your Freemarker templates and CSS files.

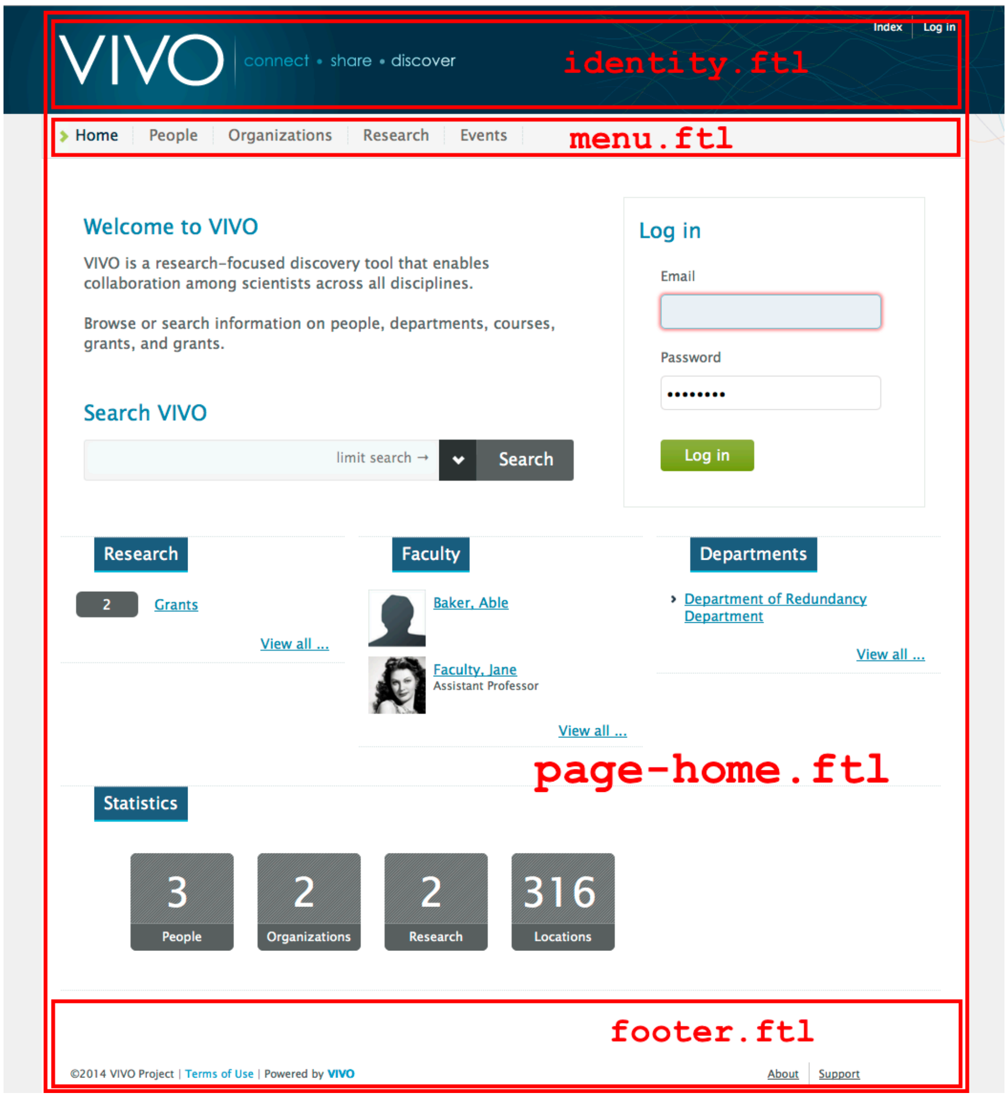
For more information about Open Research Networking Gadgets, see the ORNG web site[140].

8.3.17 How VIVO creates a page

140 http://www.orng.info/

8.3.17.1 The home page

Like the title page of a book, it is not unusual for the home page of a web site to be different from all other pages. In the default VIVO theme, the most significant difference is that the search box is moved from the header to a more prominent location on the page.



Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and grants.

2 Grants
View all ...

Baker, Able

Faculty, Jane
Assistant Professor

View all ...

Department of Redundancy Department

View all ...

3	2	2	316
People	Organizations	Research	Locations

©2014 VIVO Project | Terms of Use | Powered by VIVO

The following templates are used in the home page.

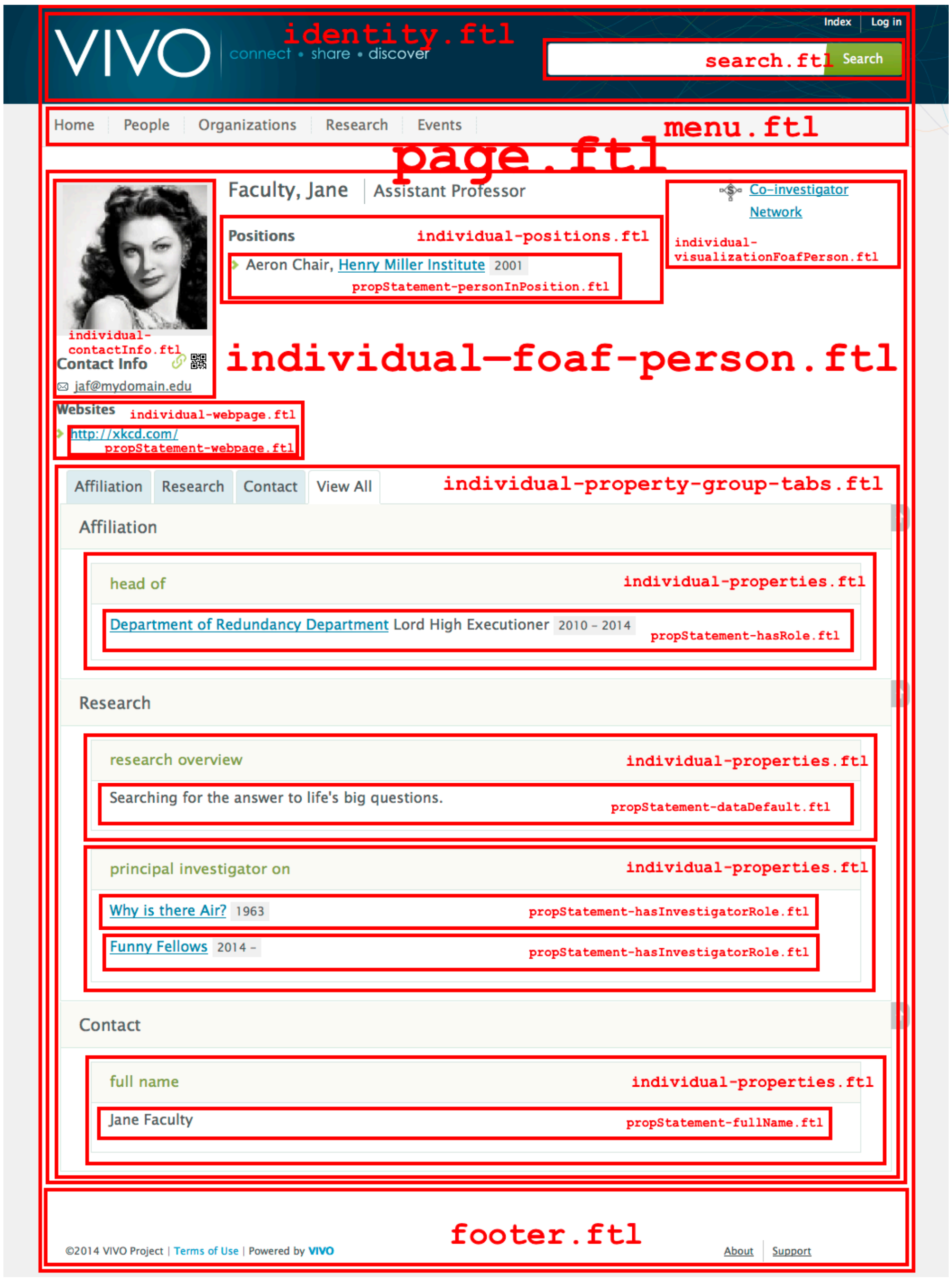
```
pageSetup.ftl
page-home.ftl
    head.ftl
        stylesheets.ftl
        headScripts.ftl
    identity.ftl
        languageSelector.ftl
    menu.ftl
```

```
        developer.ftl
    footer.ftl
        scripts.ftl
        googleAnalytics.ftl
```

Template	Purpose	From
`pageSetup.ftl`	Sets some class and formatting parameters.	Included in every page, by `TemplateProcessingHelper.java`
`page-home.ftl`	The special template used for the home page.	Specified as the page template by `HomePageController.java`, overriding the default `page.ftl`.
`head.ftl`	Creates the HTML `<HEAD>` tag.	Included by `page-home.ftl`.
`stylesheets.ftl`	Inserts links to CSS stylesheets.	Included by `head.ftl`.
`headScripts.ftl`	Inserts links to JavaScript files that must appear in the `<HEAD>` section of the page. These are somewhat unusual, since most JavaScript links appear at the end of the page.	Included by `head.ftl`.
`identity.ftl`	Draws the heading of the heading of the page, including the VIVO logo and the `Index` and `Log in` links.	Included by `page-home.ftl`.
`languageSelector.ftl`	Allows the user to select their preferred language. If the site supports only one language, this template has no effect.	Included by `identity.ftl`.
`menu.ftl`	Displays the page links (`Home`, `People`, etc.) at the top of the page.	Included by `page-home.ftl`.
`developer.ftl`	Displays the developer panel, used when testing and monitoring VIVO operation. If developer mode has not been enabled, this templates produces nothing.	Included by `menu.ftl`.
`footer.ftl`	Draws the footer of the page, including the copyright notice, and the `About` and `Support` links.	Included by `page-home.ftl`.
`scripts.ftl`	Inserts links to JavaScript files. Compare to `headScripts.ftl`.	Included by `footer.ftl`.
`googleAnalytics.ftl`	Inserts JavaScript code to work with Google Analytics. By default, this is commented out, since each site will need to insert their own ID values in order to produce meaningful results.	Included by `footer.ftl`.

8.3.17.2 A profile page

By numbers, the vast majority of pages on a VIVO site are profile pages. These are all likely to be structured around the properties of each individual. However, the format can be very different depending on whether that individual is a person, an organization, or a research grant.


VIVO
connect • share • discover
identity.ftl
Index
Log in
search.ftl
Search
Home
People
Organizations
Research
Events
menu.ftl
page.ftl
Faculty, Jane
Assistant Professor
Co-investigator Network
individual-visualizationFoafPerson.ftl
Positions
individual-positions.ftl
Aeron Chair, Henry Miller Institute 2001
propStatement-personInPosition.ftl
individual-contactInfo.ftl
Contact Info
jaf@mydomain.edu
individual–foaf-person.ftl
Websites
individual-webpage.ftl
http://xkcd.com/
propStatement-webpage.ftl
Affiliation
Research
Contact
View All
individual-property-group-tabs.ftl
Affiliation
head of
individual-properties.ftl
Department of Redundancy Department Lord High Executioner 2010 - 2014
propStatement-hasRole.ftl
Research
research overview
individual-properties.ftl
Searching for the answer to life's big questions.
propStatement-dataDefault.ftl
principal investigator on
individual-properties.ftl
Why is there Air? 1963
propStatement-hasInvestigatorRole.ftl
Funny Fellows 2014 -
propStatement-hasInvestigatorRole.ftl
Contact
full name
individual-properties.ftl
Jane Faculty
propStatement-fullName.ftl
footer.ftl
©2014 VIVO Project | Terms of Use | Powered by VIVO
About
Support

The following templates are used in this particular profile page. As explained in the notes, the choice of templates is driven in part by the content of the page.

```
pageSetup.ftl
page.ftl
    head.ftl
        stylesheets.ftl
        headScripts.ftl
    identity.ftl
        languageSelector.ftl
    search.ftl
    menu.ftl
        developer.ftl
    individual--foaf-person.ftl
        individual-setup.ftl
        individual-orcidInterface.ftl
        individual-contactInfo.ftl
        individual-webpage.ftl
            propStatement-webpage.ftl
        individual-visualizationFoafPerson.ftl
        individual-adminPanel.ftl
        individual-positions.ftl
            propStatement-personInPosition.ftl
        individual-overview.ftl
        individual-researchAreas.ftl
        individual-geographicFocus.ftl
    individual-openSocial.ftl
    individual-property-group-tabs.ftl
        individual-properties.ftl
            propStatement-hasRole.ftl
        individual-properties.ftl
            propStatement-dataDefault.ftl
            propStatement-hasInvestigatorRole.ftl
            propStatement-hasInvestigatorRole.ftl
        individual-properties.ftl
            propStatement-fullName.ftl
    footer.ftl
        scripts.ftl
        googleAnalytics.ftl
```

Template	Purpose	From
`pageSetup.ftl`	*as above.*	
`page.ftl`	The master template for most VIVO pages.	Specified by `FreemarkerHttpServlet.java`.

Template	Purpose	From

`head.ftl`
 	
`stylesheets.ftl`
 	
`headScripts.ftl`
`identity.ftl`
 	
`languageSelector.ftl`
	as above.	

`search.ftl`	Draws the search box in the header of the page.	Included by `page.ftl`.

`menu.ftl`
 	
`developer.ftl`
	as above.	

`individual--foaf-person.ftl`	The main body of the profile page.	VIVO is configured to use this template as the body of a profile page for any `foaf:Person`. **You can change this configuration: see** Class-specific templates for profile pages (see page 154). This is specified in `initialTBoxAnnotations.n3`, and recognized by `IndividualResponseBuilder.java` and `IndividualTemplateLocator.java`.
`individual-setup.ftl`	Sets some basic values for the following templates to use.	Included by `individual--foaf-person.ftl`.
`individual-orcidInterface.ftl`	Implements the VIVO integration to ORCiD. If this integration is not enabled, this template has no effect.	Included by `individual--foaf-person.ftl`.
`individual-contactInfo.ftl`	Displays the person's phone numbers and email addresses.	Included by `individual--foaf-person.ftl`.
`individual-webpage.ftl`	Displays the person's preferred web pages.	Included by `individual--foaf-person.ftl`.

Template	Purpose	From
`propStatement-webpage.ftl`	Displays a link to one of the person's preferred web pages.	VIVO is configured to use this template when displaying preferred web pages. **You can change this configuration: see** Custom List View Configurations (see page 159)**.** This is specified in `listViewConfig-webpage.xml`, which is specified in `PropertyConfig.n3` and `vivoListViewConfig.rdf`.
`individual-visualizationFoafPerson.ftl`	Displays the visualization links for co-authors, co-investigators	Included by `individual--foaf-person.ftl`.
`individual-adminPanel.ftl`	Displays links for a VIVO administrator to use when editing this person's information	Included by `individual--foaf-person.ftl`.
`individual-positions.ftl`	Displays the positions that this person currently holds.	Included by `individual--foaf-person.ftl`.
`propStatement-personInPosition.ftl`	Displays one position that this person currently holds.	VIVO is configured to use this template when displaying positions. **You can change this configuration: see** Custom List View Configurations (see page 159)**.** This is specified in `listViewConfig-personInPosition.xml`, which is specified in `PropertyConfig.n3`.
`individual-overview.ftl` `individual-researchAreas.ftl` `individual-geographicFocus.ftl`	Display additional information about the person.	Included by `individual--foaf-person.ftl`.
`individual-openSocial.ftl`	Implements the VIVO integration to OpenSocial gadgets. If this integration is not enabled, this template has no effect.	**You can configure VIVO to display OpenSocial gadgets on profile pages: see** Using OpenSocial Gadgets (see page 213)**.** Included by `individual--foaf-person.ftl`.

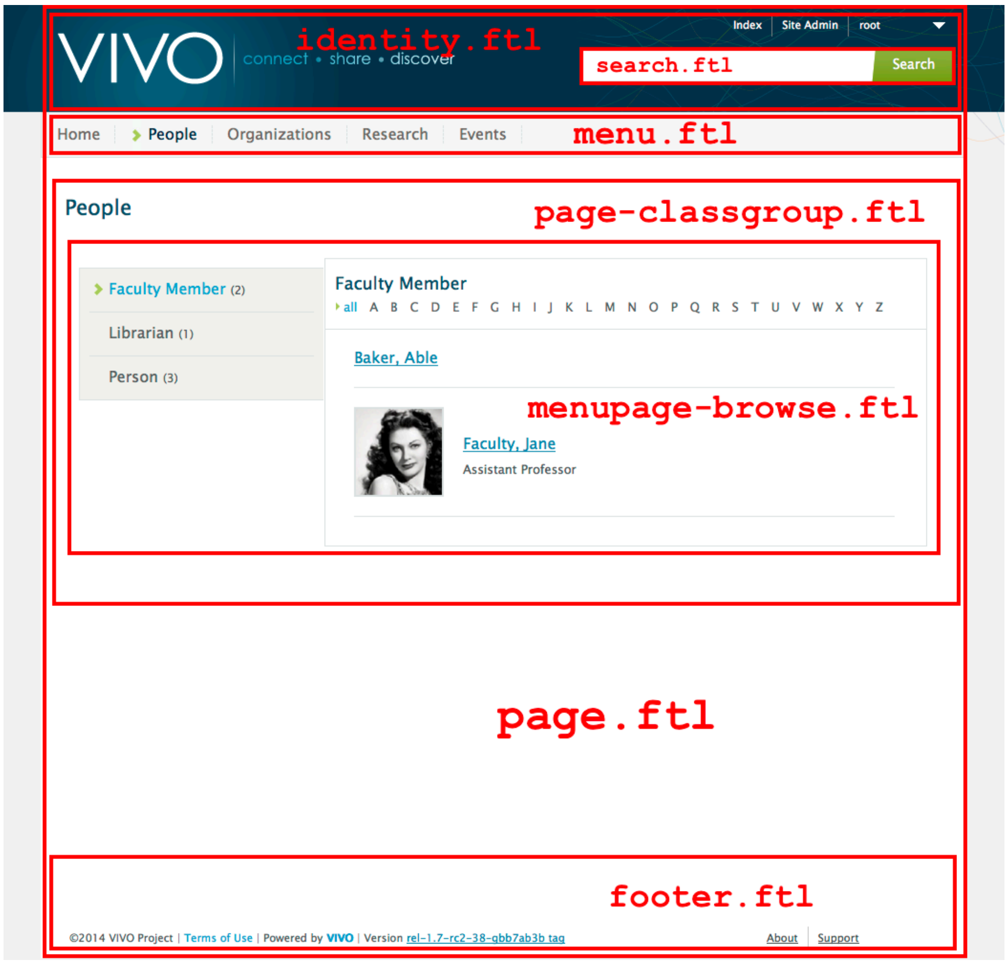
Template	Purpose	From
`individual-property-group-tabs.ftl`	Displays the groups of properties for this person.	Included by `individual--foaf-person.ftl`.

`individual-properties.ftl`
 	
`propStatement-hasRole.ftl` `individual-properties.ftl`
 	
`propStatement-dataDefault.ftl`
 	
`propStatement-hasInvestigatorRole.ftl`
 	
`propStatement-hasInvestigatorRole.ftl` `individual-properties.ftl`
 	
`propStatement-fullName.ftl`
	Each invokation of `individual-properties.ftl` displays a property group.
	
	Each subordinate template displays one property for this person.
		Each reference to `individual-properties.ftl` is included by `individual-property-group-tabs.ftl`.
		
		VIVO is configured to use these subordinate templates when displaying research overview, roles, and names. **You can change this configuration: see** Custom List View Configurations (see page 159)**.**

`footer.ftl`
 	
`scripts.ftl`
 	
`googleAnalytics.ftl`
	as above.	

8.3.17.3 The People page

The page management GUI provides an easy way for VIVO administrators to create simple pages. These pages may also be added to the menu bar. By default, VIVO is configured with eleven such pages. Five of them are listed in the menu.



The following templates are used in the People page, and in other pages that allow users to browse through a class group.

```
pageSetup.ftl
page.ftl
    head.ftl
        stylesheets.ftl
        headScripts.ftl
    identity.ftl
        languageSelector.ftl
    search.ftl
    menu.ftl
        developer.ftl
    page-classgroup.ftl
        menupage-checkForData.ftl
        menupage-browse.ftl
        menupage-scripts.ftl
    footer.ftl
        scripts.ftl
        googleAnalytics.ftl
```

Template	Purpose	From

`pageSetup.ftl`
`page.ftl`
`head.ftl`
`stylesheets.ftl`
`headScripts.ftl`
`identity.ftl`
`languageSelector.ftl`
`search.ftl`
`menu.ftl`
`developer.ftl`
	as above.	

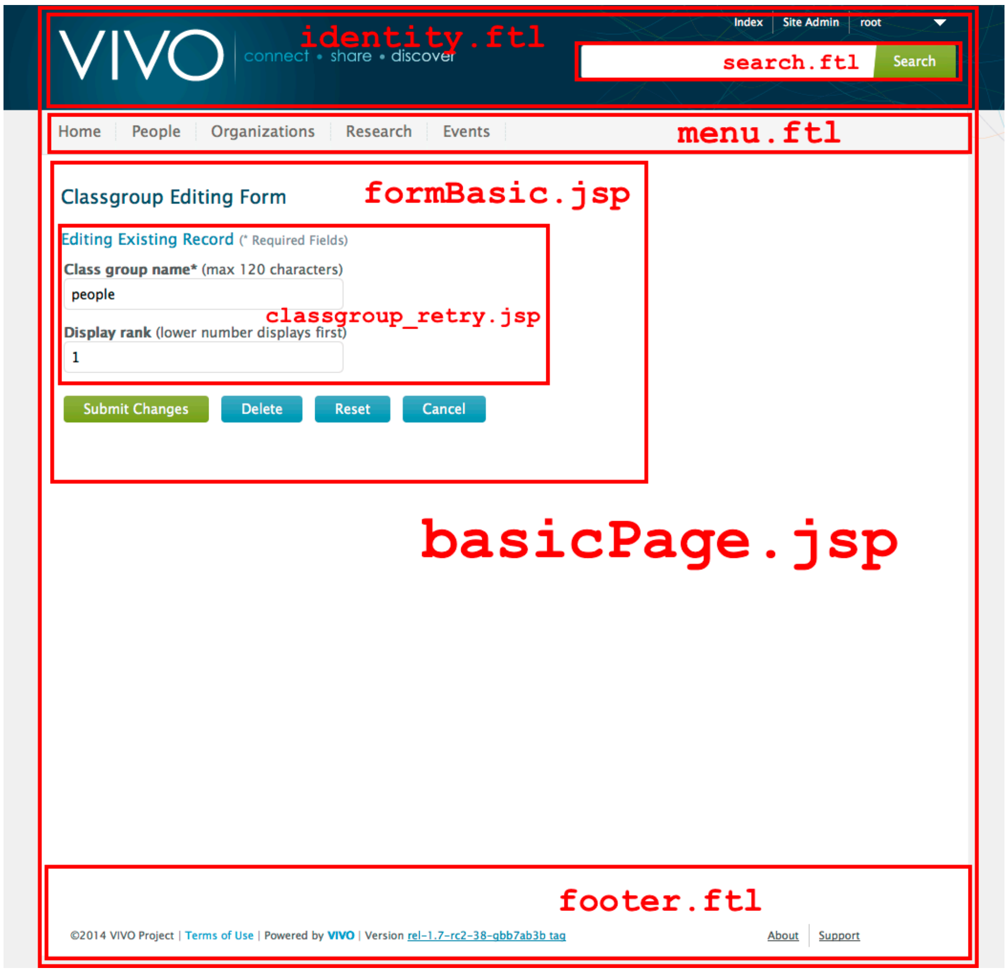
`page-classgroup.ftl`	Combines the components to create an AJAX-driven page that browses among the classes in a class group.	VIVO is configured to use this template in the `People` menu page page. **You can change this configuration: see** Menu and page management (see page 136)**.** This template is invoked by `ClassGroupPageData.java`, which is assigned to the `People` page in `menu.n3`.
`menupage-checkForData.ftl`	Checks to see if the page will be empty. Displays messages suitable to a VIVO administrator or to another user, depending on who is viewing the page.	Included by `page-classgroup.ftl`.
`menupage-browse.ftl`	Creates the page context that will be filled by AJAX calls.	Included by `page-classgroup.ftl`.

Template	Purpose	From
`menupage-scripts.ftl`	Creates or links to the JavaScripts used in browsing among classes of individuals.	Included by `page-classgroup.ftl`.

`footer.ftl`
 	
`scripts.ftl`
 	
`googleAnalytics.ftl`
	as above.	

8.3.17.4 A back-end page

VIVO provides several pages that allow administrators to edit the classes and properties in the ontology, and to create or adjust class groups and property groups. These pages are built around the older JSP (Java Server Pages) technology, although the header and footer are created from the same Freemarker templates as other pages.



Classgroup Editing Form

Editing Existing Record (* Required Fields)

Class group name* (max 120 characters)
people

Display rank (lower number displays first)
1

©2014 VIVO Project | Terms of Use | Powered by VIVO | Version rel-1.7-rc2-38-gbb7ab3b tag

The following templates and JSPs are used in creating this page.

```
basicPage.jsp
    head.ftl
        stylesheets.ftl
        headScripts.ftl
    identity.ftl
        languageSelector.ftl
    search.ftl
    menu.ftl
        developer.ftl
    formBasic.jsp
        classgroup_retry.jsp
    footer.ftl
        scripts.ftl
        googleAnalytics.ftl
```

Template	**Purpose**	**From**
`basicPage.jsp`	The master template for the VIVO back-end pages.	Specified by `ClassgroupRetryController.java`.

`head.ftl`
`stylesheets.ftl`
`headScripts.ftl`
`identity.ftl`
`languageSelector.ftl`
`search.ftl`
`menu.ftl`
`developer.ftl`
	as above.	

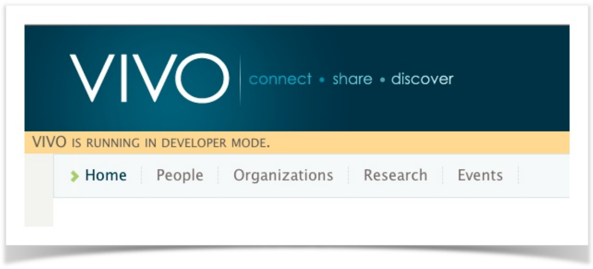
`formBasic.jsp`	A generic frame that provides title and buttons for an edit.	Specified by `ClassgroupRetryController.java`.
`classgroup_retry.jsp`	Shows the fields that may be edited for a class group.	Specified by `ClassgroupRetryController.java`.

`footer.ftl`
`scripts.ftl`
`googleAnalytics.ftl`
	as above.	

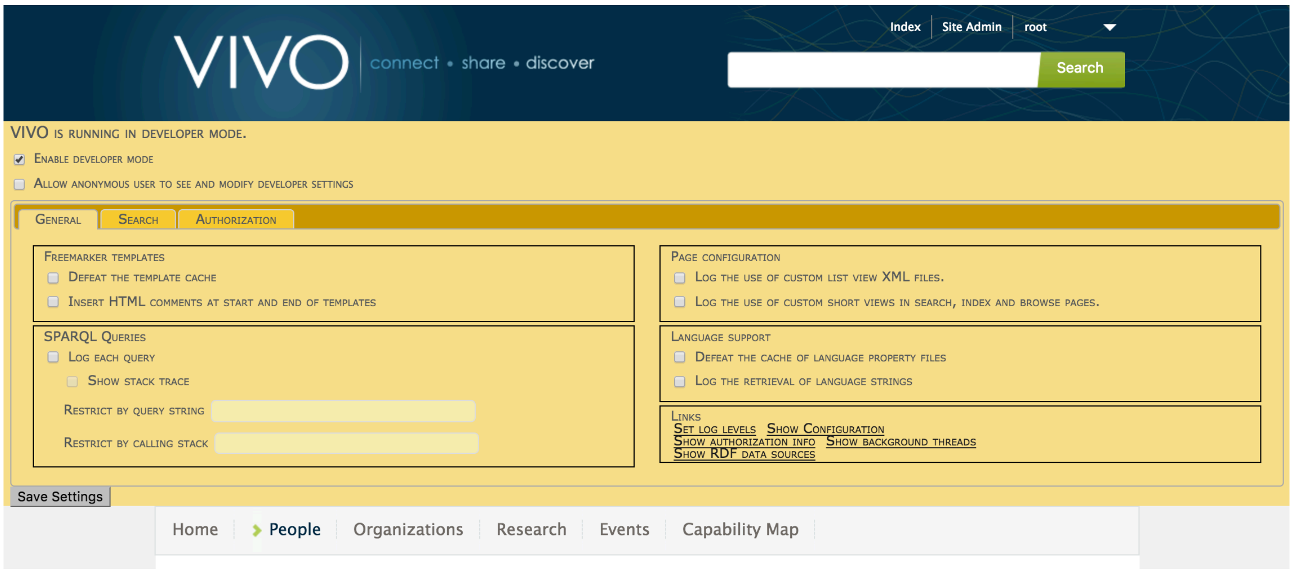
8.3.18 Tips for Interface Developers

8.3.18.1 Use the Developer Panel

Many of these techniques involve the Developer Panel. You can start the Developer Panel at Site Admin > Activate Developer Panel. When the Developer Panel has been activated, you will see:


VIVO
connect • share • discover
VIVO is running in developer mode.
Home
People
Organizations
Research
Events

When you click on the Developer Mode banner, you will see:



VIVO is running in developer mode.
- [✓] Enable developer mode
- [ ] Allow anonymous user to see and modify developer settings

General | Search | Authorization

Freemarker templates
- [ ] Defeat the template cache
- [ ] Insert HTML comments at start and end of templates

SPARQL Queries
- [ ] Log each query
  - [ ] Show stack trace
- Restrict by query string
- Restrict by calling stack

Page configuration
- [ ] Log the use of custom list view XML files.
- [ ] Log the use of custom short views in search, index and browse pages.

Language support
- [ ] Defeat the cache of language property files
- [ ] Log the retrieval of language strings

Links
Set log levels Show Configuration
Show authorization info Show background threads
Show RDF data sources

Save Settings

To close the Developer Panel, unselect "Enable Developer Mode" in the upper left hand corner, and press "Save Settings" in the lower left hand corner.

Developer Panel Settings

You can change settings on The Developer Panel interactively, while VIVO is running, or you can use a `developer.properties` file in your VIVO home directory.

A typical developer.properties file

```
developer.enabled=true
developer.permitAnonymousControl=true
developer.defeatFreemarkerCache=true
```

When any feature of The Developer Panel is active, you will see this indicator in the header of your VIVO pages:

This is to remind you that developer options may slow down your VIVO, and should not be used in production.

8.3.18.2 Iterate your code more quickly

Don't restart VIVO until you need to

VIVO will detect changes to the templates without requiring a restart. However, you will probably want to defeat the Freemarker cache (see below).

Also, VIVO will serve the latest version of CSS, JavaScript, or image files. For these files, however, you may need to clear the cache in your browser. Instructions for doing this will differ, depending on which browser you are using. If you don't know how to reset the cache in your browser, you may want to consult this web site: http://clearyourcache.com/, or just search the web for "clear browser cache".

If you change any other types of files, you will need to restart VIVO after running the build script.

Defeat the Freemarker cache

As mentioned above, VIVO will detect changes to Freemarker templates. By default, however, VIVO will not detect the changes immediately. The Freemarker framework caches the templates that it uses, and won't even look to see if a template has changed until 1 minute after it was last read from disk. In a production system, of course, that makes the accessing much more efficient. When you are making frequent changes, it's an annoyance.

Use The Developer Panel to defeat the Freemarker cache.

Customizing listViewConfigs

Ted Lawless has written an open-source Python script to assist with viewing the output of a listViewConfig[141] without having to rebuild the entire Vivo app.

Also, you can skip the unit tests when building VIVO. Unit tests do not apply to listViewConfigs.

8.3.18.3 Reveal what VIVO is doing

Insert template delimiters in the HTML

It's not always clear which template has created a particular piece of your HTML page. Templates include other templates, templates are invoked in custom list views, short views, etc. You can use The Developer Panel to insert comments in the HTML that tell you where each template begins and ends.

For example, this section of a page was produced mostly by the `identity.ftl` template. The `languageSelector.ftl` template is included, but does not generate any HTML. The next section is produced by the `menu.ftl` template, and so on.

141 http://lawlesst.github.io/notebook/vivo-listview.html

```
...

    <body >
            <!-- FM_BEGIN identity.ftl -->

<header id="branding" role="banner">

  <h1 class="vivo-logo"><a title="VIVO | connect share discover" href="/vivo">
    <span class="displace">VIVO</span>
  </a></h1>

  <nav role="navigation">
    <ul id="header-nav" role="list">
<!-- FM_BEGIN languageSelector.ftl -->
<!-- FM_END languageSelector.ftl -->
        <li role="listitem"><a href="/vivo/browse" title="Index">Index</a></li>
        <li role="listitem"><a href="/vivo/siteAdmin" title="Site Admin">Site Admin</a></li>
        <li>
          <ul class="dropdown">
            <li id="user-menu"><a href="#" title="user">Jim</a>
              <ul class="sub_menu">
                <li role="listitem"><a href="/vivo/accounts/myAccount" title="My account">My account</a></
li>
                <li role="listitem"><a href="/vivo/logout" title="Log out">Log out</a></li>
              </ul>
            </li>
          </ul>
        </li>
      </ul>
    </nav><!-- FM_END identity.ftl -->
            <!-- FM_BEGIN menu.ftl -->
</header>

...
```

8.4 Deploying additional ontologies with VIVO

The most straightforward way to load additional ontologies into VIVO is to use the Add/Remove RDF Data feature shown on the Site Admin page. This loads an ontology directly into the triple store. The disadvantage is that all additional ontologies and local edits are loaded into a single graph. This can make it cumbersome to update individual ontologies to reflect edits made outside of VIVO.

8.4.1 Filegraphs

There is another mechanism for incorporating ontologies into VIVO. This involves "filegraphs," and is how the VIVO-ISF ontology is included with the software. Filegraphs are RDF documents stored in the VIVO home directory. Each filegraph corresponds to a single graph in the triple store. Every time Tomcat starts, VIVO checks each of these graphs to ensure that its contents exactly match the triples found in the corresponding file. If the file has changed, VIVO makes the necessary modifications to the triple store. If a filegraph is removed from its directory, its graph in the triple store will be deleted the next time Tomcat starts.

8.4.1.1 Example

```
vitro.home/
    rdf/
        tbox/
            filegraph/
                agent.owl
                appControls-temp.n3
                bfo-bridge.owl
                bfo.owl
                ...
                myOntology.owl
                ...
```

Adding myOntology.owl to the directory shown above will automatically create the corresponding graph in the triple store after Tomcat is restarted:

`http://vitro.mannlib.cornell.edu/filegraph/tbox/myOntology.owl`

Modifying or removing the myOntology.owl file in the filegraph directory and restarting Tomcat will automatically update the ontology VIVO.

8.4.2 Namespace Prefixes

Additional ontologies, whether directly imported via 'Add/Remove RDF data' or implemented as filegraphs, are listed in the ontology list ('Site Admin > Ontology list'). While the ontologies name and namespace are automatically added to the list, the prefix is not. Instead, the note '(not yet specified)' appears. This behavior occurs even if the prefix is correctly specified in the RDF file.

For ontologies that are added to an existing VIVO installation, the prefix needs to be entered manually into the ontology list. If the additional ontology is to be provided with the software before installation, however, the prefix to be added automatically during the build process can be specified beforehand.

Note

The following procedure is only relevant if you want to add an ontology before the software is installed on a server.

VIVO keeps an internal record of prefixes that is read from the /rdf/tbox/firsttime/ directory. The prefixes of the ontologies that are loaded with VIVO are specified in the 'initialTBoxAnnotations.n3' file. You can add an additional prefix by adding the following lines either to this file or to a separate file:

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix vitro:   <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
```

```
<http://*URI/of/the/added/ontology*>
        rdfs:label "*Name of added ontology*" @en-US;
        vitro:ontologyPrefixAnnot "*OntologyPrefix*"^^xsd:string
```

Of course, the strings enclosed by asterisks need to be adapted according to your custom ontology. After VIVO is built, you should find the new ontology in the ontology list, with its specified prefix.

8.5 Enable an external authentication system

8.5.1 How User Accounts are Associated with Profile Pages

There are three elements in the linkage between a User Account and a Profile page:

- The user account holds the `externalAuthId`
- `runtime.properties` specifies the `URI` of the matching property
- The profile page must have a property with that `URI` whose value matches the `externalAuthId`. (The property value is either a String or an untyped literal.)

8.5.1.1 A user account may have an externalAuthId

- The `externalAuthId` is optional.
- There are several ways to create a `externalAuthId`:
  - If you are using internal authentication – managed within VIVO — then each account must be created by an admin, and the admin may choose to set the externalAuthId to a useful value.
  - If you are using external authentication – Shibboleth, or the like – then when a user without an account passes authentication, an account is created auto-magically. The externalAuthId is set to the user's Shibboleth ID.
  - Regardless of the type of authentication, you could choose to ingest the information for the user accounts, and create the externalAuthId as part of that ingest.
- In any case, the externalAuthId can be used to link to the user's profile page.
- This info is stored in the userAccounts model.
- You can confirm this by going to the SiteAdmin page, clicking on "Ingest Tools", then "Manage Jena Models", then the button labelled "RDB Models", then the "output model" link under vitro-kb-userAccounts. The output should contain statements that look something like this:

```
<http://vivo.mydomain.edu/individual/u8041>
a <http://vitro.mannlib.cornell.edu/ns/vitro/authorization#UserAccount> ;
```

```
<http://vitro.mannlib.cornell.edu/ns/vitro/authorization#emailAddress>
"jeb228@cornell.edu"^^xsd:string ;
<http://vitro.mannlib.cornell.edu/ns/vitro/authorization#externalAuthId>
"jeb228"^^xsd:string ;
```

- I don't know what would happen to a user with more than one one externalAuthID. Probably VIVO will arbitrarily choose among them.

8.5.1.2 runtime.properties may contain a value for selfEditing.idMatchingProperty

- You can confirm this value by looking in the `vivo.all.log` file in Tomcat logs. Each time VIVO starts up, the first entry written to the log contains all of the properties from `runtime.properties`. It helps to inspect this if you might possibly be reading the wrong runtime.properties file.
- At Cornell, ours looks like this:

```
selfEditing.idMatchingProperty = http://vivo.cornell.edu/ns/hr/0.9/hr.owl#netId
```

8.5.1.3 The profile page may match the externalAuthId on the user account

- To associate a profile page with a user account, the Individual must have a data property whose URI is the one from runtime.properties, and whose value is equal to the externalAuthId of the user account.
- For example, the Individual object that forms the basis for my profile page contains a statement like this:

```
<http://vivo.cornell.edu/individual/JamesBlake>
    <http://vivo.cornell.edu/ns/hr/0.9/hr.owl#netId>
    "jeb228"
    .
```

- You can confirm this by logging in as an admin, navigating to the profile page, clicking on "edit this individual" and then the button labelled "Raw Statements with this Resource as Subject"
- In the example above, the "netId" field is set to an untyped Literal. A String Literal will work also.

8.5.2 Using a Tomcat Realm for external authentication

8.5.2.1

Background

VIVO is not written to use the standard JEE or Tomcat authentication systems, so using a Tomcat Realm would require some customization. This doesn't seem very difficult, it just hasn't been a priority for us.

When VIVO is set up to use external authentication, it uses a reverse-proxy setup, where an Apache HTTP server intercepts all calls to Tomcat. The Apache server uses a Shibboleth module or other module to secure a particular page: http://localhost:8080/vivo/loginExternalAuthReturn.

If an HTTP request is made to that page, and the request does not belong to a session that is already logged it, the Shibboleth module in Apache will intercept the request and guide the user through the authentication process. When the user's credentials are accepted, the module invokes the secured page, as requested, storing the user's ID in one of the HTTP headers. The VIVO code reads the user ID from the HTTP header and stores it in the session object. Only that one page is secured, and VIVO remembers the user ID for use in subsequent requests.

Which HTTP header will VIVO inspect for the user ID? The header which is named in externalAuth.netIdHeaderName.

Most institutions that use VIVO also use Shibboleth in their web applications, or something with a similar mechanism. The IT group at the institution provides the VIVO implementers with the appropriate Apache module and configuration information.

I don't know of anyone who has tried to use a Tomcat Realm to accomplish external authentication in VIVO. I think it would require some small modification of the VIVO code, perhaps a change to ExternalAuthHelper.getExternalAuthId(). Tomcat would use the Realm to create a Principal object in the HTTP request, and VIVO would get the user ID from that Principal instead of looking in an HTTP header. Web.xml would be modified to secure the page, as you have already done.

8.5.2.2 Testing

It really was just that easy!

I added these lines to ExternalAuthHelper.getExternalAuthId(), right after the check for a null request object:

```
Principal p = request.getUserPrincipal();
if (p != null) {
   log.debug("Found a UserPrincipal in the request: " + p);
   String userId = p.getName();
   if (StringUtils.isNotEmpty(userId)) {
       log.debug("Got external auth from UserPrincipal: " + userId);
       return userId;
   }
}
```

I added these lines to the end of web.xml, just before the closing </web-app>:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>ExternalAuthPage</web-resource-name>
        <url-pattern>/loginExternalAuthReturn</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>tomcat</role-name>
    </auth-constraint>
</security-constraint>

<login-config>
    <auth-method>BASIC</auth-method>
```

```
</login-config>
```

I set this property in deploy.properties:

```
externalAuth.buttonText = Log in using basic Tomcat
```

And voila, my tomcat-users.xml file is my external authentication system!

Obviously, you will want to use FORM authentication, instead of BASIC, and something other than the default Realm. But I expect you know how to do that already.

Please, let me know how this progresses for you. This may be something that we will add to the next release.

Jim Blake

8.6 Authorization

- Writing a controller for a secured page (see page 234)
- Creating a VIVO authorization policy - an example (see page 238)
- A more elaborate authorization policy (see page 244)
- The IdentifierBundle - who is requesting authorization? (see page 250)

8.6.1 Writing a controller for a secured page

8.6.1.1 Concepts

A secured page

A secured page in VIVO is one that can not be viewed by the general public. If an unauthorized user attempts to view a secured page, even by entering the URL directly into a browser, the attempt should fail.

How is a page secured?

To secure a page, the controller code requests authorization to perform a particular `RequestedAction`. If the user is not authorized to perform that action, the controller rejects the request. For example, the `RevisionInfoController` checks to see whether the user is authorized for the `SEE_REVISION_INFO.ACTION`. If the user is not authorized for that action, they will not see the Revision Info page.

Other controllers use more complicated tests to determine whether a user is authorized. For example, the `ManageProxiesAjaxController` permits access by any user who is authorized for either the `MANAGE_PROXIES.ACTION` or the `MANAGE_OWN_PROXIES.ACTION`.

Who may view a secured page?

A secured page can never be viewed by someone who is not logged in to VIVO. Since we don't know who the user is, we can't know whether they are authorized to view the page.

If a user is logged in, there is a list of `Identifiers` associated with their account. The `Identifiers` are pieces of information about that user, including their account URI, the URI of their profile page, their assigned role, any proxy permissions, and more. When a secured page is requested, these `Identifiers` are passed to the list of active `Policy` objects. Each `Policy` applies its own logic to determine whether the user may view the secured page.

What happens if the user is not authorized?

- If the user is logged in, but does not have authorization to view the secured page, the browser will be redirected to the VIVO home page. A message at the top of the page will state that the user is not authorized to view the page he requested.
- If the user is not logged in, the browser will be redirected to the VIVO login page. When the user logs in, the browser will be redirected to the secured page, and the test is repeated.
    - If the user is authorized, the secured page will be displayed.
    - If the user is not authorized, the home page will be displayed, as previously described.

What happens when a user logs out?

If a user is viewing an unsecured page, and clicks on the "Log out" link, the page will be refreshed. For some pages, particularly profile pages, the contents of the page may have changed. Many people appreciate this feature when editing their own profiles. Log in, and you can edit. Log out, and you can see what your page looks like to the public.

If a user is viewing a secured page and clicks on the "Log out" link, the browser will be redirected to the VIVO home page.

8.6.1.2 Requested Actions

Requested actions are usually quite simple. For example, the `RevisionInfoController` requests permission to display the revision info page. The user either has that permission or they do not.

On the other hand, Requested actions can be quite detailed. For example, the `ImageUploadController` requests permission to add or modify a particular triple in the data model. If the user is logged in as root or admin, they have permission. However, if the user is logged in as a self-editor, a complex algorithm is performed to determine whether they are authorized to add or modify the triple in question. They may be authorized because the subject of the triple is the URI of their own profile page, or because they have been given proxy rights to edit the page in question, or several other possibilities.

8.6.1.3 The most common case

The most common scenario for a secured page is when a simple, unparameterized action is requested, and the user either

- has a permission that provides authorization, or
- does not have that permission and is not authorized.

The steps

Decide on a permission and requested action

Simple permissions like this are usually implemented by the `SimplePermission` class, which also provides an implementation for the corresponding `RequestedAction`.

In some cases, it makes sense to re-use an existing instance of `SimplePermission`. So for example, `SimplePermission.USE_ADVANCED_DATA_TOOLS_PAGES` authorizes the user for any and all of the RDF ingest/export pages. In other cases, it makes more sense to create a new instance. So `SimplePermission.MANAGE_PROXIES` stands alone with only one usage.

For this example, we will look at `SimplePermission.SEE_REVISION_INFO`, which has only one usage.

Write the controller to require the requested action

If the controller extends `FreemarkerHttpServlet`, override the `requiredActions()` method, like this:

```
@Override
protected Actions requiredActions(VitroRequest vreq) {
    return SimplePermission.SEE_REVISION_INFO.ACTIONS;
}
```

If the controller exends `VitroHttpServlet` (but not `FreemarkerHttpServlet`), add a test to the `doGet()` and `doPost()` methods, like this:

```
@Override
public void doPost(HttpServletRequest req, HttpServletResponse resp) {
    if (!isAuthorizedToDisplayPage(req, resp,
            SimplePermission.SEE_REVISION_INFO.ACTIONS)) {
        return;
    }
...
```

Both of these examples take advantage of the fact that each instance of `SimplePermission` defines its own `RequestedAction`, as well as its own `Actions` set.

Grant the permission to the desired users.

Each `Permission`, simple or otherwise, can be assigned to `PermissionSets` within VIVO. Each user account is associated with a `PermissionSet`, and may use the `Permissions` associated with it. The assignment of `Permissions` to `PermissionSets` occurs in the file called

```
[vitro]/webapp/rdf/auth/everytime/permission_config.n3
```

By inspecting the RDF in this file, we can see that the `SEE_REVISION_INFO` permission is assigned to `ADMIN`, `CURATOR`, and `EDITOR` `PermissionSets`. Here is an excerpt of the file with the relevent RDF:

```
@prefix auth: <http://vitro.mannlib.cornell.edu/ns/vitro/authorization#> .
@prefix simplePermission: <java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#> .

auth:ADMIN auth:hasPermission simplePermission:SeeRevisionInfo .
auth:CURATOR auth:hasPermission simplePermission:SeeRevisionInfo .
auth:ADMIN auth:hasPermission simplePermission:SeeRevisionInfo .
```

In future versions of VIVO, the `Permission/PermissionSet` framework may be extended to permit multiple `PermissionSets` per user, with GUI-based configuration.

8.6.1.4 A more complex example

TBD

- It's all about the action that your controller is requesting, and whether your user has authorization to do it.
  - Actions can be parameterized (modify this statement) or not (see the revision info page)
  - Authorization can come from a policy, or from a permission
  - Permissions can be simple, or as complex as a policy
- Look at the simplest case: RevisionInfoController
  - Not parameterized: SimplePermission.something.ACTION
- Code in HttpServlet, FreemarkerServlet, JSP
- Look at a complex case: ImageUploadController
  - Also ManageProxiesAjaxController
- In some cases, it isn't a question of whether your controller will run, but what it will do:
  - BaseIndividualTemplateModel

  -
    ```
    public boolean isEditable() {
        AddDataPropertyStatement adps = new AddDataPropertyStatement(
                vreq.getJenaOntModel(), individual.getURI(),
                RequestActionConstants.SOME_URI);
        AddObjectPropertyStatement aops = new AddObjectPropertyStatement(
                vreq.getJenaOntModel(), individual.getURI(),
                RequestActionConstants.SOME_URI,
                RequestActionConstants.SOME_URI);
        return PolicyHelper.isAuthorizedForActions(vreq, new Actions(adps).or(aops));
    }
    ```
  - LoginRedirector

    -
      ```
      private boolean canSeeSiteAdminPage() {
          return PolicyHelper.isAuthorizedForActions(request,
                  SimplePermission.SEE_SITE_ADMIN_PAGE.ACTIONS);
      }
      ```
  - BaseSiteAdminController

```
• if (PolicyHelper.isAuthorizedForActions(vreq,
  SimplePermission.MANAGE_USER_ACCOUNTS.ACTIONS)) {
        data.put("userAccounts", UrlBuilder.getUrl("/accountsAdmin"));
    }
```

8.6.2 Creating a VIVO authorization policy - an example

8.6.2.1 Overview

The ability of users to access data in VIVO is controlled by a collection of Policy objects. By creating or controlling Policy objects, you can control access to the data.

The Policy objects are instances of Java classes that implement the `PolicyIface` interface. These objects are created when VIVO starts up, and are collected in the `ServletPolicyList`. When code in VIVO needs to know whether a user is authorized to perform a particular action, the code creates a `RequestedAction` object and passes it to the Policy list for approval.

When the list is asked for approval, the first Policy in the list is asked first. It must respond with a decision that is `AUTHORIZED`, `UNAUTHORIZED`, or `INCONCLUSIVE`. If the decision is `AUTHORIZED` or `UNAUTHORIZED`, it is taken to be final, and the other Policies in the list are not consulted. If the decision is `INCONCLUSIVE`, then the next Policy in the list is asked to approve the same request, and the process repeats until a conclusive answer is obtained, or until all policies have answered. If no Policy has answered with `AUTHORIZED`, the request fails.

The code below is an example of such a Policy. The entire class is available in the attached file[142].

8.6.2.2 The example

This Policy will check each request to edit an object property statement. The request will be rejected if the statement appears in any graph that is not in the approved set.

The use case is where an individual whose data is stored in the default graph `(vitro-kb2)` links to data in other graphs which were created by ingest and may not be edited. The result of this Policy is that there will be no edit link from the profile page of the individual to that data.

142 https://wiki.duraspace.org/download/attachments/96995961/RestrictEditingByGraphPolicy.java?api=v2&modificationDate=1522787193551&version=1

Lines 1-39: imports

```
 /* $This file is distributed under the terms of the license in /doc/license.txt$ */

package edu.cornell.mannlib.vitro.webapp.auth.policy;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import com.hp.hpl.jena.query.Dataset;
import com.hp.hpl.jena.query.Query;
import com.hp.hpl.jena.query.QueryExecution;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QueryFactory;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.query.Syntax;
import com.hp.hpl.jena.rdf.model.RDFNode;
import com.hp.hpl.jena.shared.Lock;

import edu.cornell.mannlib.vitro.webapp.auth.identifier.IdentifierBundle;
import edu.cornell.mannlib.vitro.webapp.auth.identifier.common.IsRootUser;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.Authorization;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.PolicyDecision;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.PolicyIface;
import edu.cornell.mannlib.vitro.webapp.auth.requestedAction.ifaces.RequestedAction;
import edu.cornell.mannlib.vitro.webapp.auth.requestedAction.propstmt.EditObjectPropertyStatement;
import edu.cornell.mannlib.vitro.webapp.dao.jena.QueryUtils;
import edu.cornell.mannlib.vitro.webapp.servlet.setup.JenaDataSourceSetupBase;
import edu.cornell.mannlib.vitro.webapp.startup.StartupStatus;
```

Import statements for the classes used in the Policy

Lines 40-56: Class declaration, variables, constructor

```
 /**
 * Deny authorization to edit a statement from one of the prohibited graphs.
 */
public class RestrictEditingByGraphPolicy implements PolicyIface {
    private static final Log log = LogFactory
```

```
            .getLog(RestrictEditingByGraphPolicy.class);

    private static final Syntax SYNTAX = Syntax.syntaxARQ;
    private static final Set<String> PERMITTED_GRAPHS = new HashSet<>(
            Arrays.asList(new String[] { "http://vitro.mannlib.cornell.edu/default/vitro-kb-2" }));

    private final Dataset dataset;

    public RestrictEditingByGraphPolicy(ServletContext ctx) {
        this.dataset = JenaDataSourceSetupBase.getStartupDataset(ctx);
    }
```

The class must implement the `PolicyIface` interface.

The constructor stores a reference to the `startupDataset`, which will be used to execute SPARQL queries. Because this reference is taken from the context, it will contend with all other context-based references for access to a single database connection. It would be more efficient to use a `Dataset` that was provided by the `HttpServletRequest`, but a Policy never has access to the Request. This will be changed in a future release. (See this JIRA issue[143].)

The `PERMITTED_GRAPHS` constant holds the set of graph URIs for which editing is permitted. It would be a simple code change to use a `PROHIBITED_GRAPHS` constant instead.

Lines 57-68: Implement the isAuthorized() method

```
    /**
     * For each request to Edit an ObjectProperty, find out what graph the
     * statement is in. Prohibit editing if the statement is in the wrong graph.
     *
     * Note that this will not work with a DataProperty, since the
     * EditDataProperty object does not contains the value of the property. We
     * didn't anticipate that editing privileges would be determined by the
     * contents of the string.
     */
    @Override
    public PolicyDecision isAuthorized(IdentifierBundle whoToAuth,
            RequestedAction whatToAuth) {
```

Every `PolicyIFace` class must implement this method.

- `whoToAuth` is a collection of `Identifiers`, each one holding a piece of information about the user who is currently logged in.
- `whatToAuth` is the action being requested.

Lines 69-81: Make quick and easy decisions

```
        if (whoToAuth == null) {
            return inconclusiveDecision("whoToAuth was null");
        }
```

143 https://jira.duraspace.org/browse/VIVO-269

```
        if (whatToAuth == null) {
            return inconclusiveDecision("whatToAuth was null");
        }
        if (IsRootUser.isRootUser(whoToAuth)) {
            return inconclusiveDecision("Anything for the root user");
        }
        if (!(whatToAuth instanceof EditObjectPropertyStatement)) {
            return inconclusiveDecision("Only interested in editing object properties");
        }
```

Policies are called very frequently, especially when a large profile page is displayed. Whenever possible, answer the easy questions first before doing more expensive tests.

Checking for `null` arguments should not be necessary - these arguments should never be null. However, it is simple defensive programming, and not costly.

This policy is only interested in requests to edit object property statements, so we can quickly reject any other type of `RequestedAction`. Again, the `INCONCLUSIVE` decision is equivalent to saying "let someone else decide."

This policy does not attempt to restrict the editing of data property statements. This is because the `EditDataPropertyStatement` class does not include the value of the data property. At one time it was felt that this could not affect the decision of whether to permit the request. This will be changed in a future release (See this JIRA issue[144]).

This policy will not restrict the root account from attempting to edit statements.

We already have `RootUserPolicy`, which says that the root user is permitted to do anything. So why do we need this test?

We need to consider the order in which policies are called, and to remember that polling ono a `RequestedAction` will stop when any policy returns a decision that is not `INCONCLUSIVE`. So, if this Policy is placed before `RootUserPolicy`, and returns an `UNAUTHORIZED` decision, then the `RootUserPolicy` will never been consulted.

The question of "what to do when one Policy would authorize and another Policy would prohibit" is a tricky one.

Lines 82-105: Execute the SPARQL query and test the result

```
        EditObjectPropertyStatement stmt = (EditObjectPropertyStatement) whatToAuth;

        String queryString = assembleQueryString(stmt);
        List<String> graphUris = executeQuery(queryString);
        log.debug("graph URIs: " + graphUris);

        if (graphUris.isEmpty()) {
            log.warn("Can't find this statement in any graph: " + stmt);
            return inconclusiveDecision("Can't find this statement in any graph: "
                    + stmt);
        }
```

144 https://jira.duraspace.org/browse/VIVO-268

```
        graphUris.removeAll(PERMITTED_GRAPHS);
        if (graphUris.isEmpty()) {
            log.debug("Permitted: " + stmt);
            return inconclusiveDecision("Statement is only in permitted graphs: "
                    + stmt);
        }

        log.debug("Statement is prohibited: " + stmt + ", graphs=" + graphUris);
        return unauthorizedDecision("Statement is in a prohibited graph, "
                + stmt + " in " + graphUris);
    }
```

Assemble the query and execute it. This results in a list of the URIs of all Graphs that contain this statment. (See the subroutines in the next section).

What to do if we do not find the statement in any graph? It would be possible to err on the side of caution and return an `UNAUTHORIZED` decision. We could even throw a `RuntimeException` of some sort to abort the page display. In this case, we choose to return `INCONCLUSIVE` and write a warning to the log.

If the statement appears only in the permitted graphs, return a decision of `INCONCLUSIVE`, letting some other policy decide.

If the statement appears in other, prohibited graphs, return a decision of `UNAUTHORIZED`, rejecting the requested action.

Lines 106-171: Subroutines

```
    private static final String QUERY_TEMPLATE = "" + //
            "SELECT ?graph WHERE{" + //
            "   GRAPH ?graph{" + //
            "     ?s ?p ?o ." + //
            "   } " + //
            "} LIMIT 10"; //

        private String assembleQueryString(EditObjectPropertyStatement stmt) {
        String q = QUERY_TEMPLATE;
        q = QueryUtils.subUriForQueryVar(q, "s", stmt.getSubjectUri());
        q = QueryUtils.subUriForQueryVar(q, "p", stmt.getPredicateUri());
        q = QueryUtils.subUriForQueryVar(q, "o", stmt.getObjectUri());
        return q;
    }
```

We have a template for the SPARQL query. Substitute the values for this statement into the query. The only unresolved variable will be `?graph`.

```
    private List<String> executeQuery(String queryStr) {
        log.debug("select query is: '" + queryStr + "'");
        QueryExecution qe = null;
        dataset.getLock().enterCriticalSection(Lock.READ);
        try {
            Query query = QueryFactory.create(queryStr, SYNTAX);
            qe = QueryExecutionFactory.create(query, dataset);
            return parseResults(queryStr, qe.execSelect());
        } catch (Exception e) {
            log.error("Failed to execute the Select query: " + queryStr, e);
            return Collections.emptyList();
        } finally {
            if (qe != null) {
                qe.close();
            }
            dataset.getLock().leaveCriticalSection();
        }
    }

    private List<String> parseResults(String queryStr, ResultSet results) {
        List<String> uris = new ArrayList<>();
        if (results.hasNext()) {
            try {
                RDFNode node = results.next().get("graph");
                if ((node != null) && node.isResource()) {
                    uris.add(node.asResource().getURI());
                }
            } catch (Exception e) {
                log.warn("Failed to parse the query result" + queryStr, e);
            }
        }
        return uris;
    }
```

Execute the SPARQL query against the `Dataset`. Extract the graph URIs from the result.

```
    /**
     * An UNAUTHORIZED decision says
     * "Not allowed. Don't bother asking anyone else".
     */
    private PolicyDecision unauthorizedDecision(String message) {
        return new BasicPolicyDecision(Authorization.UNAUTHORIZED, getClass()
                .getSimpleName() + ": " + message);
    }

    /**
     * An INCONCLUSIVE decision says "Let someone else decide".
     */
    private PolicyDecision inconclusiveDecision(String message) {
        return new BasicPolicyDecision(Authorization.INCONCLUSIVE, getClass()
                .getSimpleName() + ": " + message);
    }
```

Convenience methods for creating `PolicyDecision` return values.

8.6.2.3 Setup when VIVO starts

When VIVO starts execution, the `StartupManager` processes the file `startup_listeners.txt`, and instantiating each class that is named in the file, and invoking the `contextsInitialized()` method on each class.

Lines 172-193: The Setup class

```
// ----------------------------------------------------------------------
// Setup class - must be specified in startup_listeners.txt before any
// policy that might be more permissive.
// ----------------------------------------------------------------------
public static class Setup implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext ctx = sce.getServletContext();
        StartupStatus ss = StartupStatus.getBean(ctx);

        RestrictEditingByGraphPolicy p = new RestrictEditingByGraphPolicy(
                ctx);
        ServletPolicyList.addPolicy(ctx, p);
        ss.info(this,
                "Editing object properties is only permitted in these graphs: "
                        + RestrictEditingByGraphPolicy.PERMITTED_GRAPHS);
    }

    @Override
    public void contextDestroyed(ServletContextEvent sce) { /* nothing */
    }
}
```

The Setup class must implement `ServletContextListener`.

On startup, create an instance of the Policy, and add it to the `ServletPolicyList`. Produce an informative message for the startup status screen.

On shutdown, there is nothing to be done. If there were resources to be freed or files to be closed, this would be the place to do it.

Invoking the Setup class

Initialize the policy in startup_listeners.txt

```
edu.cornell.mannlib.vitro.webapp.auth.policy.RestrictEditingByGraphPolicy$Setup
```

Add this line to `startup_listeners.txt`. Consult the note above regarding placement of this Policy relative to the other Policies.

8.6.2.4 A more complicated example

For another example of writing a policy, look at A more elaborate authorization policy (see page 244)

8.6.3 A more elaborate authorization policy

Suppose you want to do something more elaborate than just prohibit access to a page. For example, perhaps you want to have some profiles be accessible only to certain people.

This becomes a more interesting task, because all profiles are presented by the same controller. So how do you tell the controller that a person is authorized to view the page for one profile but not for another?

You must create a `RequestedAction` that takes parameters, and then have your `Policy` use those parameters in its decision.

Another issue is that there are several URLs that will lead to the same profile page. These URLs are equivalent:

Equivalent URLs for the same individual
http://vivo.mydomain.edu/individual/n4796
http://vivo.mydomain.edu/display/n4796
http://vivo.mydomain.edu/individual?uri=http%3A%2F%2Fvivo.mydomain.edu%2Findividual%2Fn4796

The `IndividualController` is also responsible for handling Linked Open Data requests, and again there are a variety of URLs ways to request them. How will you handle all of these URLs that lead to the same page?

8.6.3.1 The RequestedAction

The `RequestedAction` is how the `Controller` asks the `PolicyStack` whether an action is authorized. Each policy may:

- approve the action (`AUTHORIZED`)
- reject the action (`UNAUTHORIZED`)
- let another policy decide (`INCONCLUSIVE`)

If all policies return `INCONCLUSIVE`, the action is rejected.

Most policies are written to check the class of the `RequestedAction`, and to ignore everything they don't understand, like this:

```
if (!(whatToAuth instanceof DisplayDataPropertyStatement)) {
    return new BasicPolicyDecision(Authorization.INCONCLUSIVE, "Unrecognized action");
}
```

The exception to this is RootUserPolicy, which approves every action if the root user is logged in. So,if you create your own class, its likely that only your policy will approve or reject it.

Something to remember: the `Policy` objects do not have access to the current request. So your `RequestedAction` must carry all of the information that the `Policy` will require to make a decision. In this example, the `Policy` needs to know who is logged in, and which profile page they are requesting.

The {{RequestedAction}} class

```
package edu.cornell.mannlib.vitro.webapp.controller.individual;
import edu.cornell.mannlib.vitro.webapp.auth.requestedAction.ifaces.RequestedAction;
import edu.cornell.mannlib.vitro.webapp.beans.Individual;
import edu.cornell.mannlib.vitro.webapp.beans.UserAccount;
/**
 * Ask for authorization to display this individual to this user.
 */
public class DisplayRestrictedIndividualAction extends RequestedAction {
    private final UserAccount user;
    private final Individual individual;
    public DisplayRestrictedIndividualAction(UserAccount user, Individual individual) {
        this.user = user;
        this.individual = individual;
    }
    public UserAccount getUser() {
        return user;
    }
    public Individual getIndividual() {
        return individual;
    }
}
```

8.6.3.2 The Controller

So how does the controller request the action, and what does it do if the action is rejected?

There are a few ways to handle this. If your controller is a sub-class of `FreemarkerHttpServlet`, and if you are willing to accept the default behavior, you can use the `requestedActions()` method. Otherwise, you can use the `FreemarkerHttpServlet.processRequest()` method, or just the `doGet()` method.

Remember, the `IndividualController` needs to deal with several different types of URLs and types of requests. However, it has a method that analyzes the request for you, and creates an `IndividualRequestInfo` object. You can get the information you need from that, as shown in the examples below.

The `requestedActions()` method

This method is a shortcut for subclasses of `FreemarkerHttpServlet`. Just override this method so it returns an `Actions` object. The framework will check to see if the policies approve this requested action. Here is an example.

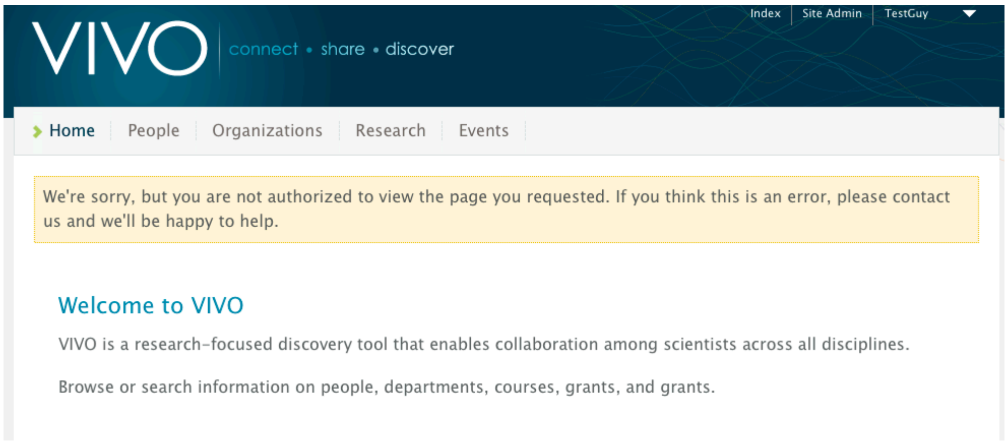
Overriding the {{requestedActions}} method

```
@Override
protected Actions requiredActions(VitroRequest vreq) {
    IndividualRequestInfo requestInfo = analyzeTheRequest(vreq);
    Individual individual = requestInfo.getIndividual();
    UserAccount user = LoginStatusBean.getCurrentUser(vreq);
    return new Actions(new DisplayRestrictedIndividualAction(user, individual));
}
```

What happens if the Policy does not authorize the Action?

If the `PolicyStack` rejects the action, one of two things will happen.

- If the user is not logged in, they will be sent to the login screen. No explanation is offered, but after they log in, the request is repeated.
- If the user is logged in, they will be sent to the home page. A message will appear, like this:



We're sorry, but you are not authorized to view the page you requested. If you think this is an error, please contact us and we'll be happy to help.

Welcome to VIVO

VIVO is a research-focused discovery tool that enables collaboration among scientists across all disciplines.

Browse or search information on people, departments, courses, grants, and grants.

Calling `isAuthorizedToDisplayPage()`

If your controller is not a sub-class of `FreemarkerHttpServlet`, you can accomplish the same result by calling `isAuthorizedToDisplayPage()`. This method takes one or more `RequestedAction` objects, and behaves exactly the same as `requestedActions()` does in a `FreemarkerHttpServlet`.

You must control the code flow yourself, however. If the method returns `false`, your code should immediately return. In that case, the framework has already set the `HttpServletResponse` to redirect as described above.

Simply the code looks like this:

```
public void doGet(HttpServletRequest request, HttpServletResponse response) {
    if (!isAuthorizedToDisplayPage(request, response, new MyRequestedAction())) {
        return;
    }
...
```

If you search the VIVO code base, you will find this pattern in several controller classes.

What happens if the Policy does not authorize the Action?

The result is the same as with the `requiredActions()` method.

For finer control,

In some cases, the default behavior is not wanted. For example, you may want to have your controller display one thing if the action is approved, but display another thing if the action is rejected. In neither case would you want to forward the user to a different page.

In that case, you can call the `isAuthorizedForActions()` method on the `PolicyHelper` class.

```
if (PolicyHelper.isAuthorizedForActions(vreq, new MyRequestedAction())) {
    showAuthorizedResult(request, response);
} else {
    showUnauthorizedResult(request, response);
}
```

What happens if the Policy does not authorize the Action?

That's completely up to you.

8.6.3.3 The Policy

Let's return to the example with the `IndividualController` and the `DisplayRestrictedIndividualAction`. What might the policy look like? Here is a rather silly example. In all likelihood, the actual policy would certainly be more elaborate.

The policy class

```
/* $This file is distributed under the terms of the license in /doc/license.txt$ */

package edu.cornell.mannlib.vitro.webapp.controller.individual;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import edu.cornell.mannlib.vitro.webapp.auth.identifier.IdentifierBundle;
import edu.cornell.mannlib.vitro.webapp.auth.policy.BasicPolicyDecision;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ServletPolicyList;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.Authorization;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.PolicyDecision;
import edu.cornell.mannlib.vitro.webapp.auth.policy.ifaces.PolicyIface;
import edu.cornell.mannlib.vitro.webapp.auth.requestedAction.ifaces.RequestedAction;
import edu.cornell.mannlib.vitro.webapp.beans.Individual;
import edu.cornell.mannlib.vitro.webapp.beans.UserAccount;

public class PermitProfilesPolicy implements PolicyIface {
    private static final Log log = LogFactory
            .getLog(PermitProfilesPolicy.class);
```

```
@Override
public PolicyDecision isAuthorized(IdentifierBundle whoToAuth,
        RequestedAction whatToAuth) {
    if (!(whatToAuth instanceof DisplayRestrictedIndividualAction)) {
        return inconclusiveDecision("Only interested in displaying profiles");
    }
    DisplayRestrictedIndividualAction action = (DisplayRestrictedIndividualAction) whatToAuth;

    UserAccount user = action.getUser();
    Individual individual = action.getIndividual();
    if (user == null) {
        return inconclusiveDecision("User is not logged in.");
    }
    if (individual == null) {
        return inconclusiveDecision("Not on a profile page.");
    }

    return isAuthorized(user, individual);
}

/**
 * This is totally bogus. Presumably you would have more sensible criteria.
 */
private PolicyDecision isAuthorized(UserAccount user, Individual individual) {
    if (individual.getURI().equals(
            "http://vivo.mydomain.edu/individual/n4526")) {
        log.debug("Permit access to " + individual.getLabel());
        return authorizedDecision("I'll let anybody can see this guy.");
    } else {
        log.debug("Deny access to " + individual.getLabel());
        return inconclusiveDecision("Some other policy might approve it, but I won't.");
    }
}

/**
 * An AUTHORIZED decision says "Go ahead. Don't need to ask anyone else".
 */
private PolicyDecision authorizedDecision(String message) {
    return new BasicPolicyDecision(Authorization.AUTHORIZED, getClass()
            .getSimpleName() + ": " + message);
}

/**
 * An INCONCLUSIVE decision says "Let someone else decide".
 */
private PolicyDecision inconclusiveDecision(String message) {
    return new BasicPolicyDecision(Authorization.INCONCLUSIVE, getClass()
            .getSimpleName() + ": " + message);
}

// ----------------------------------------------------------------------
// Setup class
// ----------------------------------------------------------------------
```

```
    public static class Setup implements ServletContextListener {
        @Override
        public void contextInitialized(ServletContextEvent sce) {
            ServletPolicyList.addPolicy(sce.getServletContext(),
                    new PermitProfilesPolicy());
        }

        @Override
        public void contextDestroyed(ServletContextEvent sce) {
            // Nothing to clean up.
        }
    }
}
```

As in the previous example (Creating a VIVO authorization policy - an example (see page 238)), the policy's `Setup` class must be added to `startup_listeners.txt`

8.6.4 The IdentifierBundle - who is requesting authorization?

The policy interface has a single method, and looks like this:

```
public interface PolicyIface  {
    public PolicyDecision isAuthorized(IdentifierBundle whoToAuth, RequestedAction whatToAuth);
}
```

The nature of `whatToAuth` is covered in Creating a VIVO authorization policy - an example (see page 238) and A more elaborate authorization policy. (see page 244) This page is about `whoToAuth`.

8.6.4.1 The challenge of identity and authorization

A user's level of authorization may depend on a variety of information:

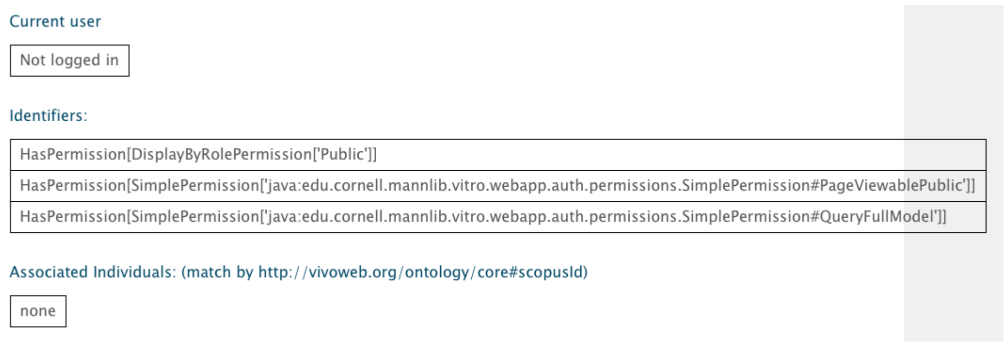
- are they logged in?
- what is their role?
- do they have a profile page?
- what information is in their profile page?
- do they have "proxy authorization" to edit additional pages?

These questions are made more complex because this information is stored in multiple data models. Also, the policy does not have access to the current request or session, so it is not always easy to obtain information.

8.6.4.2 The IdentifierBundle to the rescue

Notice that the `isAuthorized` method receives an argument of the type `IdentifierBundle`. This consists of many `Identifier` objects, and each `Identifier` contains a small piece of information about the current user.

You can see the contents of this bundle (as well as many other things) by directing your browser to `/vivo/admin/showAuth`. This screen shot shows information about an anonymous (not logged in) session:



Current user

Not logged in

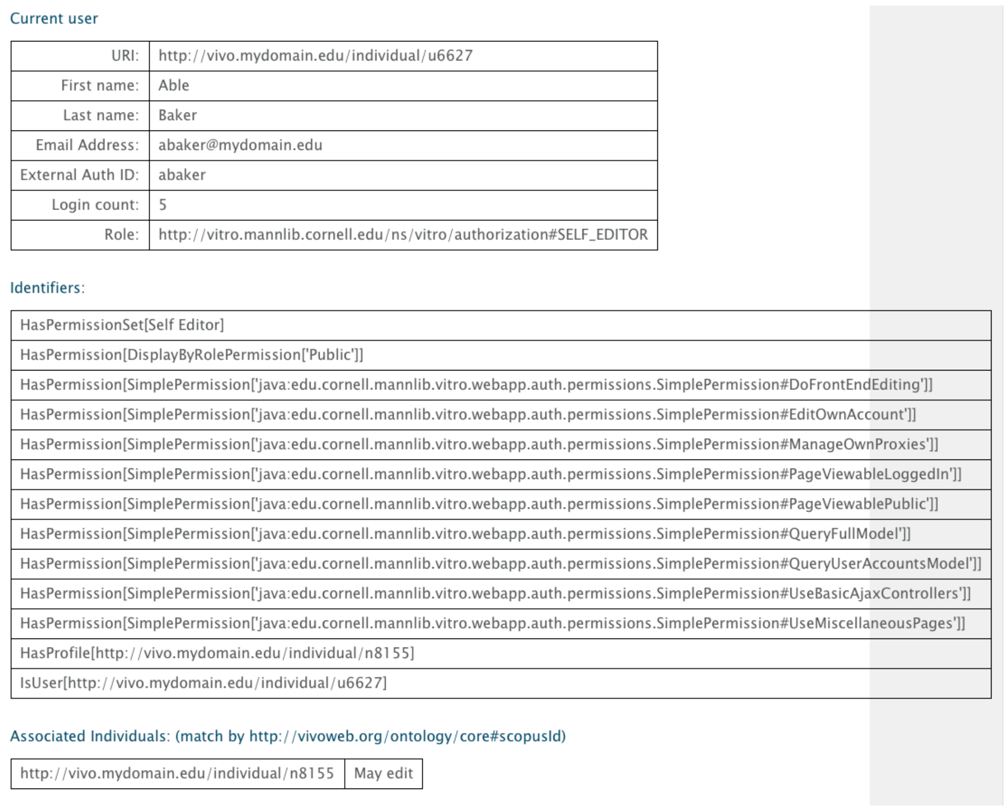
Identifiers:

HasPermission[DisplayByRolePermission['Public']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#PageViewablePublic']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#QueryFullModel']]

Associated Individuals: (match by http://vivoweb.org/ontology/core#scopusId)

none

And this one shows information about a user who is logged in as a self-editor.



Current user

URI:	http://vivo.mydomain.edu/individual/u6627
First name:	Able
Last name:	Baker
Email Address:	abaker@mydomain.edu
External Auth ID:	abaker
Login count:	5
Role:	http://vitro.mannlib.cornell.edu/ns/vitro/authorization#SELF_EDITOR

Identifiers:

HasPermissionSet[Self Editor]
HasPermission[DisplayByRolePermission['Public']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#DoFrontEndEditing']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#EditOwnAccount']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#ManageOwnProxies']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#PageViewableLoggedIn']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#PageViewablePublic']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#QueryFullModel']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#QueryUserAccountsModel']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#UseBasicAjaxControllers']]
HasPermission[SimplePermission['java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#UseMiscellaneousPages']]
HasProfile[http://vivo.mydomain.edu/individual/n8155]
IsUser[http://vivo.mydomain.edu/individual/u6627]

Associated Individuals: (match by http://vivoweb.org/ontology/core#scopusId)

http://vivo.mydomain.edu/individual/n8155	May edit

Your policy has access to these `Identifier` objects, and the `Identifier` classes have static methods that make it easier to find the information you want.

For example, in `edu.cornell.mannlib.vitro.webapp.auth.identifier.common.IsUser`

```
    String userUri = null;
    Collection<String> userUris = IsUser.getUserUris(whoToAuth);
    if (!userUris.isEmpty()) {
        userUri = userUris.iterator().next();
    }
    // null means not logged in.
    // Non-null is the URI of the user account.
```

And, in `edu.cornell.mannlib.vitro.webapp.auth.identifier.common.HasProfile`

```
String profileUri = null;
Collection<String> profileUris = HasProfile.getProfileUris(whoToAuth);
if (!profileUris.isEmpty()) {
    profileUri = profileUris.iterator().next();
}
// null means either not logged in, or no profile.
// Non-null is the URI of the profile page.
```

In most cases, the policy is more interested in the URI of the profile page, rather than the URI of the user account. However, either one might come in handy.

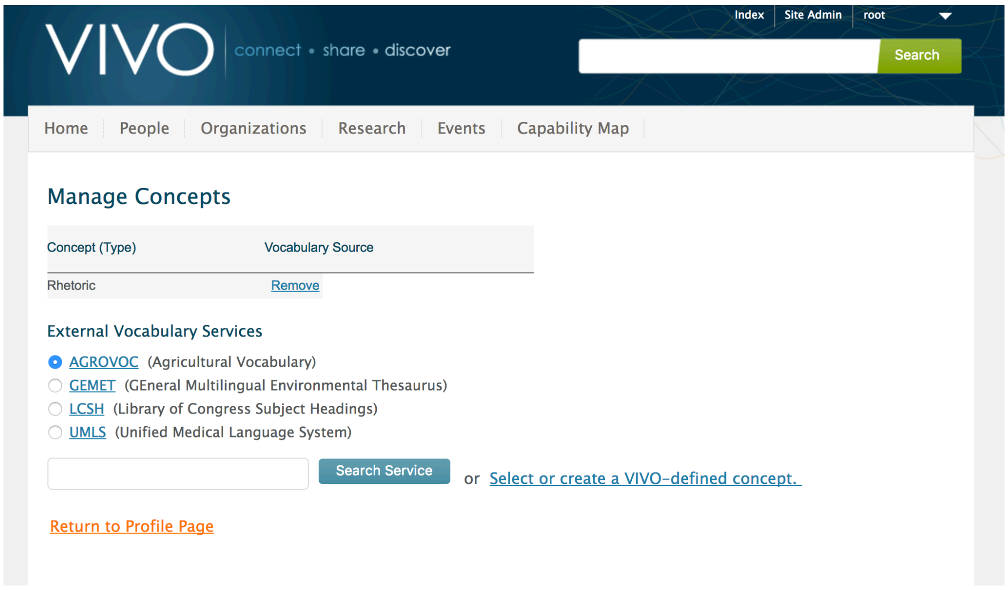
It might be worth noting that `HasProfile` and `HasProxyEditingRights` are both subclasses of `HasAssociatedIndividual`. That means that you can easily distinguish between them, or not, according to the needs of your particular policy.

8.7 Linking to External Vocabularies

8.7.1 Overview

VIVO provides the ability to use external vocabularies to represent the research areas of scholars, and the concepts pertaining to scholarly works. External vocabularies that provide RDF can be used with VIVO. Using an external service, a curator, or page owner may query the external vocabulary for terms, and select terms representing the work or scholar. The terms are fetched from the external service and added to the VIVO triple store. Links from the work or person and added to connect the person or work to the selected term or terms.

See below



Manage Concepts

Concept (Type)	Vocabulary Source
Rhetoric	Remove

External Vocabulary Services

- ● AGROVOC (Agricultural Vocabulary)
- ○ GEMET (GEneral Multilingual Environmental Thesaurus)
- ○ LCSH (Library of Congress Subject Headings)
- ○ UMLS (Unified Medical Language System)

Search Service or Select or create a VIVO-defined concept.

Return to Profile Page

8.7.2 VIVO RDF statements referencing external concepts

When external concepts are added to VIVO, they retain their original URI from the external vocabulary. Since we have no way of knowing whether these URIs represent OWL classes or RDF instance data, VIVO does not assert a type for the concepts, which will therefore only be interpreted as being of type owl:Thing.

subject	**predicate**	**object**
http://vivo.cornell.edu/individual/individual22972	http://vivoweb.org/ontology/core#hasResearchArea	http://link.informatics.stonybrook.edu/umls/CUI/C1518584
http://link.informatics.stonybrook.edu/umls/CUI/C1518584	rdfs:label	ontology
http://vivo.cornell.edu/individual/individual22972	http://vivoweb.org/ontology/core#hasResearchArea	http://link.informatics.stonybrook.edu/umls/CUI/C0036612
http://link.informatics.stonybrook.edu/umls/CUI/C0036612	rdfs:label	semantic
http://vivo.cornell.edu/individual/individual22972	http://vivoweb.org/ontology/core#hasResearchArea	http://www.eionet.europa.eu/gemet/concept/3645
http://www.eionet.europa.eu/gemet/concept/3645	rdfs:label	geographic information system
http://vivo.cornell.edu/individual/individual22972	http://vivoweb.org/ontology/core#hasResearchArea	http://link.informatics.stonybrook.edu/umls/CUI/C0599807
http://link.informatics.stonybrook.edu/umls/CUI/C0599807	rdfs:label	informatics
http://vivo.cornell.edu/individual/individual22972	http://vivoweb.org/ontology/core#hasResearchArea	http://link.informatics.stonybrook.edu/umls/CUI/C0872261
http://link.informatics.stonybrook.edu/umls/CUI/C0872261	rdfs:label	repositor

8.7.3 Adding a new external vocabulary service to VIVO

External vocabulary services are defined in the graph http://vitro.mannlib.cornell.edu/filegraph/abox/vocabularySource.n3. You can explore the contents of this graph by navigating to System Admin / Ingest tools / Manage Jena models. Find vocabularySource.n3 in the list of models. Click Output Model. You will get a file containing the assertions made to define external vocabulary services in your VIVO.

8.8 Search Engine Optimization (SEO)

8.8.1 Overview

VIVO is often used by institutions to highlight and promote the works of their scholars. Promotion works best if the VIVO profile and related pages are easily found by search engines, and considered to be high value by search engines. VIVO provides citation metatags and a site map to help search engines recognize the value of VIVO pages in search results.

VIVO sites can do more to improve boost SEO. See the recommendations from the University of California San Francisco below for additional ideas.

8.8.2 Citation Metatags

Citation meta tags are included on the pages of works. An example from the VIVO sample data is shown below.

Citation metatags regarding a publication

```
<meta tag="citation_author" content="Bogart, Andrew " />
<meta tag="citation_author" content="Roberts, Patricia " />
<meta tag="citation_author" content="Stevens, Emily K" />
<meta tag="citation_date" content="2015" />
<meta tag="citation_journal_title" content="Journal of Political Rhetoric" />
<meta tag="citation_firstpage" content="1" />
<meta tag="citation_lastpage" content="54" />
<meta tag="citation_volume" content="15" />
<meta tag="citation_issue" content="2" />
```

8.8.3 Sitemap

For better indexing and discoverability of your VIVO installation, a sitemap generator is included - only profile pages are included in the sitemap. To see your sitemap, append `/sitemap.xml` to your VIVO URL.

8.8.4 Additional SEO Considerations

The University of California San Francisco has done considerable work on SEO for research networking systems. They have compared VIVO installations to other systems, and provided guidelines for enhancing SEO. We strongly recommend sites implement as many of their recommendations as possible to boost their SEO for their scholars and their works.

- RNS SEO 2016: How 90 research networking sites perform on Google — and what that tells us https://biomed20.ucsf.edu/2016/08/18/rns-seo-2016/
- RNS SEO: How 52 research networking sites perform on Google, and what that tells us https://biomed20.ucsf.edu/2015/08/14/rns-seo/
- SEO for Research Networking: How to boost Profiles/VIVO traffic by an order of magnitude https://biomed20.ucsf.edu/2014/08/25/seo-for-research-networking/

9 System Administration

9.1 Background

VIVO system administration requires experience in operating systems, Java application administration, Tomcat, MySQL (or triple store or database being used as persistent storage), backup process and Internet security. In addition, familiarity with VIVO data representation (ontology, triples, and RDF formats) is recommended.

9.2 Creating and Managing User Accounts

9.2.1 Overview

In VIVO, the basic functions of browsing and searching are open to anyone. However, if a VIVO user wants to view restricted data, or to manage VIVO, he must log in to a User Account.

When a user logs in, he provides his credentials and is associated with a User Account. The credentials are often an Email address and password, but might be different information, depending on how VIVO is configured.

Each User Account has a Role assigned to it. The Role determines how much the user is authorized to do. The lowest Role will permit the user to edit his own profile page. Higher Roles permit editing additional data properties, modifying the ontologies, and administering the VIVO application.

9.2.2 Authentication

9.2.2.1 Internal Authentication

Every VIVO system allows users to log in to an existing User Account by supplying the Email Address and password to the account. Even in an installation that relies on external authentication, there are administrative pages that allow a user to login with Email Address and password.

9.2.2.2 External Authentication

VIVO can be configured to work with an External Authentication system like Shibboleth or CUWebAuth. In that case, the user provides whatever information the External Authentication system requires, and the External Authentication system passes an ID value to VIVO. VIVO recognizes that the user is logged in to the User Account whose "External Authentication ID" field matches that ID.

If a user passes External Authentication, but no User Account matches the ID, VIVO prompts the user to enter his Email Address, First Name, and Last Name, and creates a User Account with that information.

NOTE: To configure VIVO for an External Authentication system, please consult the Installation Guide, and refer to the section entitled ‘Using an External Authentication System with VIVO’. Note also that the value of the property (the designated External Authentication ID field) must be an **exact match** for the username/email of the user.

9.2.2.3 External-Only Accounts

When creating an account, an administrator may indicate that it is for external authentication only. In that case, no password is assigned to the account, since the External Authentication system manages its own passwords or other credentials.

9.2.3 What is a User Account?

Each User Account is identified by the user’s Email address. Each account will have the user’s first name and last name, and a role. The account will have additional information, depending on how it is used.

- External Authentication ID – permits logging in by the External Authentication system.
  NOTE: Two User Accounts may not have the same External Authentication ID
- Password – permits logging in by the Internal Authentication system.
- Matching ID – can be used to associate the User Account with a profile page.

9.2.4 User Roles

In VIVO there are four user roles that can be assigned: administrator, curator, editor, and self-editor. Future releases will allow VIVO administrators to create additional roles. Permissions provided to roles will determine access options available to user accounts within VIVO. It is important to consider what a new user’s role may be, prior to setting up the new account.

Self-Editor -– The self-editor may create data properties, relationships and entities directly associated to his or her profile.

Editor -– The editor may add, delete and modify entities, object properties and data properties.

Curator -– In addition to performing the tasks of the Editor, the Curator may modify the ontologies, class groups, property groups, and edit site information, including the text displayed on the About page and contact email address.

System Administrator –- In addition to the abilities of the Curator, the Administrator may access the menu management, user accounts, and advanced data tools features. The advanced data tools section include the ingest menu, Add/Remove RDF data, RDF export, SPARQL query, and SPARQL query builder privileges.

9.2.5 Profile Pages

Each User Account may be matched with an Individual in the VIVO data model. The display page for that Individual is known as the “profile” for that User Account.

A common use of this feature is matching a profile to each member of the campus community. When a user logs in to VIVO, he is directed to his profile page, and is authorized to edit the information on that page.

It is typical for a university to ingest information into VIVO, including the “network ID” for each member of the campus community. When a user logs in to VIVO using the External Authentication system, the ID from the authenticating system is matched against the “network ID” on the individual, and VIVO matches the User Account to the profile.

It is also possible for an administrator to match a User Account with a profile by editing the User Account.

NOTE: To configure VIVO to match User Accounts with profiles, please consult the Installation Guide, and refer to the section entitled 'Specify Deployment Properties'.

9.2.6 The Root User Account

Each VIVO installation has a special User Account, called the root account. The root account has no Role. Nonetheless, the root account is authorized:

- to see all data elements
- to edit all data elements
- to view any page
- to modify the ontologies

Since the root account can do all of these things, it can be particularly useful and particularly dangerous. It can also give you a distorted view of what your VIVO site looks like. Use the root account to create other User Accounts or to access VIVO in emergencies, and use it with deliberation.

The email address for the root account is specified as part of the VIVO installation process.

NOTE: To configure the root account, please consult the Installation Guide, and refer to the section entitled 'Specify Runtime Properties'.

9.2.7 Managing User Accounts

9.2.7.1 Normal workflow

In normal operation, users will receive an Email message when a VIVO account is created for them, when their password is reset by an administrator, or when the Email address on their User Account is changed. One benefit of this is that the administrator does not need to know the user’s password, and does not need to tell the user his password.

As noted above, when a new account is created, or when an administrator resets the user’s password, the user receives an Email message. The message describes the action that has occurred, and includes a link for the user to click, to set the password on the account.

Note: *User Accounts that are created for External Authentication do not require passwords, so no such link is sent.*

9.2.7.2 Workflow without Email

Email notifications can be disabled by configuring VIVO without a “Reply-To” address. In that case, users are not notified when User Accounts are created or changed.

When creating a new User Account, the administrator must set a password, and must inform the user of the password (unless the account is to be used for External Authentication only). When the user first logs in to the account, he will be prompted to change the password. Resetting the password on an account involves a similar process.

Note: *To disable Email notifications, please consult the Installation Guide, and refer to the section entitled 'Specify Deployment Properties'.*

9.2.7.3 External Authentication

In many VIVO installations, the creation of most User Accounts is simple and routine. A user presents credentials to the External Authentication system, and VIVO creates an account with minimal privilege, prompting the user for name and Email Address. In this case, an administrator may edit such an account to assign a higher Role, if desired.

Alternatively, an administrator may create a User Account, add an External Authentication ID, and assign a high-level Role. When the user log in for the first time, they will already have an account with the desired level of privilege.

9.3 Backup and Restore

There are four components that you will want to backup

1. The VIVO home directory
   a. This holds your site's user accounts and encrypted passwords, along with other authorization and display settings.
   b. Holds the Solr search index. The search index is not vital to a backup, since it can be rebuilt. However, rebuilding the index is time-consuming
   c. Also holds any uploaded image files, and any customized RDF files.
   d. Holds your `runtime.properties` file.
2. The VIVO relational database

    a. This holds all of your instance data (people, organizations, etc), as well as any customizations that you entered through the GUI.
3. The VIVO RDF store
    a. In most cases, the VIVO RDF store is held in the VIVO relational database (above), but at some sites it might be in a separate triple-store.
4. The VIVO installation directory
    a. If you have customized the templates or the Java code, you will want to preserve those changes.
    b. At a minimum, this directory contains your `build.properties` file.

9.4 Inferences and Indexing

9.4.1 Recompute Inferences

The inference engine / Reasoner may need to be told to run, and that is achieved by a user with administrative privileges visiting a job specific site.

```
http://vivo.mydomain.edu/RecomputeInferences
```

9.4.2 Re-building the search index

The Solr search may need to have its index re built, and that is achieved by a user with administrative privileges visiting a job specific site.

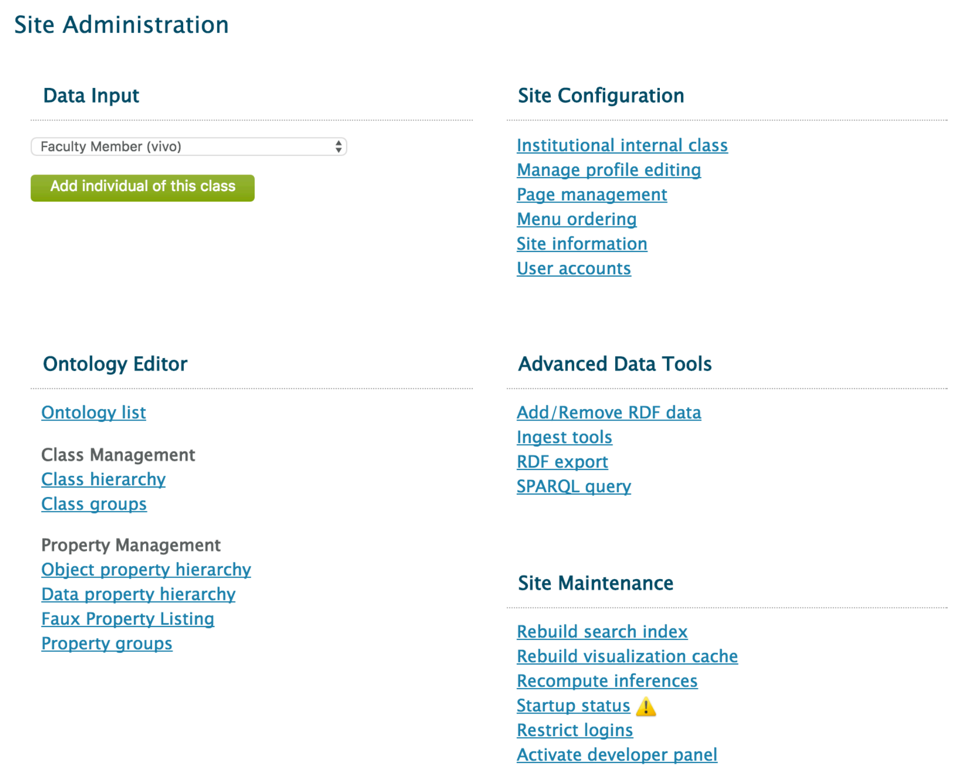
```
http://vivo.mydomain.edu/SearchIndex
```

9.5 The Site Administration Page

9.5.1 Site Administration

Once you are logged into VIVO, you will notice in the upper right hand portion of the page links to "Index" and "Site Admin", alongside a drop-down menu with your name on it, and containing links to "My account" and "Log out".

Once you have logged into VIVO, clicking on the “Site Admin” link takes you to the “Site Administration” page. As an administrator, you will be able to access all five feature and content areas of VIVO: Data Input, Ontology Editor, Site Configuration, Advanced Data Tools, and Site Maintenance. Each is introduced below.



9.5.2 Data Input

There are three ways to manually input data into VIVO. 1) on the Site Administration Menu, a new individual of any class may be added directly through the Data Input menu. 2) Selections can be made on many of the pages to add individuals. For example, on user profile pages, users with editing privileges can add, change and remove data. 3) Data can be added as a batch, a collection of RDF triples in a format known to VIVO and expressed in the ontologies known to VIVO.

On the Site Administration page, you can enter a new individual of any type by selecting the type from the drop down menu, and pressing "Add Individual of this class." VIVO will add an individual of the class you have selected, creating a new URI for the individual, and creating an assertion that the individual is a member of the class you have selected. Once an individual has been created, object and data properties may be added for that individual on the page displaying the individual’s profile. The object and data properties presented for editing will vary by the type of the individual, in accordance with the ontology;

9.5.3 Ontology Editor

In VIVO, information is identified by references to Unique Resource Identifiers (URIs). URIs can be used by other web pages and applications to locate and retrieve specific chunks of data. The detailed level to which VIVO captures information enables complex relationships among data to be represented.

The VIVO web application is built using RDF "triples" or statements consisting of a subject (known as an individual, item, or entity), a predicate (an object property or a data property) and an object (any individual in VIVO). Subject-predicate-object triples express the relationships among the individuals in VIVO via object properties and support attributes of individuals via data properties.

The first two parts (subject and predicate) of every triple are URIs. An object property triple has the URI of another individual in VIVO its object, while the third element of a data property triple is a data value – typically a text string, number, or date.

Ontology List - VIVO supports keeping an internal list of ontology namespaces and corresponding prefixes to facilitate using external ontologies as well as to help differentiate local ontology additions from VIVO core.

9.5.3.1 Class Management

Individuals in VIVO are typed as members of one or more classes organized and displayed as a hierarchy.

Class hierarchy - The class hierarchy provides a framework to help identify the different types of individuals modeled in a VIVO application. In the Class Hierarchy page, you can edit/add classes, add entities to a class, and add auto links.

Class groups - Class groups are a VIVO-specific extension to support using VIVO as a public website as well as an ontology and content editor. Class groups are a means to organize the classes in VIVO into groups. They represent the facets seen when VIVO is searched (people, activities, events, organizations, etc).

9.5.3.2 Property Management

If classes define what each individual in VIVO is, properties define how that individual relates to other individuals and allow an individual to have attributes of its own. VIVO has two property editors, one for object properties and another for data properties.

Object property hierarchy - Object properties represent the relationship between entities (also known as items or individuals) in VIVO. Object properties can be created and edited from the Object Property Hierarchy.

Data property hierarchy - A data property connects a single subject individual (e.g., a Person or Event) with some form of attribute data. Data properties can be created and edited from the Data property hierarchy link.

Faux property listing - Faux properties are a VIVO-specific extension to allow the same object property to be used in various contexts, with a context specific label and context specific domain and range. A listing of the faux properties in VIVO. You can display the list alphabetically, or organized by base property. The Site Administration page provides a simple view-only listing. See Create and edit faux properties (see page 143) to manage faux properties.

Property groups - Like class groups, property groups are a VIVO-specific extension to support using VIVO as a public website as well as an ontology and content editor.

9.5.4 Site Configuration

This section discusses the site configuration aspects of VIVO. It enables administrators to add or adjust to their institution's site specific details, as well as to manage menus, tabs, and user accounts.

Institutional internal class – set the class that will be used to indicate that individuals are part of your institution. See Create, Assign, and Use an Institutional Internal Class (see page 59)

Manage profile editing – Assign profile editors to individual profiles. Use this feature to allow someone other than the profile owner to edit the owner's profile.

Page management – Create and manage custom pages, as well the presence of pages on menus. See Menu and page management (see page 136)

Menu Ordering – order the menus on the main VIVO navigation banner. See Menu and page management (see page 136)

9.5.4.1 Site Information

The Site information link provides administrators with the capabilities of editing and adding site specific details for that institution's instance of VIVO.

Site Name — Text entered here will be displayed in the browser title bar and bookmark label. It is set to "VIVO" by default.

Contact email address — This field is the email address or listserv that you want the Contact Us form to use. The SMTP host in your configuration file (runtime.properties) must be set for the Contact Us form to work as intended.

Theme — The default theme is "wilma". If you create a new theme (see Creating a custom theme (see page 181)), then it should be available to choose in this drop-down pick list.

Copyright text — Text entered here for a label in the footer for the copyright URL.

Copyright URL — The URL you want the copyright to go to in the footer. It could be your institution's copyright information or the actual institution.

9.5.5 Advanced Tools

The Advanced tools are VIVO's built-in features for data management and export. Please refer to the Advanced Tools section below for detailed instructions.

In addition, many VIVO adopters may require additional information regarding the importing and exporting of RDF data and creating SPARQL queries.

There are several avenues available to acquire guidance with these advanced tools. Information sources such as the VIVO Data Ingest Guide, the W3C's Resource Description Framework model, and the W3C's SPARQL Query Language for RDF, to name a few. Please refer to Appendix A for links and additional resources.

Add/Remove RDF data – This tool allows for the manipulation of RDF data in the main model through importing RDF documents for addition or removal.

Ingest tools – A suite of data management tools. See below for a description of each tool.

RDF export - This tool allows for the export of ontology and data in a variety of RDF formats. Options include:

- Export all instance data
- Export a specific ontology such as FOAF, VIVO core, SKOS, etc.
- Export the entire ontology for VIVO

SPARQL query - This tool allows SPARQL select, construct, and describe statements against the main model to be saved in a variety of formats including: CSV, RDF/XML, N3 and more.

9.5.5.1 Ingest tools

Manage Jena Models – This tool allows for the management of the main webapp, as well as separate data models and datasets. The ability to attach separate models to the webapp, load RDF data to a mode, clear statements, and output models as N3 RDF is performed here.

Subtract One Model from Another — This tool allows for the comparison of models for updating information that already exists in VIVO. By subtraction of a current model from a newly constructed model (from the same data source) and vice versa, the additions and subtractions for updating the data are generated.

Convert CSV to RDF — This tool allows for VIVO to read and convert CSV (comma-separated values) and Tab-delimited data into RDF

Convert XML to RDF — This tool allows for VIVO to read and convert well-formed XML into RDF

Execute SPARQL CONSTRUCT — This tool allows for using SPARQL to produce desired RDF from one or multiple source models. This tool is commonly used to map classes and properties to VIVO namespace(s).

Generate Tbox — Tbox statements describe the terms of controlled vocabularies, for example, a set of classes and properties that constitute the ontology. This tool allows for the creation of a Tbox from one or multiple source models.

Name Blank Nodes — This action turns blank nodes, a node in an RDF graph which is not identified by a URI and is not a literal, into nodes with either randomly generated or pattern based URIs.

Smush Resources — This tool allows for using a compression method to distinguish like entities and "Smush" them together based on the specified URI of a property.

Merge Resources — This tool allows two individuals with different URIs to be collapsed into a single URI. Any statements using the "duplicate individual URI" will be rewritten using the "primary individual URI." If there are multiple statements for a property that can have only a single value, the extra statements will be retracted from the model and offered for download.

Process Property Value Strings — This tool allows for an arbitrary method on a Java class available on the application class path to transform string values of a given property. The method should take a single String as a parameter and return a String.

Change Namespace of Resources — This tool will change all resources in the supplied "old namespace" to be in the "new namespace." Additionally, the local names will be updated to follow the established "n" + random integer naming convention.

Split Property Value Strings into Multiple Property Values — This tool allows for parsing multiple property values from a single ingested string. This can be used to parse MeSH Terms, controlled vocabulary, and keywords associated with the ingested data.

Execute Workflow — This tool allows for a simple way of scripting actions (specified in RDF) that would otherwise require manual interaction with the ingest tools.

Dump or restore the knowledge base – dump or restore configuration models or content models

9.5.6 Site Maintenance

Rebuild search index – in some situations, you may need to rebuild the SOLR search index. See Inferences and Indexing (see page 260)

Rebuild visualization cache – Large-scale visualizations like the Temporal Graph or the Map of Science involve calculating total counts of publications or of grants for some entity. Since this means checking also through all of its sub-entities, the underlying queries can be both memory-intensive and time-consuming. For a faster user experience, we wish to save the results of these queries for later re-use. To this end we have devised a caching solution which will retain information about the hierarchy of organizations-namely, which publications are attributed to which organizations-by storing the RDF model. We're currently caching these models in memory. The cache is built (only once) on the first user request after a server restart. Because of this, the same model will be served until the next restart. This means that the data in these models may become stale depending upon when it was last created. To avoid restarting the server in order to refresh the cache, administrators can use the Rebuild visualization cache link.

Recompute inferences – in some cases, you may wish to recompute the inferences in VIVO. See Inferences and Indexing (see page 260)

Startup status – shows the messages that were produced during VIVO startup.

Restrict logins – toggles user login. When logins are restricted, only the root user may login

Activate Developer panel – Shows the developer panel from which additional debugging information is available. See Tips for Interface Developers (see page 226)

9.6 The VIVO log file

The VIVO log file contains time-stamped statements intended to help you

- identify the configuration of VIVO,
- monitor the progress of the application, and
- diagnose problems that occur.

The log file is written to the `logs` directory of your Tomcat application. It is usually called `vivo.all.log`, but the name may vary, depending on how your VIVO was installed.

The log file can also be helpful during development and debugging. This is particularly true if the developer takes advantage of the different logging levels.

9.6.1 What does a log message look like?

Here is an example of some code that writes to the log

```
private static final Log log = LogFactory.getLog(WebappDaoSetup.class);
...
log.info(elapsedSeconds + " seconds to set up models and DAO factories");
```

and here is the resulting line in the log:

```
2012-11-15 12:20:37,406 INFO [<span class="confluence-link">WebappDaoSetup</span>] 3 seconds to set up
models and DAO factories
```

The log holds the time that the statement was written, the severity level of the message,
the name of the Java class that wrote the statement, and the contents of the statement itself.

Writing exceptions to the log can be tricky: check out this page on Writing Exceptions to the Log (see page 269)

9.6.2 What is the right level for a log message?

Each log message has an output level (sometimes known as a severity level).
The most common levels are DEBUG, INFO, WARN, ERROR.
Each level conveys a sense of how important the message is.

ERROR	Serious errors which need to be addressed and may result in unstable state.
WARN	Runtime situations that are undesirable or unexpected, but not necessarily "wrong", especially if the system can compensate; "almost" errors.
INFO	Interesting runtime events; routine monitoring information. Commonly used to describe how the system starts up, or changes that are worth noting as the system runs.
DEBUG	Used by developers when debugging their code. These messages will not appear in the log unless specifically enabled (see below)

The logging framework also supports the levels of `FATAL` for very serious errors,
and `TRACE` for verbose debugging messages, but these are much less commonly used.

9.6.3 Setting the output levels

9.6.3.1 Production settings

The output levels for VIVO are determined by a file called `[vitro-core]/webapp/config/log4j.properties`

This file sets the general output level to `INFO`, which means that messages at the `INFO` level or higher will be written to the log. Messages at `DEBUG` or lower will not be written to the log.

The file also sets higher output levels for some classes that are otherwise too chatty with their log messages. So for example, the `StartupStatus` class is assigned an output level of `WARN`. This means that messages at the `WARN` level or higher will be written to the log, and messages at `INFO` or lower will not.

9.6.3.2 Developer settings

Developers can make temporary changes to these settings by creating a file called `[vitro-core]/webapp/config/debug.log4j.properties`

When VIVO is rebuilt, the settings in this file will be used instead of the settings in the default file. A developer will commonly change the output level of the classes or packages he is currently working on, using this file.

The debug settings file is ignored by Git. As a result it remains unique to the individual developer, and can be changed without concern.

The debug settings file should not be present in a VIVO that is being built for production use.

9.6.3.3 Changing levels while VIVO is running

You can change the log levels for individual Java classes while VIVO is running.

Direct your browser to `[vivo]/admin/log4j.jsp` This page requires that you log in to VIVO as an administrator.

This page shows a list of all Java classes with active Logger components. Each class has a drop-down list that allows you to set the log output level for that class. Select the level(s) you want, and scroll to the bottom of the page to click the button labeled `Submit changes to logging levels`. The change is effective immediately.

This feature should be used with care. A log level of `DEBUG` can significantly slow down some Java classes, and can result in very large amounts of output to the log of a busy system.

*Note: The `log4j.jsp` page shows only the classes with **active** Loggers. This means that you can't set use this page to set the output level of a class prior to the first time it is used. Java loads classes dynamically, and until the class is loaded, it does not have an active Logger.*

9.6.4 Customizing the logging configuration

9.6.4.1

9.6.4.2 Overview

VIVO uses the Log4J package for logging status messages. VIVO is shipped with a configuration file that sets up the logging properties, so the VIVO log is written to `vivo.all.log` in the *`[tomcat]`*`/logs` directory. Most sites find this default configuration suitable when they start out, but often as people become more experienced with VIVO, they prefer to change the logging options.

9.6.4.3 The default configuration

The configuration file is found at *`[vitro]`*`/webapp/config/log4j.properties`. The file looks something like this:

```
log4j.appender.AllAppender=org.apache.log4j.RollingFileAppender
log4j.appender.AllAppender.File=$${catalina.home}/logs/${webapp.name}.all.log
log4j.appender.AllAppender.MaxFileSize=10MB
```

```
log4j.appender.AllAppender.MaxBackupIndex=10
log4j.appender.AllAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.AllAppender.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{1}] %m%n

log4j.rootLogger=INFO, AllAppender

log4j.logger.edu.cornell.mannlib.vitro.webapp.startup.StartupStatus=WARN
log4j.logger.edu.cornell.mannlib.vitro.webapp.dao.jena.pellet.PelletListener=WARN
log4j.logger.org.springframework=WARN
log4j.logger.com.hp.hpl.jena.sdb.sql.SDBConnection=ERROR
```

(The listing above has been abridged for clarity. Comments have been removed, as have some repetitious lines.)

The file creates an "appender", which tells Log4J where to write the log messages, and how to manage them. It creates a "root logger" which will set the default properties for all logging: using the named appender and omitting any messages that are lower than INFO level. Finally, it overrides the logging threshold level for some special classes and packages.

In more detail (by line numbers):

(1) Use a RollingFileAppender. This will write messages to the named file, until the file becomes too large. Then the accumulated messages are "rolled over" to a backup file, and logging continues.

(2) Specify the name and location of the log file. During the build process, `${webapp.name}` will be replaced by `vivo`, or whatever you have chosen as the name of your webapp. When VIVO starts, Log4J will replace `$${catalina.home}` with the value of the system property named `catalina.home`. This is the Tomcat home directory.

(3) Files will roll over when they reach 10 MegaBytes of content.

(4) No more than 10 files will be kept

(5, 6) The message layout is determined by this pattern. It consists of the date and time, the severity of the message, the name of the class writing the message, and the message itself (followed by a linefeed).

(8) The root logger, and by default all loggers, will write to this appender. Only messages with a level of INFO or higher will be written to the log. That is, messages with levels of DEBUG or TRACE will not be written.

(10, 11) Override the defaults for these classes. The write too many INFO messages, so we restrict them to WARN or higher.

(12) Override the default for the entire package of `org.springframework`

(13) Don't show messages from `com.hp.hpl.jena.sdb.sql.SDBConnection` unless they are ERROR or FATAL.

9.6.4.4 Writing some messages to a special log

Here is an example of how to override the defaults for particular classes in VIVO. In this example, the messages associated with rebuilding the search index are to be written to a special log file. The messages about re-inferencing are also to be written to that file.

The lines below can be added to the end of the default configuration:

```
log4j.appender.SpecialAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.SpecialAppender.DatePattern='.'yyyy-MM-dd
log4j.appender.SpecialAppender.File=/usr/local/vivo/logs/inference_and_indexing.log
```

```
log4j.appender.SpecialAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.SpecialAppender.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{1}] %m%n
log4j.logger.edu.cornell.mannlib.vitro.webapp.search.indexing.IndexBuilder=SpecialAppender
log4j.logger.edu.cornell.mannlib.vitro.webapp.search.indexing.IndexWorkerThread=SpecialAppender
log4j.logger.edu.cornell.mannlib.vitro.webapp.reasoner.ABoxRecomputer=SpecialAppender
```

Here we define a second appender, and tell three particular Java classes to use that appender.

Again, by line numbers:

> (15, 16) Use a DailyRollingFileAppender. Unlike the RollingFileAppender, this log file will roll over at midnight every day. There is no maximum number of files.
>
> (17) The log file will be `/usr/local/vivo/logs/inference_and_indexing.log`. At midnight, the file will be renamed to `inference_and_indexing_log.2013-06-21` (for example).
>
> (18, 19) The layout of the message is the same as for the main log file
>
> (20, 21, 22) These three classes will write to the new appender.

Notice that the log messages for these classes will now be written both to the main log file and to this special file. By default, the appenders are "added" to the classes where they are specified. If you want these classes to only write to the special file, you must turn off the "additivity" property of those classes, as shown below:

```
log4j.additivity.edu.cornell.mannlib.vitro.webapp.search.indexing.IndexBuilder=false
log4j.additivity.edu.cornell.mannlib.vitro.webapp.search.indexing.IndexWorkerThread=false
log4j.additivity.edu.cornell.mannlib.vitro.webapp.reasoner.ABoxRecomputer=false
```

9.6.4.5 More information

Log4J is a very powerful and flexible framework. Many different options are available through the use of appenders, layouts, and filters. For more information, you may want to consult

- The Log4J manual[145] – a compact discussion of the many aspects of Log4J.
- The Log4J API documentation[146]
- The documentation of the Log4j properties file[147]

9.6.5 Writing Exceptions to the Log

9.6.5.1

145 http://logging.apache.org/log4j/1.2/manual.html
146 http://logging.apache.org/log4j/1.2/apidocs/
147 http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PropertyConfigurator.html

9.6.5.2 Not the Right Way

This is not a good way to handle an exception:

```
} catch(Exception e) {
}
```

An exception occurred, but we ignored it. Don't do this. Please.

This isn't very good either (although, to be fair, it is better than a kick in the head):

```
} catch(Exception e) {
  e.printStackTrace();
}
```

In Vivo/Vitro the stack trace is printed to catalina.out instead of vivo.all.log. In the Vivo Harvester it is printed to standard out (System.out). It has no timestamp and no source information, so we can't correlate it with other messages in the log. Were any other messages produced by the same request? We'll never know.

9.6.5.3 Declaring a Logger

In Vivo and Vitro, we use Apache Commons Logging. Create a logger in your Java code with a couple of imports and a static variable:

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class MyClass {
    private static final Log log = LogFactory.getLog(MyClass.class);
    ...
```

In the Vivo Harvester, we use Simple Logging Facade 4 Java. Create a logger in your Java code much like ACL:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MyClass {
    private static Logger log = LoggerFactory.getLogger(MyClass.class);
    ...
```

9.6.5.4 Bad, Better, Good

So, if this isn't good, how can we improve on it?

```
} catch(Exception e) {
}
```

This is better. We're still ignoring it, but we could stop ignoring it just by raising the logging level:

```
} catch(Exception e) {
  log.debug(e, e);
}
```

This is better still. Here is a clue as to why we're ignoring the exception.

```
} catch(Exception e) {
  // This happens if the model data is bad - it's not important
  log.debug(e, e);
}
```

What if we do want to write the exception to the log? What's the right way to do it?

Not like this, for reasons mentioned earlier:

```
} catch(Exception e) {
  e.printStackTrace();
}
```

This is better:

```
} catch(Exception e) {
  log.error(e, e);
}
```

If you have an idea of why a certain exception might be occurring, this would be the best:

```
} catch(IllegalStateException e) {
  log.error("One of the flay-rods has gone out of skew.", e);
} catch(Exception e) {
  log.error(e, e);
}
```

But alas, sometimes no useful message occurs to us.

9.6.5.5 Whoops

Unlike some other logging frameworks (Log4J, for example) Apache Commons Logging won't check to see whether your first argument is an exception. Instead, it just converts it to a String and prints it to the log.

So, this probably doesn't do what you wanted:

```
} catch(Exception e) {
  log.error(e);
}
```

It logs the class of the exception, and the message in the exception, but it doesn't write the stack trace. That's why this is better:

```
} catch(Exception e) {
  log.error(e, e);
}
```

This way, the Exception class and it's message are written to the log twice, but that's a small price to pay – at least you get the stack trace in the log as well.

And this is best:

```
} catch(ExpectedTypeAException e) {
  log.error("Some informative message explaining why TypeA might occur", e);
} catch(ExpectedTypeBException e) {
  log.error("Some informative message explaining why TypeB might occur", e);
} catch(Exception e) {
  log.error("Some informative message explaining that an unexpected error occurred", e);
}
```

Because you get to provide more information, you don't write anything twice, and you do get the stack trace.

9.7 Activating the ORCID integration

9.7.1 Overview

VIVO contains code that will converse with the ORCID registry through its API. When this conversation is enabled, a VIVO user can authoritatively confirm his ORCID iD in VIVO, and cite his VIVO page in his ORCID record as an external identifier.

In order to activate the VIVO-ORCID integration, your organization must have a membership in ORCID. You may then register your VIVO installation as a client application, and obtain the credentials needed for that connection.

Once you have the credentials, you can enter them in the runtime.properties file and restart VIVO.

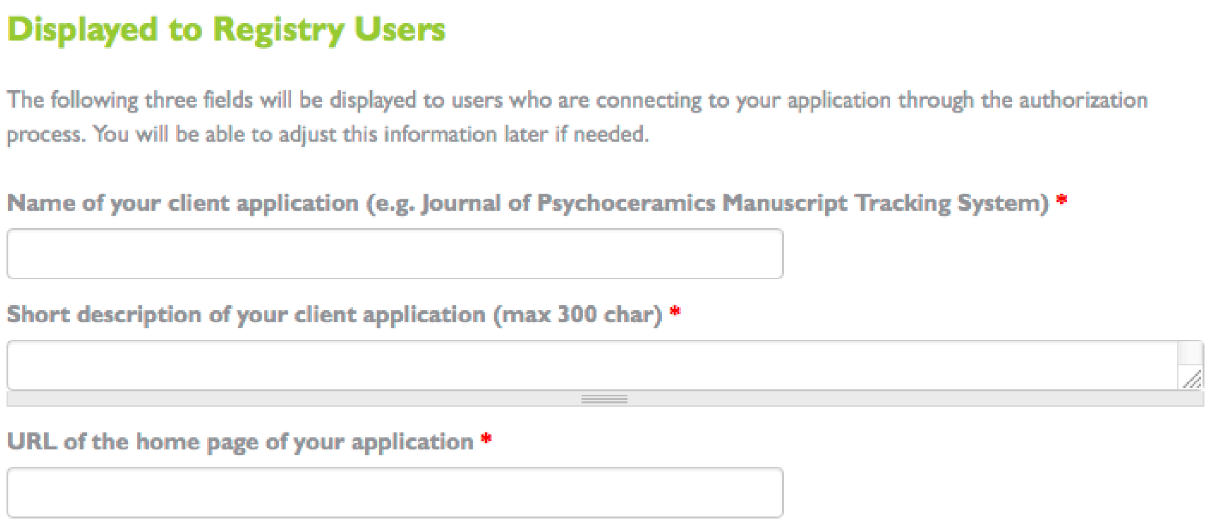
You may want to start by obtaining credentials for ORCID's sandbox API. This will let you see how the integration appears. If you have made local modifications to VIVO, you will want to ensure that they do not interfere with the integration before going into production.

Once you are satisfied that the integration is working as expected, you can apply for credentials on ORCID's production registry.

9.7.2 When applying for credentials

9.7.2.1 Informing the users

The user must grant authorization before VIVO can read or write to their ORCID record. Some of the text they see will come from your credentials. Notice this section of the application:



Displayed to Registry Users

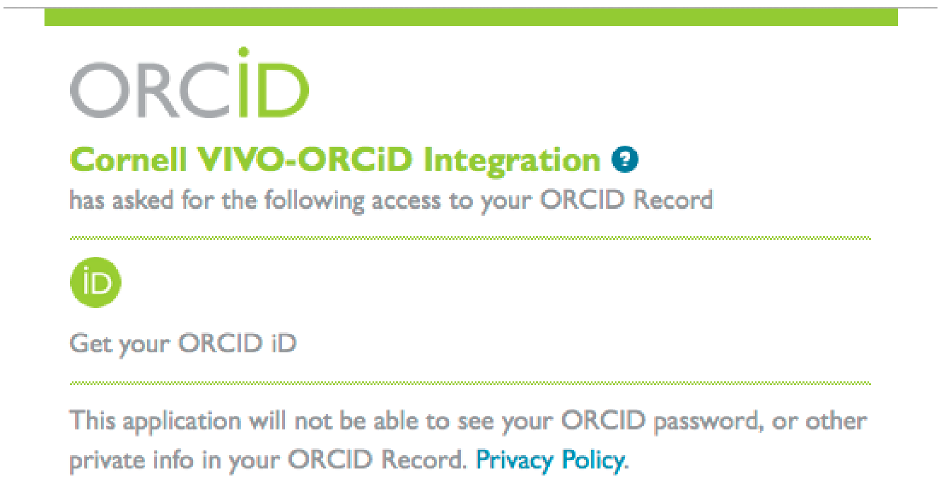
The following three fields will be displayed to users who are connecting to your application through the authorization process. You will be able to adjust this information later if needed.

Name of your client application (e.g. Journal of Psychoceramics Manuscript Tracking System) *

Short description of your client application (max 300 char) *

URL of the home page of your application *

The name of your client application will be displayed to the user as they use the integration screens. Here is an example, where the name of the client application is "Cornell VIVO-ORCID Integration".


ORCiD

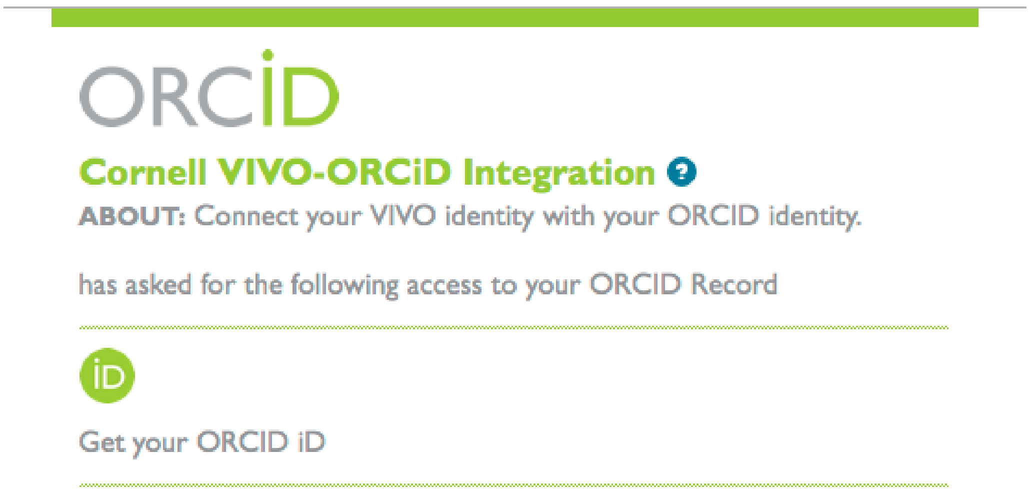
Cornell VIVO-ORCiD Integration

has asked for the following access to your ORCID Record

Get your ORCID iD

This application will not be able to see your ORCID password, or other private info in your ORCID Record. Privacy Policy.

If the user clicks on the question mark, they will see the short description of your client application. In this example, the short description is "Connect your VIVO identity with your ORCID identity."


ORCiD

Cornell VIVO-ORCiD Integration

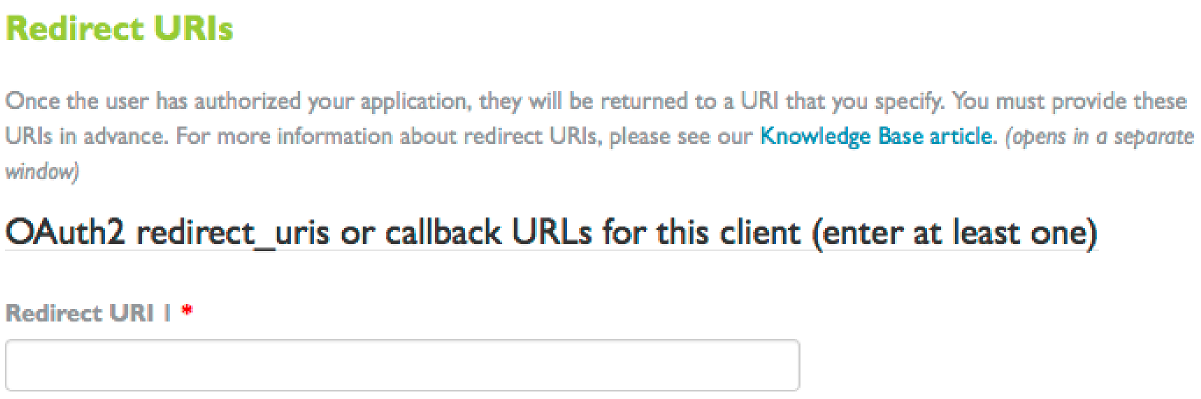
ABOUT: Connect your VIVO identity with your ORCID identity.

has asked for the following access to your ORCID Record

Get your ORCID iD

9.7.2.2 Connecting to your application

Once the user logs in to their ORCID account, and grants authorization to your application, the ORCID pages will transfer control of the session back to VIVO. In order to do that, it needs to know where your application is located. Notice this section of the application:


Redirect URIs

Once the user has authorized your application, they will be returned to a URI that you specify. You must provide these URIs in advance. For more information about redirect URIs, please see our Knowledge Base article. *(opens in a separate window)*

OAuth2 redirect_uris or callback URLs for this client (enter at least one)

Redirect URI I *

You may provide just the domain of your application, such as `http://vivo.mydomain.edu`.

9.7.3 Configuring VIVO

To converse with ORCID, VIVO requires these values in the `runtime.properties` file.

Property name	**`orcid.clientId`**
Description	**The Client ID from your ORCID credentials** When your application for credentials is accepted, you will receive a Client ID to be used in communications with the API. If you apply for sandbox credentials first, and then production credentials, you will likely receive two different Client IDs.
Default value	NONE
Example value	`0000-0012-0661-9330`

Property name	**`orcid.clientPassword`**
Description	**The Client Secret from your ORCID credentials** When your application for credentials is accepted, you will receive a Client Secret to be used in communications with the API. If you apply for sandbox credentials first, and then production credentials, you will likely receive two different Client Secrets.
Default value	NONE
Example value	`103de999-1a37-400c-309f-2094ba72c988`

Property name	**`orcid.webappBaseUrl`**
Description	**The base URL for your VIVO application, as seen from outside.** VIVO will use this to construct a callback URL that the ORCID API can use to return control to VIVO. The actual callback URL will be the string you provide here with the suffix of `/orcid/callback` added at the end.
Default value	NONE
Example value	`http://vivo.mydomain.edu` `http://some.domain.edu/vivo/`

Property name	**`orcid.apiVersion`**
Description	**The version of ORCIDs API protocol that VIVO will expect.** Versions `1.0.23, 1.2, or 2.0`
Default value	NONE
Example value	`2.0`

Property name	**`orcid.externalIdCommonName`**
Description	**The label used to describe a VIVO profile page** If the user authorizes the addition of their VIVO profile page to their ORCID record, it will appear as an "external ID", with this label
Default value	NONE
Example value	`VIVO profile page at Great Western University`

Property name	**`orcid.api`**
Description	**The entry point for ORCID's public API.** This changes, depending on whether you are using the sandbox API or the production API.
Default value	NONE

Example value	`sandbox` `release`

9.8 Performance Tuning

9.8.1 SDB - MySQL Tuning

By default, MySQL has reasonable defaults for a regular RDBMS application. However, SDB has a slightly unusual database layout - it has very few tables, some of which grow quite large, very quickly. Whilst the SDB code is well optimised for the majority of cases, to get the best performance, you should tune MySQL to take into account the table, index and join sizes.

9.8.1.1 Version Recommendation

It is recommended that you use 5.5 or later of MySQL (or the MariaDB equivalent).

9.8.1.2 MySQL DB Engine

It is recommended that you use innodb with the barracuda file format. You should also configure MySQL to use a file for each table.

```
innodb_file_per_table = 1
innodb_file_format = barracuda
```

9.8.1.3 MySQL Buffers

Although this won't affect an initial query, having large buffers for the indexes will help query performance once they have been warmed.

```
join_buffer_size = 32M
read_rd_buffer_size = 32M
innodb_buffer_pool_size = 1536M
```

9.8.1.4 Temporary Tables

SDB can generate some large joins, and by default anything over 16MB will be spooled to disk. This can slow large queries down dramatically. To avoid this, increase the temporary table sizes.

```
max_heap_table_size=256M
tmp_table_size=256M
```

9.8.2 Additional Performance Tips

9.8.2.1 What is performance?

Performance can mean different things to different sites including the length of time it takes to render a large page (e.g., a person with 800 - 1500 publications), to display a visualization, to load new data, to regenerate the search index or recompute inferences, or to generate an export of RDF data.

9.8.2.2 What kind of performance is normal? How do I know if I have a problem?

This section gives some very rough guidelines for determining whether your VIVO is performing similarly to established production installations on typical modern server hardware or virtual machines. The numbers below assume that VIVO is otherwise idle; that is, not loaded with concurrent public page requests or performing other background operations.

Individual page display

The time it takes to render an individual page can vary significantly depending on the types of data involved. The page for a person with many publication citations will take longer to render than one with simple links to other individuals. As a very general rule, your VIVO should be able to handle around 100 data items (properties) per second when displaying an individual page. Thus, if the page for a person with 500 publication links displays in five seconds, there may be relatively little room for performance tweaking short of caching the entire page. If the page takes 50 seconds to appear, there is very likely a serious performance bottleneck somewhere in the installation or a hardware deficiency that needs to be addressed.

RDF loading

Loading RDF through VIVO is slower than inserting it directly into the triple store because VIVO performs additional operations such as inference and search index maintenance as the data are changed. You should still expect to see at least several hundred triple insertions per second.

Inference recomputation and search index rebuilding

These operations are important for VIVO installations that modify data directly in the triple store instead of adding or removing RDF through VIVO. You should expect inference recomputation to average about 20-25 milliseconds per individual. (You can find your values in `vivo.all.log`.) Search index rebuilding is typically faster, on the order of 10 ms per individual.

9.8.2.3 Tools for measuring performance

Members of the VIVO community have found the following tools helpful in testing and measuring a site's performance:

- Google Analytics. Records some basic performance metrics in the Behavior > Site Speed section, such as average page load time.
- JMeter. Generates simultaneous connections for testing of performance under real-world production loads.
- New Relic. Software analytics suite including JVM and MySQL monitoring.

Testing without local modifications

Local code modifications – especially custom list views and filter policies – can introduce inefficiencies that lead to poor performance. Similarly, code under development may contain performance regressions or new features that have not yet been optimized. If you have made any such modifications or are using pre-release code, it is important to test performance when your VIVO database is used with an official VIVO release. If the observed performance differs significantly from that exhibited by a modified version, the modifications are suspect.

9.8.2.4 Tuning for improved performance

Memory

Ensure that that Java JVM for your VIVO has been allocated sufficient memory (heap space). This is a critical element of the installation process, as the default Java heap setting will cause VIVO to run extremely slowly. A production VIVO installation should typically be allocated several gigabytes of heap space.

Additionally, ensure that your server has enough memory to support the heap space you have allocated. Otherwise, data may be swapped to disk, which can seriously degrade performance. On a server that runs only VIVO, the available memory should be about double the Java heap space.

Server connections

A production VIVO installation often involves an Apache web server, the Tomcat servlet container, and a MySQL database server. The numbers of available connections between each of these servers should be set to prevent unnecessary bottlenecks. Thus, the maximum number of database connections should slightly exceed the number of possible concurrent Tomcat threads, which should in turn exceed the number of simultaneous Apache worker threads or child processes.

MySQL configuration

Data display in VIVO often depends on complex SPARQL queries that, when using the default SDB triple store, are translated into similarly complex SQL queries. Tuning the MySQL database server can significantly increase performance. There are a number of tools available for assisting with this process, such as mysqltuner.pl[148] (https://github.com/rackerhacker/MySQLTuner-perl). There are also a few typical parameters that often require adjustment.

In-memory temporary tables

The nature of the SQL queries generated by the triple store often requires the generation of temporary tables. Ideally these temporary tables will remain in memory; if they exceed the threshold where MySQL writes them to disk, this can result in serious slowdowns. Depending on the amount of data in your VIVO and your server's available memory, you may need to increase the size limit for in-memory temporary tables.

Consult the MySQL documentation for the parameters

- tmp_table_size
- max_heap_table_size

Key buffer size

If your VIVO database uses MySQL's traditional MyISAM storage engine, consult the documentation for the key_buffer_size parameter. Increasing this value can yield significant performance benefit.

InnoDB buffer pool size

If your VIVO database uses MySQL's newer InnoDB storage engine, consult the documentation for the innodb_buffer_pool_size parameter. Setting this value as large as possible given available memory will improve performance.

Transaction logging

Changing MySQL's transaction logging settings can lead to dramatic improvements to the speed at which triples are added to or removed from the database. For more details, see „Writing the MySQL transaction log" here: MySQL tuning, and troubleshooting (see page 281)

HTTP caching

If VIVO's dynamically-generated pages do not exhibit acceptable load times, you may wish to enable HTTP caching. See Use HTTP caching to improve performance (see page 283). With this configuration, subsequent requests for pages whose contents have not changed will result in those pages being served directly from a cache instead of being regenerated from data in the triple store.

Alternative triple stores

While VIVO is tested with and configured by default to use Jena's SDB triple store with the MySQL database server, VIVO also includes support for TDB and Virtuoso as well as the ability to connect via HTTP to a SPARQL 1.1-complaint endpoint. Use of a different store may yield performance improvements, offer additional possibilities for performance tuning, or enable features such as clustering and load balancing. In addition, configuring SDB to use a database server other than MySQL may offer advantages for your installation.   Note that some of the SPARQL

148 http://mysqltuner.pl

queries in the list views (see page 278) employed by VIVO in page rendering have been optimized for SDB/MySQL with substitution of UNION for OPTIONAL. These queries should be modified for optimum performance with other stores that do not exhibit the same quirks.

Misbehaving robots

In some cases, poor VIVO performance has been traced to search engine robots that either ignore or misread directives in VIVO's robots.txt file, or which issue requests for large pages at a rate that greatly exceeds the demand otherwise encountered in typical production use. If the search engine in question is not critical to VIVO's visibility, it may be advisable to restrict access to the associated robots. In some situations, institutional search appliances are responsible for the excessive server load. Here, discussions with local IT staff may be warranted.

9.8.3 MySQL tuning, and troubleshooting

9.8.3.1

9.8.3.2 Tuning MySQL

From Stony Brook –

By popular request, I've been asked to re-send information about the MySQLTuner tool. It helped give us feedback on several key mysql tuning parameters. And it gives suggestions on settings that may help your system run more efficiently, and thus your VIVO run a little bit faster.
The mysqltuner.pl[149] script can be found at:
https://github.com/rackerhacker/MySQLTuner-perl

From Mark at Griffith Uni -
We use an enterprise hosted MySQL ie. remote to our vivo server via gigabit ethernet. In this configuration we have found MySQL to be a real performance bottleneck. Here are some parameters that we have found it worthwhile experimenting with:
innodb_flush_log_at_trx_commit=2

- this resulted in about a 3x speedup (especially for big ingests)

tmp_table_size

max_heap_table_size

149 http://mysqltuner.pl/

key_buffer_size (needed because many of our queries include a group or sort)

Writing the MySQL transaction log

MySQL allows you to control its logging behavior, using the the `innodb_flush_log_at_trx_commit` parameter. On some systems, changing the value of this parameter can dramatically improve performance.

Using the default setting, the log is written to the file buffer and the buffer is flushed to disk at the end of each transaction. This is necessary to insure full ACID compliance, but the overhead is substantial. Most of VIVO is not transaction-oriented: each statement is added or deleted in its own transaction. So the default setting means that a physical write to disk is required for each new RDF statement.

Setting `innodb_flush_log_at_trx_commit` to 0 or 2 will greatly improve throughput, while adding a minimal level of risk to the data. Under some circumstances, with some settings, up to one second of transactions can be lost. Most VIVO installations will find this to be an acceptable level of risk.

setting	meaning	worst case risk
1 (default)	Write the log after each transaction. Flush to disk after each transaction.	If MySQL crashes, lose transactions in progress. On power failure or system crash, lose transactions in progress.
2	Write the log after each transaction. Flush to disk once per second.	If MySQL crashes, lose transactions in progress. On power failure or system crash, lose one second of transactions.
0	Write the log once per second. Flush to disk once per second.	If MySQL crashes, lose one second of transactions. On power failure or system crash, lose one second of transactions.

This page provides full details regarding `innodb_flush_log_at_trx_commit`: http://dev.mysql.com/doc/refman/5.1/en/innodb-parameters.html#sysvar_innodb_flush_log_at_trx_commit

Setting the MySQL query cache size

Increasing the MySQL query cache size will likely translate into improved VIVO performance in that once large pages have been fetched once, they're typically quite a bit faster to load on later fetches.

Tracing back from SQL to SPARQL

If we identify particularly slow SQL queries, we can try to trace them back to SPARQL queries in the code and look for optimizations to those queries or attempt to solve the problem in a different way.

One approach is to watch the status of the MySQL query process during slow queries or page rendering to see what it's doing and/or do an EXPLAIN SELECT on the generated SQL.

Regenerating MySQL indexes

If performance is abysmal on a simple query, check for missing or corrupted MySQL indexes that may cause the query engine to do full table scans.

TCMalloc and MySQL

Interesting GitHub blog post (https://github.com/blog/1422-tcmalloc-and-mysql) describing debugging MySQL performance issues, and using tools like the open source Percona Toolkit[150] and the Google-contributed TCMalloc from gperftools[151].

9.8.4 Use HTTP caching to improve performance

As a VIVO implementation grows in size and tracks more and more scholarly activity, profile pages can be pulling in hundreds of relationships to render the page, which results in more data being retrieved from the underlying triple store and longer page load times. For example, a profile page for a faculty member with hundreds of publications, which isn't uncommon, can lead to multiple second page loads.

Instead of querying the database each time a page is loaded, a cached version of the page can be served, provided the user is not logged in. VIVO supports HTTP caching directly. To enable, uncomment the "http.createCacheHeaders = true" line in runtime.properties:

runtime.properties

```
# Tell VIVO to generate HTTP headers on its responses to facilitate caching the
# profile pages that it creates.
#
# For more information, see this wiki page:
# https://wiki.duraspace.org/display/VIVO/Use+HTTP+caching+to+improve+performance
#
# Developers will likely want to leave caching disabled, since a change to a
# Freemarker template or to a Java class would not cause the page to be
# considered stale.
#
  http.createCacheHeaders = true
```

VIVO will now generate eTags for caching, which are stored in VIVO's Solr index. More information is available from Ted Lawless, who originally demonstrated the eTag method, here[152].

Next, enable mod_cache in Apache by uncommenting LoadModule lines in httpd.conf:

httpd.conf

```
LoadModule cache_module modules/mod_cache.so
LoadModule cache_disk_module modules/mod_cache_disk.so
```

and adding the following configuration lines to httpd.conf or in its own .conf file within Apache's conf.d directory:

150 http://www.percona.com/doc/percona-toolkit/2.1/
151 http://code.google.com/p/gperftools/
152 https://lawlesst.github.io/notebook/vivo-caching.html

mod_cache.conf

```
#The default expire needs to be 0 in a self-editing environment so that E-Tags can be reverified.
#Requests to cached URLs that haven't expired will never reach the VIVO web application.
#

<IfModule mod_cache.c>
     CacheRoot /var/cache/apache2
     CacheEnable disk /display
     CacheEnable disk /individual
     CacheIgnoreNoLastMod On
     CacheDefaultExpire 0
     CacheMaxExpire 0
     CacheIgnoreHeaders Set-Cookie
</IfModule>
```

The above configuration was provided by Ted Lawless. Restart Apache and Tomcat. Large pages should now load significantly faster for logged-out users.

You can verify http caching is occurring by looking in the directory specified as CacheRoot and seeing if files are being added. You can also use your browser's debugging tools, like Firebug or Chrome debug tools, to inspect the HTTP status code of the response for a profile page. In Chrome, enable Developer Tools (View > Developer > Developer Tools, or ⌥⌘I) and select 'Network' on the pane that appears. Cached pages will return a 304 "Not Modified" response.

9.8.5 HTTP Cache Awareness (*)

VIVO adds headers to some HTTP responses, to assist in caching profile pages

9.8.5.1 Overview

VIVO doesn't cache, but it helps to support caching.

9.8.5.2 How to enable cache awareness

What runtime properties are used to control it? Can it be controlled in developer mode?

9.8.5.3 What pages can be cached?

Only works on profile pages, and only if you are not logged in.

9.8.5.4 What do the caching headers look like?

Show a simple request with a cacheable response. Show a conditional request with a current ETag, Show a conditional request with a stale ETag.

9.8.5.5 How to configure your cache

It's up to you to insure that you don't cache something without an ETag. You should assume that all pages are stale.

9.9 Virtual Machine Templates

- Docker (see page 285)
- Vagrant (see page 285)

9.9.1 Docker

Justin Littman[153] has created code for dockerizing VIVO. Docker for VIVO is available on GitHub[154]

9.9.2 Vagrant

Ted Lawless[155] has created a Vagrant box to allow for quickly installing and testing the full VIVO application. The VIVO Vagrant is available on Github[156].

9.10 Moving your VIVO Instance

This page describes what you would need to do to move your VIVO instance from one machine to another.

153 https://wiki.duraspace.org/display/~justinlittman
154 https://github.com/gwu-libraries/vivo-docker
155 https://wiki.duraspace.org/display/~tlawless
156 https://github.com/lawlesst/vivo-vagrant

9.10.1 Step-by-step guide

1. Make a backup of your current VIVO source directory
2. Make a backup of your current VIVO relational database
3. If different from your relational database, make a backup of your current VIVO triple store
4. Copy these backup files to your new machine
5. Create the vivo database, using the same username and password as the previous machine
6. Load the relational database from the backup
7. If you're installing everything into the same place that they were installed on the original machine, then there are no configuration changes to be made
8. Otherwise, you'll need to modify your `build.properties` in the VIVO source directory, and `runtime.properties` in the VIVO home directory, changing any paths necessary
9. If your relational database and triple store information are the same as before (same graphs, same usernames, same passwords), then there are no configuration changes to be made
10. Otherwise, you'll need to modify your `*.properties` files (see above), changing any username and password information for relational and semantic stores
11. Make sure tomcat is NOT running prior to building and installing VIVO
12. Build and install VIVO
13. Start tomcat

And that should be it.

9.11 Regaining access to the root account

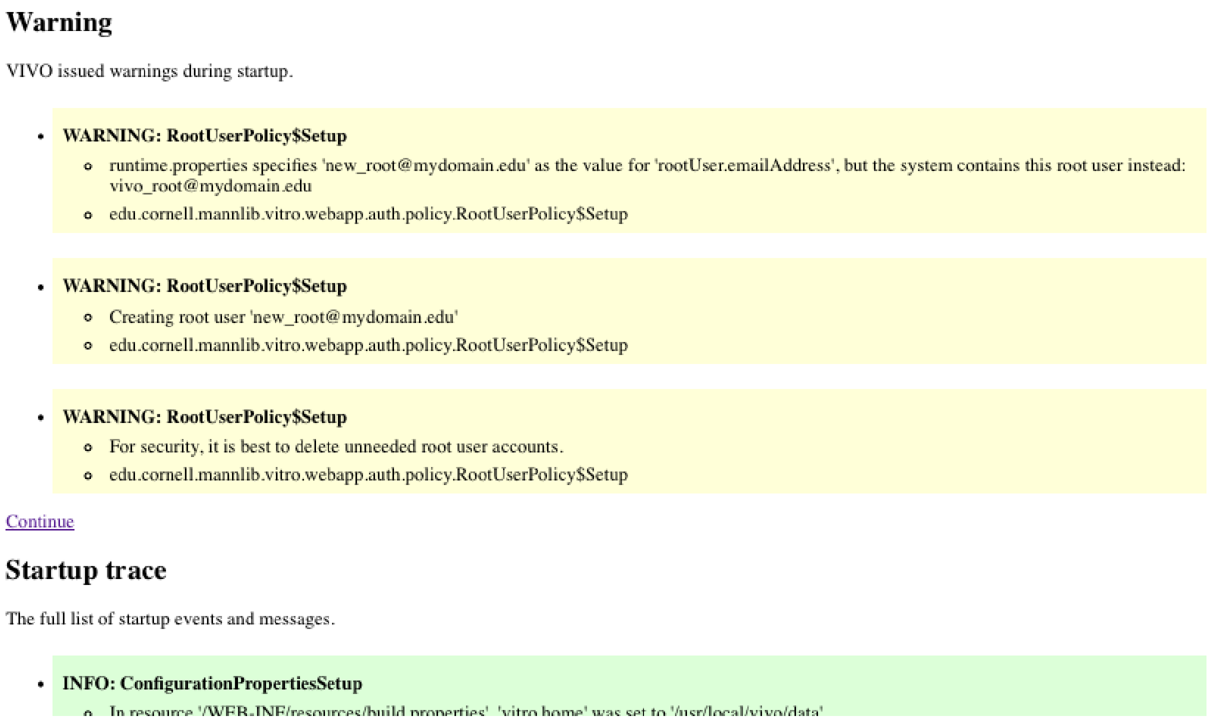
⚠ This page is intended to make access easier for VIVO developers and maintainers. An attacker cannot use these techniques to gain access to your VIVO installation. These techniques can only be used by someone who already has full access to your installation.

To gain access to the database, create a new root account.

- Modify the `runtime.properties` file to include a root account of your choosing, and restart VIVO

```
rootUser.emailAddress = new_root@mydomain.edu
```

- Open VIVO in the browser. You will see a warning screen like the following:



Warning

VIVO issued warnings during startup.

- **WARNING: RootUserPolicy$Setup**
  - runtime.properties specifies 'new_root@mydomain.edu' as the value for 'rootUser.emailAddress', but the system contains this root user instead: vivo_root@mydomain.edu
  - edu.cornell.mannlib.vitro.webapp.auth.policy.RootUserPolicy$Setup
- **WARNING: RootUserPolicy$Setup**
  - Creating root user 'new_root@mydomain.edu'
  - edu.cornell.mannlib.vitro.webapp.auth.policy.RootUserPolicy$Setup
- **WARNING: RootUserPolicy$Setup**
  - For security, it is best to delete unneeded root user accounts.
  - edu.cornell.mannlib.vitro.webapp.auth.policy.RootUserPolicy$Setup

Continue

Startup trace

The full list of startup events and messages.

- **INFO: ConfigurationPropertiesSetup**
  - In resource '/WEB-INF/resources/build.properties', 'vitro.home' was set to '/usr/local/vivo/data'

  Click `Continue` to view the VIVO home page.
- Log in using the new root account. The first-time password for your new root account will be `rootPassword`, and you will be asked to assign a new password.

You now have two root accounts, and you know the password to the new one. Use the User Accounts pages to either

- Delete the old root account,
  or
- Set a fresh password on the old root account and delete the new root account.

9.12 Altmetrics Support

- Overview (see page 287)
- Display (see page 288)
- Configuration (see page 288)

9.12.1 Overview

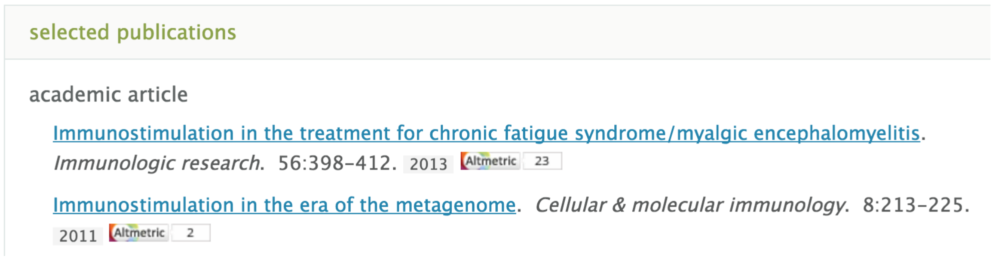
"Altmetrics" is a general term for non-traditional metrics related to scholarly works. See Wikipedia: https://en.wikipedia.org/wiki/Altmetrics

"Altmetric" is a company, a division of Digital Science, that collects altmetrics and makes them available via APIs. See http://altmetric.com

VIVO uses APIs provided by Altmetric to provide altmetrics on scholarly works. Altmetrics makes its service available without fee or license restriction.

9.12.2 Display

Scholarly works identified by DOI, PubMed ID, or ISBN have altmetric "badges" associated with them. These badges are links to altmetrics information provided by Altmetrics.



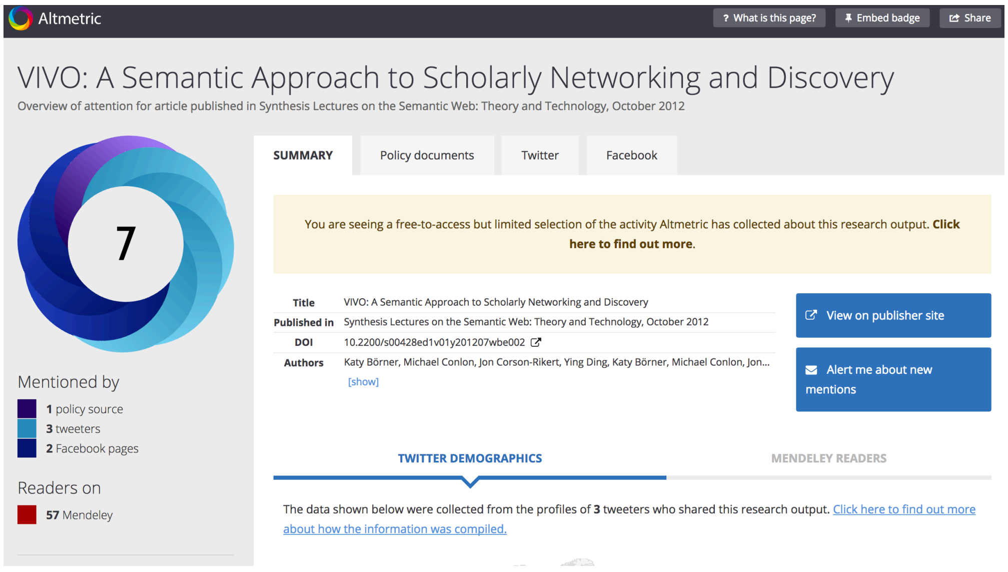
selected publications

academic article

Immunostimulation in the treatment for chronic fatigue syndrome/myalgic encephalomyelitis. *Immunologic research.* 56:398-412. 2013 Altmetric 23

Immunostimulation in the era of the metagenome. *Cellular & molecular immunology.* 8:213-225. 2011 Altmetric 2

Clicking on a badge takes you to the Altmetrics web site page for the scholarly work. For example:



VIVO: A Semantic Approach to Scholarly Networking and Discovery

Overview of attention for article published in Synthesis Lectures on the Semantic Web: Theory and Technology, October 2012

You are seeing a free-to-access but limited selection of the activity Altmetric has collected about this research output. **Click here to find out more.**

Title	VIVO: A Semantic Approach to Scholarly Networking and Discovery
Published in	Synthesis Lectures on the Semantic Web: Theory and Technology, October 2012
DOI	10.2200/s00428ed1v01y201207wbe002
Authors	Katy Börner, Michael Conlon, Jon Corson-Rikert, Ying Ding, Katy Börner, Michael Conlon, Jon... [show]

Mentioned by

■	**1** policy source
■	**3** tweeters
■	**2** Facebook pages

Readers on

■	**57** Mendeley

The data shown below were collected from the profiles of **3** tweeters who shared this research output. Click here to find out more about how the information was compiled.

9.12.3 Configuration

Six configuration parameters regulate how VIVO uses and displays altmetrics. See Configuration Reference (see page 295)

9.13 Troubleshooting

- Having problems with your VIVO installation? (see page 288)
- Can't find any individuals? (see page 289)
- Mail not working? (see page 289)

9.13.1 Having problems with your VIVO installation?

- Check your $TOMCAT DIRECTORY/logs - specifically catalina.out and vivo.all.log

- If you can't find vivo.all.log check that the data folder defined in your runtime.properties file (commonly /usr/local/vivo/home) is defined properly and is writable by Tomcat.

9.13.2 Can't find any individuals?

- First, try restarting Tomcat and go to [yourhost]/vivo/SearchIndex to see whether rebuilding the search index will fix the problem
- In the [tomcat]/logs directory, check vivo.all.log to see whether there are any error messages related to Solr
- Go to [yourhost]/vivosolr to see whether the Solr greeting page appears
  - If it does appear, then Vivo just can't reach it. Make sure that vitro.local.solr.url is set correctly in runtime.properties.
  - If you get a 403 HTTP error, then the authorization on Solr is a problem. Check your permissions.
  - If it does not appear, and you don't get a 403, then Solr did not install properly. Try cleaning the [tomcat]/webapps directory and [tomcat]/conf/Catalina/localhost directory, and rebuild VIVO using Maven
- To see your individual, go to the Site Admin page
  - click on 'Class Hierarchy'
  - navigate to the FacultyMember class link and select that link
  - on the left side of the page select the button 'show all individuals in this class'
- If an individual is found, you can select 'raw statements with this individual as subject' and you can also select 'display this individual (public)' and from there select the 'RDF' link to show the underlying RDF for the Person and some associated enitities.

9.13.3 Mail not working?

- In order for VIVO to send e-mails, it needs to have access to an SMTP server. In runtime.properties, you can set email.smtphost to the name of an SMTP server that will accept messages from your VIVO host.
- If you don't have access to an SMTP server, comment out the line for email.smtphost. VIVO will detect this, and will not attempt to send e-mails to the users. Instead, you will be required to set a password on each account as you create it, and the user will be required to change that password the first time he logs in.
- You may want to test emailing people from your server.

9.13.4 Troubleshooting Tips

9.13.4.1

- Warning screen at startup
- Rebuilding the Search Index
- How to Serve Linked Data
- Long URLS

9.13.4.2 Warning screen at startup

As VIVO goes through its startup process, it executes a series of "smoke tests" to try to confirm that the configuration is correct. For example, it checks to see that the home directory exists, and that VIVO has permission to write to it. It checks that VIVO can connect to the database. It checks that Solr is running, and that VIVO can connect to it.

If any of these tests fail, you will see a warning or error message when you direct your browser to VIVO. If the message is a warning (yellow), you may click the "continue" link to ignore the warning. If the message is an error (red), it is considered fatal, and VIVO will not respond to any requests.

Some of the warnings or errors may be cryptic, but they are intended to offer clues as to why your VIVO installation will not work properly.

9.13.4.3 Rebuilding the Search Index

The search index of VIVO is used not just for full text search but also for the menu pages and index pages. If the system is not displaying the individuals that you would expect to see, the search index may need to be rebuilt. To rebuild the index log in as an administrative user and request

```
http://vivo.example.edu/SearchIndex
```

This page will allow you to start a rebuild of the search index. A rebuild may take some time. The browser page will refresh every few seconds. Once the index rebuild is set up, the page will display how much time the rebuild has taken, and an estimate of how much additional time will be needed. When the indexing is completed, the page will return to its previous state.

9.13.4.4 How to Serve Linked Data

The default namespace value set during installation needs to match the domain name where you are serving your VIVO application from (VIVO web address).

Examples of VIVO web addresses and default namespace values:

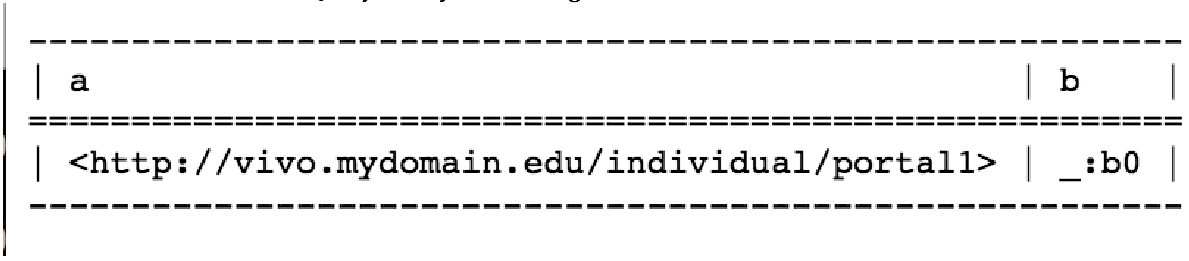
VIVO web address (url)	**Default namespace value**
http://vivo.example.edu	http://vivo.example.edu/individual
http://vivo.example.edu/vivo/	http://vivo.example.edu/vivo/individual/
http://vivoTEST.example.edu:8080/	http://vivoTEST.example.edu:8080/individual/

To check what your default namespace is currently set for:

1. Log into VIVO as an administrator, go to Site Admin -> SPARQL query.
2. Clear all of the text from the text area, enter the following query in the text area:

   ```
   SELECT ?a ?b WHERE { ?a <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#rootTab> ?b }
   ```

3. Scroll down and click “Run Query” and you should get a result like this:



```
-------------------------------------------------------
| a                                            | b    |
=======================================================
| <http://vivo.mydomain.edu/individual/portal1> | _:b0 |
-------------------------------------------------------
```

4. To get the default namespace from the result, take everything in braces up to and including the last forward slash. In this case the default namespace is

```
http://vivo.mydomain.edu/individual/
```

5. If the default namespace does not match the domain name where your VIVO application is installed, follow the steps below:
    a. Use the “Change Namespace of Resources” option under Site Admin – Ingest Tools to set the default namespace to match your VIVO application domain name as in the above examples.
    b. Set `Vitro.defaultNamespace` in `runtime.properties` to the value for your namespace
    c. Restart Tomcat

9.13.4.5 Long URLS

If you checked your default namespace and ensured it matches the domain name where your VIVO application is installed, you may find that you still have long URLs on some people profiles.

In other words, you expect to have URLs like this: *

```
http://vivo.example.edu/individual/n5143
```

But instead, you have URLs like this:

```
http://example.edu/individual?uri=http%3A%2F%2Fvivo.example.edu%2Fsomethingl%2Fn5143
```

In this case, you have individuals with URIs that are not in your VIVO application’s default namespace. There are a couple of ways that this could have happened:

The individuals could have been created using a ingest process that did not create individuals in the default namespace.

The individuals could have been created when the system had a different default namespace.

The individuals could be from RDF data that was imported.

In general, once you have the default namespace set up correctly for your VIVO application, then all the individuals you create using the web interface will have the default namespace. You have to be careful to make sure that any individuals created by an ingest process use the default namespace.

Some individuals that are shipped with the application are not in the default namespace. For example, the countries and geographical locations are in a different namespace. Do not attempt to change the namespace of these individuals.

9.14 High Availability

9.14.1 Overview

VIVO, as delivered, is not a high availability application. Single points of failure in the application are addressed below. Some of these can be improved by approaches to deployment as noted. Others would require additional development to provide high availability deployment options.

9.14.2 Session management

VIVO code makes use of HttpSession objects. Sessions can be replicated in Tomcat and/or sticky routing to the servers.

9.14.3 Caching

VIVO does limited caching. VIVO caches some information in the visualisation stack. This is not critical to the operation of VIVO, as application servers can each build their own cache. Sticky routing, so that people get consistent graphs in a single session may be sufficient, even if each server could vary slightly in what is displayed.

9.14.4 Solr

Every server must use a single Solr cluster, rather than relying on Solr being installed alongside VIVO. Any changes being written to the index would then be shared by all instances. A shared cluster also takes care of the file system storage of Solr, which is currently maintained in the VIVO home directory.

9.14.5 Home directory

VIVO uses static configuration information, the config and rdf directories, and runtime.properties. These need to be consistent across multiple servers. That could be achieved via a shared home directory, or just multiple identical deployments.

There are three additional areas in the home directory that are of concern. The configuration triple store (tdbModels) is addressed below. Solr indexes are addressed above. The upload directory stores thumbnails for people, etc. If you allowing real-time upload of photos, this directory needs to be on a shared HA filesystem. If you are only batch ingesting thumbnails from external sources, then syncing the directory across servers could suffice. If you are simply linking to externally hosted images, the uploads folder will not be a concern.

9.14.6 Content triple store

By default, this is SDB, stored in MySQL. An HA MySQL configuration should permit multiple application servers to access the same MySQL server cluster.

9.14.7 Configuration triple store

The configuration triple store is TDB, stored in the tdbModels folder in the home directory. TDB requires that you only have one JVM accessing a TDB triple store. Replication is not possible while the TDB files are open. There are two potential solutions. Through disciplined system administration you may find that the material in the configuration triple store can be considered static. The triple store can then be replicated across each server using a copy. A second approach would involve storing the configuration triple store using SDB in an HA MySQL cluster. This would involve recoding relevant parts of the Vitro application, which appears to be feasible.

... (see page 292)

9.15 Replicating Ontology Changes Across Instances

9.15.1 Purpose

Suppose changes are made to the VIVO core ontology through the web interface on one VIVO instance, and these changes are needed in another instance. For example, changes are made in a development instance, tested, and approved for deployment in production. Changes may include:

- changing the display label of a core class or property
- changing the property group of a core class or property
- changing the display rank of a core class or property

The procedure below describes how such changes can be replicated between instances.

9.15.2 Procedure

In the steps below, instance #1 is the the instance that contains the changes you have made to the core ontology. Instance #2 is the instance you wish to copy the ontology changes to.

1. On instance #1, go to Site Admin > Ingest Tools > Manage Jena Models.
2. Find "http://vitro.mannlib.cornell.edu/default/asserted-tbox" and click "output model."
3. On instance #2, locate the same model and click "clear statements."
4. On instance #2, under the same model, click "load RDF data."
5. Load the file output in step 2 (N3 format).
6. Restart instance #2.

9.15.3 Best Practice

Semantic additions to the core ontology (new classes and properties) should made in a local ontology, isolated from the core ontology. Additions should be discussed on vivo-tech@googlegroups.com[157] to insure they are necessary and represent common ontological practice. Edits to the core ontology should be rare.

157 mailto:vivo-tech@googlegroups.com

10 Reference

10.1 Overview

This section contains reference material for the VIVO and Vitro systems. These materials take the form of glossaries and lists. They are not intended in the form of instructional materials. For processes used to support VIVO and Vitro, see the System Administration (see page 256) section and the introductory sections in particular for processes and instructional material.

10.2 Configuration Reference

10.2.1 Overview

VIVO's operation can be determined by setting corresponding properties in runtime.properties.

10.2.2 VIVO Runtime Properties

Property	Description
`Vitro.defaultNamespace = http://vivo.mydomain.edu/individual/`	This namespace will be used when generating URIs for objects created in the editor. In order to serve linked data, the default namespace must be composed as follows (optional elements in parentheses): scheme + server_name (+ port) (+ servlet_context) + "/individual/" For example, Cornell's default namespace is: http://vivo.cornell.edu/individual/
`rootUser.emailAddress = vivo_root@mydomain.edu`[158]	The email address of the root user for the VIVO application. The password for this user is initially set to "rootPassword", but you will be asked to change the password the first time you log in.

158 mailto:vivo_root@mydomain.edu

Property	Description
`VitroConnection.DataSource.url = jdbc:mysql://localhost/vitrodb` `VitroConnection.DataSource.username = vitrodbUsername` `VitroConnection.DataSource.password = vitrodbPassword`	The basic parameters for a database connection. Change the end of the URL to reflect your database name (if it is not "vitrodb"). Change the username and password to match the authorized database user you created.
`email.smtpHost = smtp.mydomain.edu`[159] `email.replyTo = vivoAdmin@mydomain.edu`[160]	Email parameters which VIVO can use to send mail. If these are left empty, the "Contact Us" form will be disabled and users will not be notified of changes to their accounts.
`vitro.local.solr.url = http://localhost:8080/vivosolr`	URL of Solr context used in local VIVO search. This will usually consist of: scheme + server_name + port + vivo_webapp_name + "solr" In the standard installation, the Solr context will be on the same server as VIVO, and in the same Tomcat instance. The path will be the VIVO webapp.name[161] (specified in build.properties) + "solr" Example: vitro.local.solr.url = http://localhost:8080/vivosolr
`selfEditing.idMatchingProperty = http://vivo.mydomain.edu/ns#networkId`	How is a logged-in user associated with a particular Individual? One way is for the Individual to have a property whose value is the username of the user. This value should be the URI for that property.
`externalAuth.netIdHeaderName = remote_userID`	If an external authentication system such as Shibboleth or CUWebAuth is to be used, this property says which HTTP header will contain the user ID from the authentication system. If such a system is not to be used, leave this commented out. See Using an external authentication system[162]

159 http://smtp.mydomain.edu
160 mailto:vivoAdmin@mydomain.edu
161 http://webapp.name
162 https://wiki.duraspace.org/display/VTDA/Using+an+external+authentication+system

Property	Description
`VitroConnection.DataSource.pool.maxActive = 40`	The maximum number of active connections in the database connection pool. Increase this value to support a greater number of concurrent page requests.
`VitroConnection.DataSource.pool.maxIdle = 10`	The maximum number of database connections that will be allowed to remain idle in the connection pool. Default is 25% of the maximum number of active connections.
`VitroConnection.DataSource.dbtype = MySQL` `VitroConnection.DataSource.driver = com.mysql.jdbc.Driver` `VitroConnection.DataSource.validationQuery = SELECT 1`	Parameters to change in order to use VIVO with a database other than MySQL. These parameters allow you to change the relational database that is used as the back end for Jena SDB. If you want to use a triple store other than SDB, you will need to edit applicationSetup.n3. See the installation instructions for more details.
`OpenSocial.shindigURL = http://localhost:8080/shindigorng`	For OpenSocial integration, the base URL of the ORNG Shindig server. Usually, this is the same host and port number as VIVO itself, with a context path of "shindigorng".
`OpenSocial.tokenService = myhost.mydomain.edu`[163]`:8777`	For OpenSocial integration, The host name and port number of the service that provides security tokens for VIVO and Shindig to share. For now, the host name must be the actual host, not "localhost" or "127.0.0.1" The port number must be 8777
`OpenSocial.tokenKeyFile = /usr/local/vivo/data/shindig/openssl/securitytokenkey.txt`	For OpenSocial integration. The path to the key file that will be used when generating security tokens for VIVO and shindig to share.
`OpenSocial.sandbox = True`	For OpenSocial integration. Only set sandbox to True for dev/test environments. Comment out or set to False in production

163 http://myhost.mydomain.edu

Property	Description
`RDFService.languageFilter = false`	Show only the most appropriate data values based on the Accept-Language header supplied by the browser. Default is false if not set.
`languages.forceLocale = en_US`	Force VIVO to use a specific language or Locale instead of those specified by the browser. This affects RDF data retrieved from the model, if RDFService.languageFilter is true. This also affects the text of pages that have been modified to support multiple languages.
`languages.selectableLocales = en_US, es_GO`	A list of supported languages or Locales that the user may choose to use instead of the one specified by the browser. Selection images must be available in the i18n/images directory of the theme. This affects RDF data retrieved from the model, if RDFService.languageFilter is true. This also affects the text of pages that have been modified to support multiple languages. This should not be used with languages.forceLocale, which will override it.

`orcid.clientId = 0000-0000-0000-000X`
`orcid.clientPassword = 00000000-0000-0000-0000-000000000000`
`orcid.webappBaseUrl = http://localhost:8080/vivo`
 	
`orcid.messageVersion = 1.0.23`
`orcid.externalIdCommonName = VIVO Cornell Identifier`
	ORCiD integration parameters. See Activating the ORCID integration (see page 272)

Property	Description

`orcid.publicApiBaseUrl = http://pub.sandbox.orcid.org/v1.1`
 	
`orcid.authorizedApiBaseUrl = http://api.sandbox.orcid.org/v1.1`
 	
`orcid.oauthAuthorizeUrl = http://sandbox.orcid.org/oauth/authorize`
 	
`orcid.oauthTokenUrl = http://api.sandbox.orcid.org/oauth/token`
	Setup for the ORCID sandbox

`orcid.publicApiBaseUrl = http://localhost:8080/mockorcid/mock/`
 	
`orcid.authorizedApiBaseUrl = http://localhost:8080/mockorcid/mock/`
 	
`orcid.oauthAuthorizeUrl = http://localhost:8080/mockorcid/mock/oauth/authorize`
 	
`orcid.oauthTokenUrl = http://localhost:8080/mockorcid/mock/oauth/token`
	Setup for the mockorcid app

Property	Description
`google.maps.key=`	To use the Google Maps (e.g. Map of Science), you need to have a key for Google Maps. See https://developers.google.com/maps/documentation/javascript/get-api-key When you have a key, uncomment the line below and add it here
`resource.altmetric=disabled`	Uncomment and set this to disabled if you don't want AltMetric badges
`resource.altmetric.displayto=right`	Display the badge to the left or right of the title (default = right). Options: left, right
`resource.altmetric.badge-type=donut`	Badge type to display (default = donut) Options: See AltMetric documentation[164] - recommended settings: donut, medium-donut
`resource.altmetric.hide-no-mentions=true`	Hide the badge if there are no mentions (default = true) Options: true, false
`resource.altmetric.badge-popover=right`	Display more details about the score when you hover over the badge (default = right) Options, right, left, up, down
`resource.altmetric.badge-details=right`	Display extended details alongside the badge (default = none)
`homePage.geoFocusMaps=enabled`	When the following flag is set to enabled, the VIVO home page displays a global map highlighting the geographical focus of foaf:person individuals. See Home page customizations (see page 127)
`multiViews.profilePageTypes=enabled`	VIVO supports the simultaneous use of a full foaf:Person profile page view and a "quick" page view that emphasizes the individual's webpage presence. Implementing this feature requires an installation to develop a web service that captures images of web pages or to use an existing service outside of VIVO. See Multiple profile types for foaf:Person (see page 209)
`http.createCacheHeaders = true`	Tell VIVO to generate HTTP headers on its responses to facilitate caching the profile pages that it creates. See Use HTTP caching to improve performance (see page 283) Developers will likely want to leave caching disabled, since a change to a Freemarker template or to a Java class would not cause the page to be considered stale.

164 https://api.altmetric.com/embeds.html#badge-types

Property	Description
`harvester.location = /usr/local/vivo/harvester/`	Absolute path on the server of the Harvester root directory. You must include the final slash. Setting a value for harvester.location indicates that the Harvester is installed at this path. This will enable the Harvester functions in the Ingest Tools page.
`visualization.topLevelOrg = http://vivo.mydomain.edu/individual/topLevelOrgURI`	The temporal graph visualization is used to compare different organizations/people within an organization on parameters like number of publications or grants. By default, the app will attempt to make its best guess at the top level organization in your instance. If you're unhappy with this selection, uncomment out the property below and set it to the URI of the organization individual you want to identify as the top level organization. It will be used as the default whenever the temporal graph visualization is rendered without being passed an explicit org. For example, to use "Ponce School of Medicine" as the top organization: visualization.topLevelOrg = http://vivo.psm.edu/individual/n2862
`visualization.temporal = enabled`	The temporal graph visualization can require extensive machine resources. This can have a particularly noticeable impact on memory usage if The organization tree is deep, The number of grants and publications is large. VIVO 1.3 release mitigates this problem by the way of a caching mechanism hence we can safely set this to be enabled by default.
`proxy.eligibleTypeList = http://xmlns.com/foaf/0.1/Person, http://xmlns.com/foaf/0.1/Organization`	Types of individual for which we can create proxy editors. If this is omitted, defaults to http://www.w3.org/2002/07/owl#Thing

Property	Description
`Vitro.reconcile.defaultTypeList = http://vivoweb.org/ontology/core#Role, core:Role; http://vivoweb.org/ontology/core#AcademicDegree, core:AcademicDegree; http://purl.org/NET/c4dm/event.owl#Event, event:Event; http://vivoweb.org/ontology/core#Location, core:Location; http://xmlns.com/foaf/0.1/Organization, foaf:Organization; http://xmlns.com/foaf/0.1/Person, foaf:Person; http://purl.obolibrary.org/obo/IAO_0000030, obo:IAO_0000030`	Default type(s) for Google Refine Reconciliation Service. The format for this property is id, name; id1, name1; id2, name2 etc. For more information, see Service Metadata from this page: https://github.com/OpenRefine/OpenRefine/wiki/Reconciliation-Service-Api

10.3 Directories and Files

10.3.1 Overview

The directory structure below is for the VIVO source distribution. The binary distribution omits some directories. These are noted below.

The Vitro source distribution has an analogous structure.

10.3.2 High Level Directories

Directory	Description
`./api`	Java source for the webapp
`./home`	RDF and other files needed to load the webapp
`./installer`	Files used by the Maven installer
`./legacy`	Legacy directories and files
`./selenium`	VIVO Selenium Tests. See http://docs.seleniumhq.org
`./webapp`	Templates and other files for building the webapp

10.3.3 Directory Structure

Directory	Description
`./api/src/main`	VIVO source files. Will not be present in the binary distribution
`./api/src/test`	VIVO source test files. Will not be present in the binary distribution
`./api/target/generated-sources`	
`./api/target/generated-test-sources`	
`./api/target/maven-archiver`	
`./api/target/maven-status`	
`./api/target/surefire-reports`	
`./api/target/test-classes`	
`./home/rdf/abox`	

Directory	Description
./home/rdf/applicationMetadata	
./home/rdf/auth	
./home/rdf/display	
./home/rdf/displayDisplay	
./home/rdf/displayTbox	
./home/rdf/tbox	
./home/solr/conf	
./home/solr/data	
./home/src/main	
./home/target/archive-tmp	
./home/upgrade/knowledgeBase	
./home/uploads/file_storage_root	
./installer/home/src	
./installer/home/target	
./installer/solr/src	
./installer/solr/target	
./installer/webapp/src	
./installer/webapp/target	
./legacy/config/licenser	
./legacy/doc/licenses	
./legacy/languages/es_GO	
./legacy/utilities/acceptance-tests	
./legacy/utilities/ISF-transition	
./legacy/utilities/languageSupport	

Directory	**Description**
`./legacy/utilities/LoadTesting`	
`./legacy/utilities/orcid`	
`./legacy/utilities/performance-measurement`	
`./legacy/utilities/pre-compileJSPs`	
`./legacy/utilities/release1.6.1-scripts`	
`./legacy/utilities/releaseScripts`	
`./legacy/utilities/xslt`	
`./selenium/src/test`	
`./selenium/test-output/Command line suite`	
`./selenium/test-output/junitreports`	
`./selenium/test-output/old`	
`./webapp/src/main`	
`./webapp/target/maven-archiver`	
`./webapp/target/vivo-webapp-1.9.1`	
`./webapp/target/war`	

10.4 Graph Reference

10.4.1 Overview

VIVO stores its information in graphs – named collections of triples. Graphs keep data organized by kind, and provide the opportunity for different access rights and management practices to be applied at the graph level. All graphs are available to the VIVO SPARQL query interface. When using SPARQL to query the VIVO data, one does not need to know the graph the data is contained in. Triples in all graphs are available to the query. When updating data in VIVO using CONSTRUCT or UPDATE, knowledge of the graph may be necessary.

Here we show how to list the graphs in a VIVO, and provide a reference for the purpose of each graph.

10.4.2 Listing the graphs used by VIVO

To list the graphs being used by your VIVO, you can run the SPARQL query shown below. Caution: If you have a significant amount of data in your VIVO, the query may take quite a while to run. With tens of thousands of entities in your VIVO, the query should complete in a few minutes.

SPARQL query to list the graphs in a VIVO

```
SELECT ?g
WHERE
{
    GRAPH ?g {
        ?s ?p ?o .
    }
}
GROUP BY ?g
ORDER BY ?g
```

To list the triples in a named graph, use the query below, substituting the name of the graph you wish to list. Caution: listing the triples in larger graphs may take significant time.

SPARQL query to list the triples in a named graph

```
SELECT ?s ?p ?o
WHERE
{
    GRAPH <http://vitro.mannlib.cornell.edu/filegraph/tbox/sameAs.n3> {
        ?s ?p ?o .
    }
}
```

10.4.3 The graphs used by VIVO

Graph name	Contents
http://vitro.mannlib.cornell.edu/default/asserted-tbox	All ontology triples as asserted
http://vitro.mannlib.cornell.edu/default/inferred-tbox	Triples infered from the asserted ontology triples
http://vitro.mannlib.cornell.edu/default/vitro-kb-2	The main triple store for content

Graph name	Contents
http://vitro.mannlib.cornell.edu/default/vitro-kb-applicationMetadata	Triples controlling the application
http://vitro.mannlib.cornell.edu/default/vitro-kb-inf	Triples created by the inferencer
http://vitro.mannlib.cornell.edu/filegraph/abox/academicDegree.rdf	Controlled vocabulary for academic degrees
http://vitro.mannlib.cornell.edu/filegraph/abox/continents.n3	Data provided regarding the continents
http://vitro.mannlib.cornell.edu/filegraph/abox/dateTimeValuePrecision.owl	Controlled vocabulary for dateTimePrecision
http://vitro.mannlib.cornell.edu/filegraph/abox/documentStatus.owl	Controlled vocabulary for documentStatus
http://vitro.mannlib.cornell.edu/filegraph/abox/geopolitical.abox.ver1.1-11-18-11.owl	Data provided regarding geopolitical entities
http://vitro.mannlib.cornell.edu/filegraph/abox/grid.n3	*Example of a data package.* Grid data regarding organizations
http://vitro.mannlib.cornell.edu/filegraph/abox/us-states.rdf	Data provided regarding US states and territories
http://vitro.mannlib.cornell.edu/filegraph/abox/validation.n3	Data regarding validated ORCiD identifiers
http://vitro.mannlib.cornell.edu/filegraph/abox/vocabularySource.n3	Data regarding external vocabulary services
http://vitro.mannlib.cornell.edu/filegraph/tbox/agent.owl	Ontology assertions regarding agents
http://vitro.mannlib.cornell.edu/filegraph/tbox/appControls-temp.n3	?
http://vitro.mannlib.cornell.edu/filegraph/tbox/bfo-bridge.owl	Ontology assertions relating VIVO entities to BFO
http://vitro.mannlib.cornell.edu/filegraph/tbox/bfo.owl	Ontology assertions regarding Basic Formal Ontology (BFO)
http://vitro.mannlib.cornell.edu/filegraph/tbox/classes-additional.owl	Ontology assertions regarding classes used in VIVO
http://vitro.mannlib.cornell.edu/filegraph/tbox/clinical.owl	Ontology assertions regarding clinical trials

Graph name	Contents
http://vitro.mannlib.cornell.edu/filegraph/tbox/contact-vcard.owl	Ontology assertions regarding vcard
http://vitro.mannlib.cornell.edu/filegraph/tbox/contact.owl	Ontology assertions regarding OBO classes
http://vitro.mannlib.cornell.edu/filegraph/tbox/data-properties.owl	Ontology assertions regarding data properties
http://vitro.mannlib.cornell.edu/filegraph/tbox/dataDomains.rdf	Ontology assertions regarding data domains
http://vitro.mannlib.cornell.edu/filegraph/tbox/dataset.owl	Ontology assertions to define Dataset class
http://vitro.mannlib.cornell.edu/filegraph/tbox/date-time.owl	Ontology assertions to define DateTimeInterval class
http://vitro.mannlib.cornell.edu/filegraph/tbox/dateTimeValuePrecision.owl	Ontology assertions to define DateTimeValuePrecision classes
http://vitro.mannlib.cornell.edu/filegraph/tbox/documentStatus.owl	Ontology assertions regarding document status
http://vitro.mannlib.cornell.edu/filegraph/tbox/education.owl	Ontology assertions regarding educational processes, roles, and classes
http://vitro.mannlib.cornell.edu/filegraph/tbox/event.owl	Ontology assertioons regarding event processes, roles, and classes
http://vitro.mannlib.cornell.edu/filegraph/tbox/geo-political.owl	Ontology assertions required by geopolitical data
http://vitro.mannlib.cornell.edu/filegraph/tbox/grant.owl	Ontology assertions regarding grants, roles, and classes
http://vitro.mannlib.cornell.edu/filegraph/tbox/linkSuppression.n3	?
http://vitro.mannlib.cornell.edu/filegraph/tbox/location.owl	Ontology assertions regarding location classes
http://vitro.mannlib.cornell.edu/filegraph/tbox/object-properties.owl	Ontology assertions to define object properties
http://vitro.mannlib.cornell.edu/filegraph/tbox/object-properties2.owl	Ontology assertions to define BFO properties
http://vitro.mannlib.cornell.edu/filegraph/tbox/object-properties3.owl	Ontology assertions to define more object properties

Graph name	Contents
http://vitro.mannlib.cornell.edu/filegraph/tbox/objectDomains.rdf	Ontology assertions to define object domains
http://vitro.mannlib.cornell.edu/filegraph/tbox/objectRanges.rdf	Ontology assertions to define object ranges
http://vitro.mannlib.cornell.edu/filegraph/tbox/ontologies.owl	?
http://vitro.mannlib.cornell.edu/filegraph/tbox/orcid-interface.n3	Ontology assertions for ORCiD and confirmed ORCiD
http://vitro.mannlib.cornell.edu/filegraph/tbox/other.owl	Ontological definitions of object properties
http://vitro.mannlib.cornell.edu/filegraph/tbox/outreach.owl	Ontological definitions of BFO object properties
http://vitro.mannlib.cornell.edu/filegraph/tbox/personTypes.n3	Ontology assertions to define types of people
http://vitro.mannlib.cornell.edu/filegraph/tbox/process.owl	Ontology assertions to define miscelleaneous processes
http://vitro.mannlib.cornell.edu/filegraph/tbox/publication.owl	Ontology assertions to define publication types and related roles
http://vitro.mannlib.cornell.edu/filegraph/tbox/relationship.owl	Ontology assertions to define relationships, including positions
http://vitro.mannlib.cornell.edu/filegraph/tbox/relationshipAxioms.n3	Ontology axioms regarding restrictions on relationships
http://vitro.mannlib.cornell.edu/filegraph/tbox/research-resource-iao.owl	Ontology assertions to define elements of IAO needed for research resources
http://vitro.mannlib.cornell.edu/filegraph/tbox/research-resource.owl	Ontology assertions to define ERO research resources
http://vitro.mannlib.cornell.edu/filegraph/tbox/research.owl	Ontology assertions to define clinical trials
http://vitro.mannlib.cornell.edu/filegraph/tbox/role.owl	Ontology assertions to define miscellaneous roles
http://vitro.mannlib.cornell.edu/filegraph/tbox/sameAs.n3	Ontology assertions defining sameAs
http://vitro.mannlib.cornell.edu/filegraph/tbox/skos-vivo.owl	Ontology assertions to define various classes as subtypes of skos:Concept

Graph name	Contents
http://vitro.mannlib.cornell.edu/filegraph/tbox/teaching.owl	Ontology assertions for Course and TeacherRole
http://vitro.mannlib.cornell.edu/filegraph/tbox/vitro-0.7.owl	Ontology assertions for the Vitro application, internal
http://vitro.mannlib.cornell.edu/filegraph/tbox/vitroPublic.owl	Ontology assertions for the Vitro application, public. Defines files and file types

10.4.4 Notes

1. Graphs named "default" are built and managed by the Vitro application. Graphs names "filegraph" are loaded from files when VIVO starts. Graphs named "filegraph/abox" are data. Graphs named "filegraph/tbox" are ontology.
2. filegraph graphs are named with the name of the file they were loaded from.
3. filegraph files may be in several formats. You will see graphs loaded from files with type n3, owl and rdf.
4. The content in some of the filegraphs may repeat content found in other filegraphs. This does not impact the application.
5. Data you load by placing a file in filegraph/abox will appear as a result of the graph listing query above. See grid.n3 below for an example. grid.n3 is not distributed with VIVO.

10.5 Ontology Reference

- Overview (see page 310)
- Reference Materials (see page 310)
- Issue Tracking (see page 311)

10.5.1 Overview

VIVO uses a collection of ontologies to represent scholarship. The Integrated Semantic Framework ontology modules for VIVO (the VIVO-ISF ontology) provide a set of types (classes) and relationships (properties) to represent researchers and the full context in which they work. Content in any local VIVO installation may be maintained manually, brought into VIVO in automated ways from local systems of record, such as HR, grants, course, and faculty activity databases, or from database providers such as publication aggregators and funding agencies. Additional ontologies provide context and meaning for attributes and entities defined in VIVO-ISF.

VIVO-ISF is maintained by OpenRIF[165]. Other ontologies used in VIVO are maintained by the W3C[166] and other groups.

10.5.2 Reference Materials

- Source ontologies for VIVO (see page 311)

165 https://github.com/openrif
166 https://www.w3.org/standards/semanticweb/ontology

10.5.3 Issue Tracking

Improvements to the ontologies used in VIVO are treated like all other feature requests and are tracked in the VIVO JIRA issue tracker[167]. Issues involving ontological development in VIVO-ISF are replicated in the OpenRIF GitHub issue tracker[168].

10.5.4 Source ontologies for VIVO

10.5.4.1 Background

Source ontologies may be imported in their entirety or included selectively through the MIREOT[169] approach – minimum information to reference an external ontology term – used when importing an entire ontology would include unnecessary classes, properties, or axioms.

Imports vs. modules

The Integrated Semantic Framework is maintained in a file repository on GitHub [full ISF][170] [VIVO-ISF][171]) that reflects the source ontologies while creating distribution modules grouping classes and properties more by function, such as grants, agents, education, etc.

10.5.4.2 Ontologies Integrated into the Integrated Semantic Framework

The ISF ontology leverages the following ontologies in a unified, semantic structure:

- VIVO – http://vivoweb.org/ontology/core
- eagle-i Resource Ontology (ERO) – http://code.google.com/p/eagle-i
- Basic Formal Ontology (BFO) – http://www.ifomis.org/bfo
- Bibliographic Ontology (BIBO) – http://code.google.com/p/bibotools
- Cell Ontology (CL) – http://cellontology.org/?q=download
- Event Ontology – http://motools.sourceforge.net/event/event.html
- Friend of a Friend (FOAF) – http://www.foaf-project.org/
- Gene Ontology (GO) – http://geneontology.sourceforge.net/#code
- Geopolitical.owl[172], from the U.N. Food and Agriculture Organization
- Information Artifact Ontology (IAO) – http://code.google.com/p/information-artifact-ontology/

167 https://jira.duraspace.org/projects/VIVO/summary
168 https://github.com/openrif/community/issues
169 http://obi-ontology.org/page/MIREOT
170 https://github.com/SEssaid/connect-isf
171 https://github.com/vivo-isf
172 http://www.fao.org/countryprofiles/geopol_v10/ontologies/geopolitical.owl.html

- Ontology for Biomedical Investigations (OBI) – http://obi.sourceforge.net/ontologyInformation/
- Ontology of Clinical Research (OCRe) – http://code.google.com/p/ontology-of-clinical-research/
- Reagent Ontology (ReO) – http://code.google.com/p/reagent-ontology/
- Relations Ontology (RO) – http://obofoundry.org/ro/
- Software Ontology (SWO) – http://theswo.sourceforge.net/
- Sequence Ontology (SO) – http://www.sequenceontology.org/
- SKOS (Simple Knowledge Organization System) – http://www.w3.org/2004/02/skos/
- Uberon (Uber anatomy ontology) – http://obo.svn.sourceforge.net/viewvc/obo/uberon/releases/
- vCard – http://www.w3.org/TR/vcard-rdf/

10.5.5 VIVO Classes

10.5.5.1

10.5.5.2 Overview

VIVO uses a large number of classes from several different ontologies to represent scholarship. See Source ontologies for VIVO (see page 311). The classes and their ontologies are shown in the figure below. You may have additional classes as a result of local extensions.

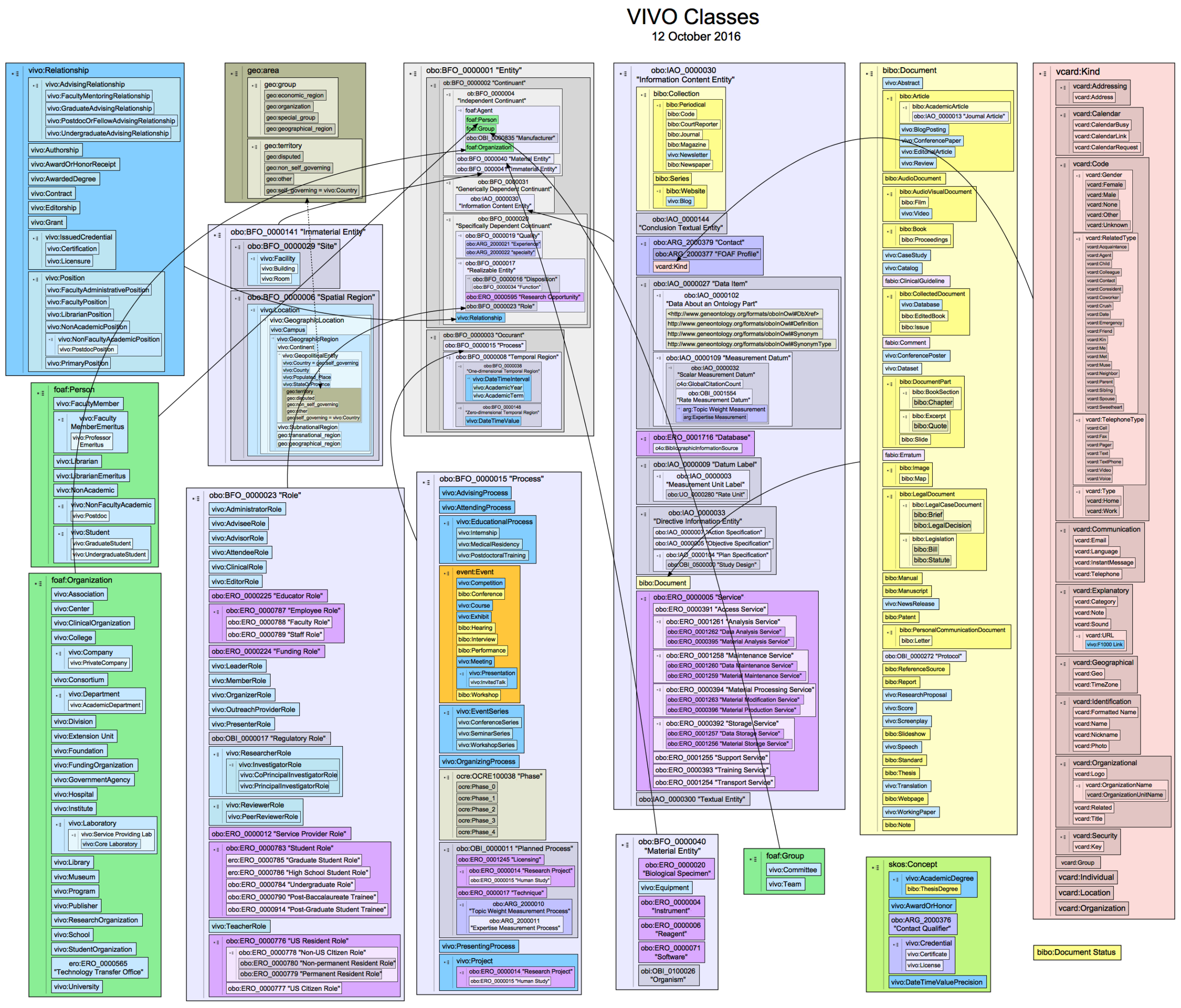
10.5.5.3 Finding the Classes in your VIVO

To find the classes in your VIVO, you can use the SPARQL query below.

```
SELECT ?s ?label
WHERE
{
   ?s a owl:Class .
   FILTER(regex(?s, "http"))
   ?s rdfs:label ?label .
}
ORDER BY ?s
```

10.5.5.4 VIVO Classes

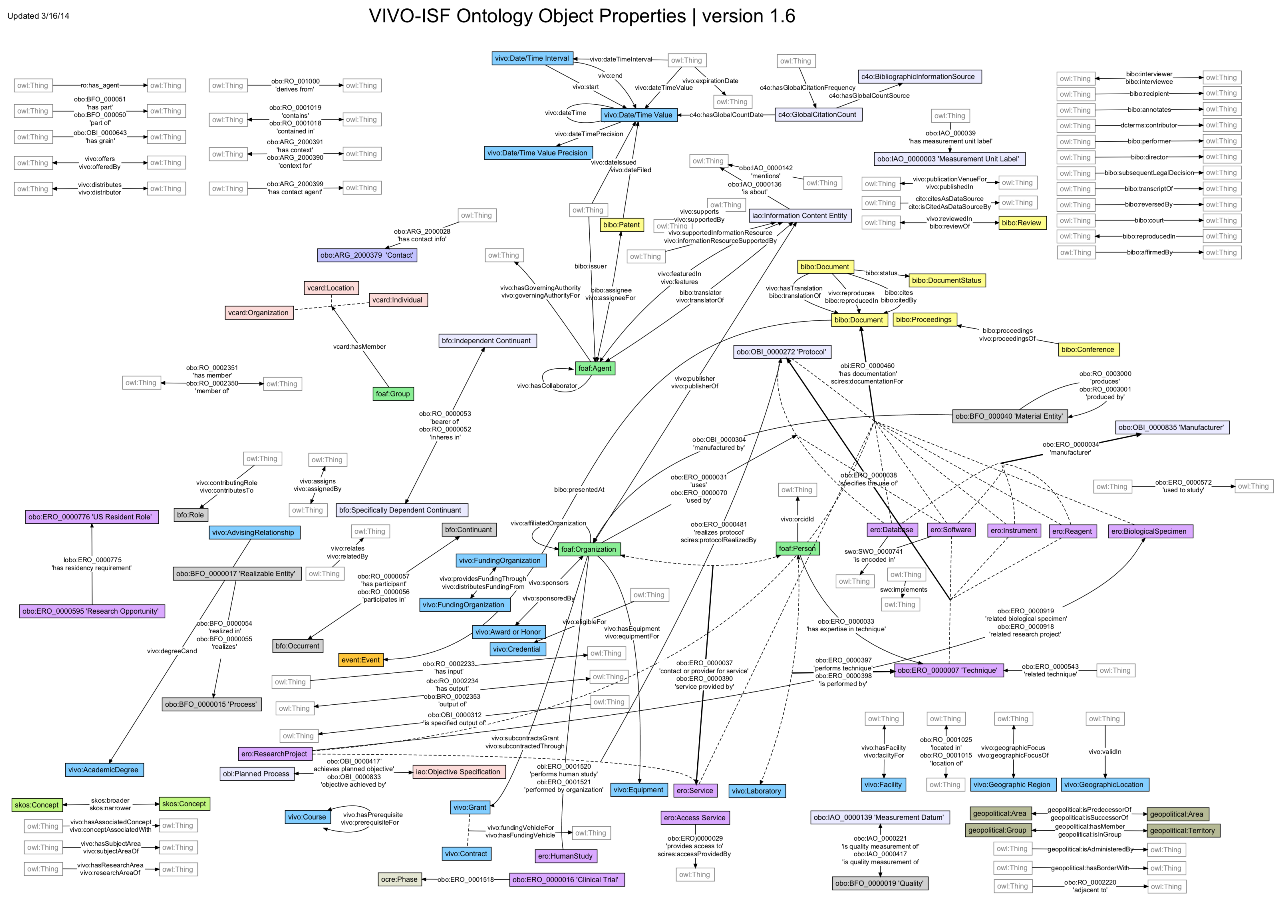
All classes delivered with VIVO should be included in the diagram below. Classes in the respective ontologies, but not delivered in VIVO, are not included. For figures related to the ontologies on which VIVO is based, see the corresponding ontology projects.



10.5.6 Ontology Overview : Object Properties

The diagram reflects a catalog of available object properties more than a diagram of how they may typically be populated in a VIVO instance.

Faux properties are managed through the application ontology and provide contextual labeling for commonly used properties when they are used in the context of different domain and range classes.



10.5.7 Ontology Diagrams

These diagrams shows the relationships between entities in VIVO. Diagrams for the primary entities of scholarship such as people and publications have diagrams centered on the primary entity.

Diagrams are drawn using VUE (Visual Understanding of the Environment), open source software from Tufts University. You can learn more about VUE at the VUE website[173]. VUE versions of each diagram are attached to the corresponding ontology diagram page.

These diagrams focus on the entity of interest. Related entities are not shown in complete detail.

These diagrams focus on common attributes and relationships. They are not comprehensive. Additional attributes and relationships are available. See the VIVO interface, use SPARQL queries, and examine the ontology to discover additional attributes and relationships.

- Organization Model (see page 315)
- Concept Model (see page 316)
- DateTimeValue and DateTimeInterval Models (see page 318)
- Journal Model (see page 320)
- Person Model (see page 321)
- Teaching Model (see page 323)
- Publication Model (see page 324)
- Grant Model (see page 325)

173 http://vue.tufts.edu

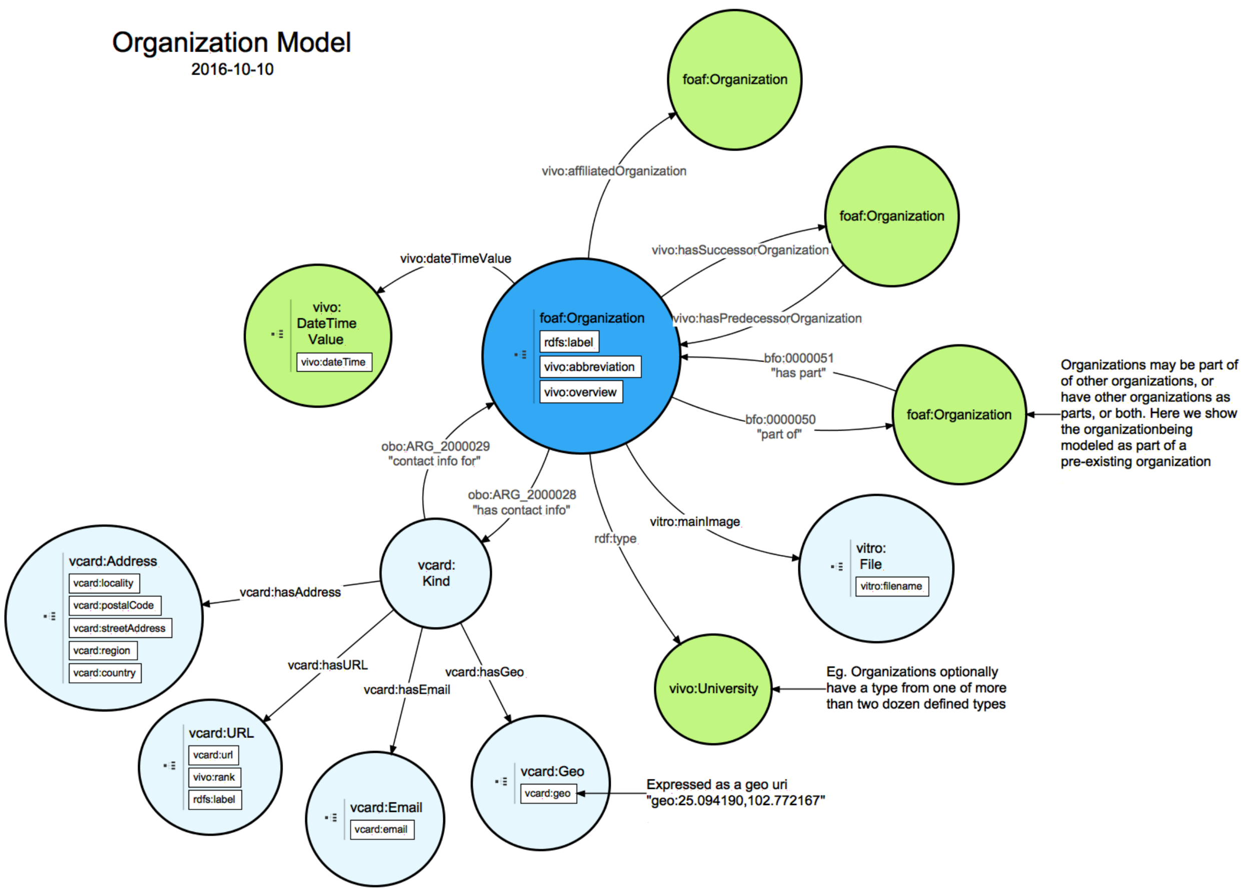
10.5.7.1 Organization Model

Notes

1. Organizations in VIVO are entities with `rdf:type foaf:Organization`. `vivo:overview` is used to provide a text description of the organization typically displayed on its profile page. A vcard is used to record contact information, URLs and geolocation.
2. VIVO provides a controlled vocabulary of organization types as `rdfs:subClassOf foaf:Organization.` To create a list of the available organization types, use the SPARQL query below:

```
SELECT ?s
WHERE
{
    ?s rdfs:subClassOf foaf:Organization .
}
```

3. Organizations may have relationships to other organizations. The "part of" relationship describes an organization as part of another in a hierarchical sense. For example, the History Department may be part of a College of Liberal Arts. The "successor" relationship describes an organization which no longer exists, and for which a successor organization now exists. The "affiliatedOrganization" organization describes an organization affiliated with the primary organization. The relationship is not symmetric, that is, the inverse is not inferred by the Inferencer. Assert the reverse affiliation as needed.
4. Many other attributes and relationships are available for organizations. The model shown here is typical for VIVO implementations.



Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.2 Concept Model

Notes

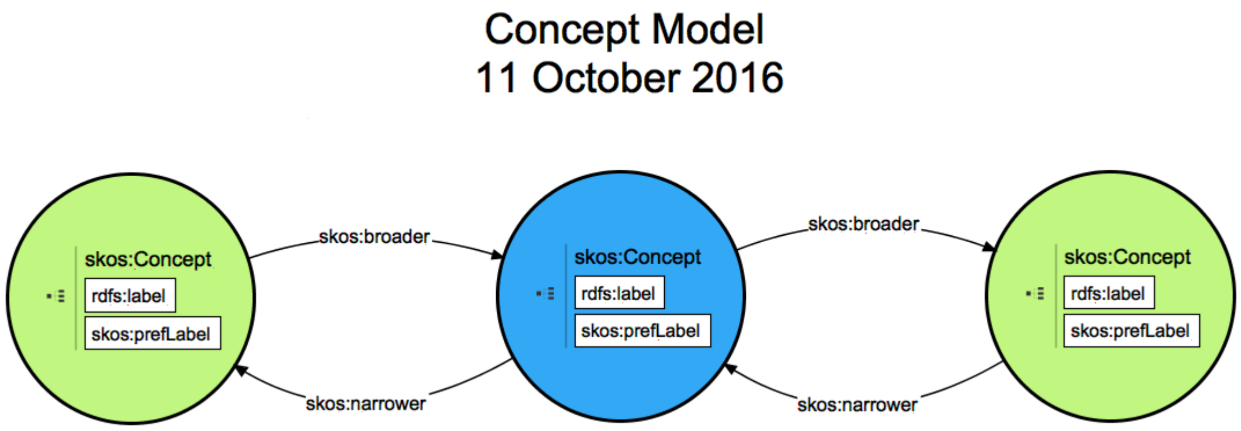
1. Concepts in VIVO are modeled using the SKOS (Simple Knowledge Organization System) ontology. SKOS is quite simple, and is a good place to start for those learning about ontologies, and how VIVO uses ontologies to represent information as triples in RDF. See The SKOS Primer[174], a readable introduction to SKOS and how it is represented in RDF.

174 https://www.w3.org/TR/skos-primer/

2. A concept is typically represented in VIVO as two triples, one declaring the URI of the concept as a skos:Concept, and one providing a text label for the concept. A third triple may use the skos:prefLabel to repeat the text label for those applications expecting the concept to have a preferred label. The triples might look like those below:

```
<http://vivo.myschool.edu> rdf:type skos:Concept .
<http://vivo.myschool.edu> rdfs:label "Molecular Biology"^^@en .
<http://vivo.myschool.edu> skos:prefLabel "Molecular Biology"^^@en .
```

3. Concepts are used throughout VIVO to indicate research and subject areas for people and other entities.



Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

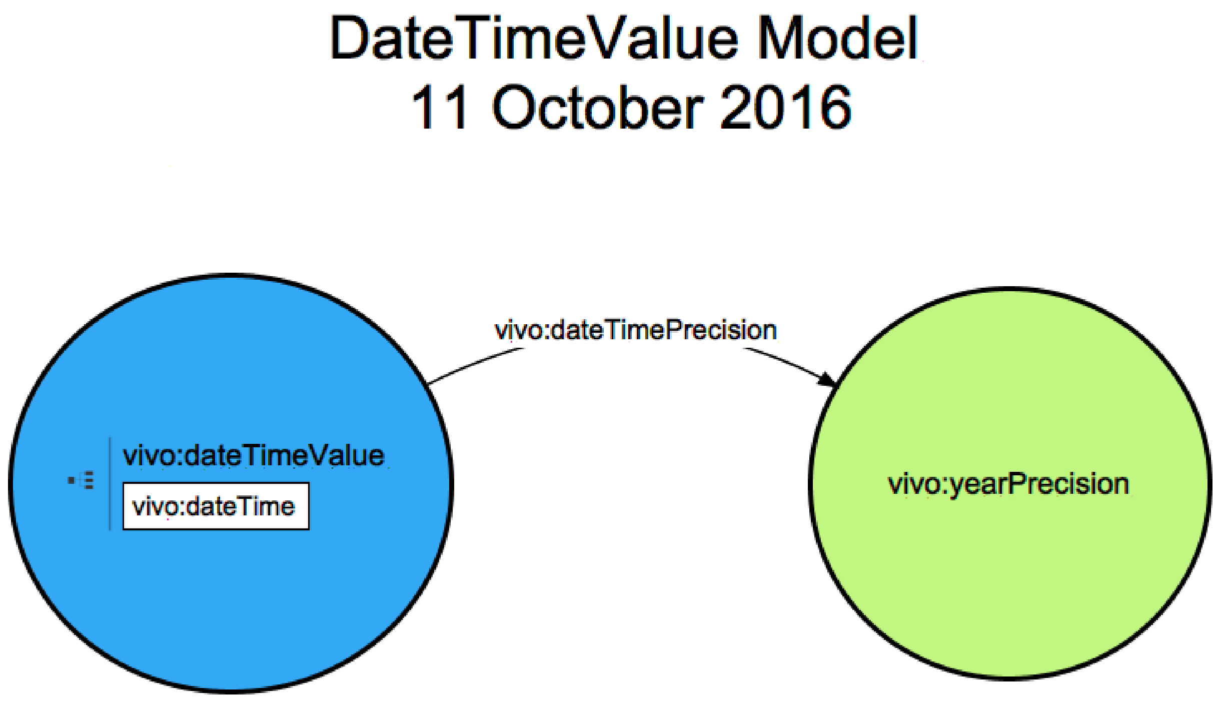
10.5.7.3 DateTimeValue and DateTimeInterval Models

Notes

1. VIVO uses DateTimeValue and DateTimeInterval to model dates and datetimes. These are objects, not literal values. The object models are simple (see below). VIVO DateTimeValue supports the concept of a precision, which indicates whether a particular DateTimeValue is accurate to the day, or perhaps only to the month, or perhaps only to the year. Precision is an important idea – publication dates, for example, are often known only to year precision, and sometimes to year and month.
2. The model indicates that creating a DateTimeValue requires three triples – one to specify the type, one to specify the literal value of the datetime, and one to indicate the precision.

```
<http://vivo.myschool.edu/individual/n123> rdf:type vivo:DateTimeValue .
<http://vivo.myschool.edu/individual/n123> vivo:dateTime "2010-11-12T12:00:00"^^xsd:datetime .
<http://vivo.myschool.edu/individual/n123> vivo:dateTimePrecision vivo:yearPrecision .
```

3. VIVO provides the precisions shown below:
    - `<http://vivoweb.org/ontology/core#yearMonthDayTimePrecision>`
    - `<http://vivoweb.org/ontology/core#yearMonthPrecision>`
    - `<http://vivoweb.org/ontology/core#yearPrecision>`
    - `<http://vivoweb.org/ontology/core#yearMonthDayPrecision>`



Ontology Diagram Legend

Dark blue – the entity being modeled

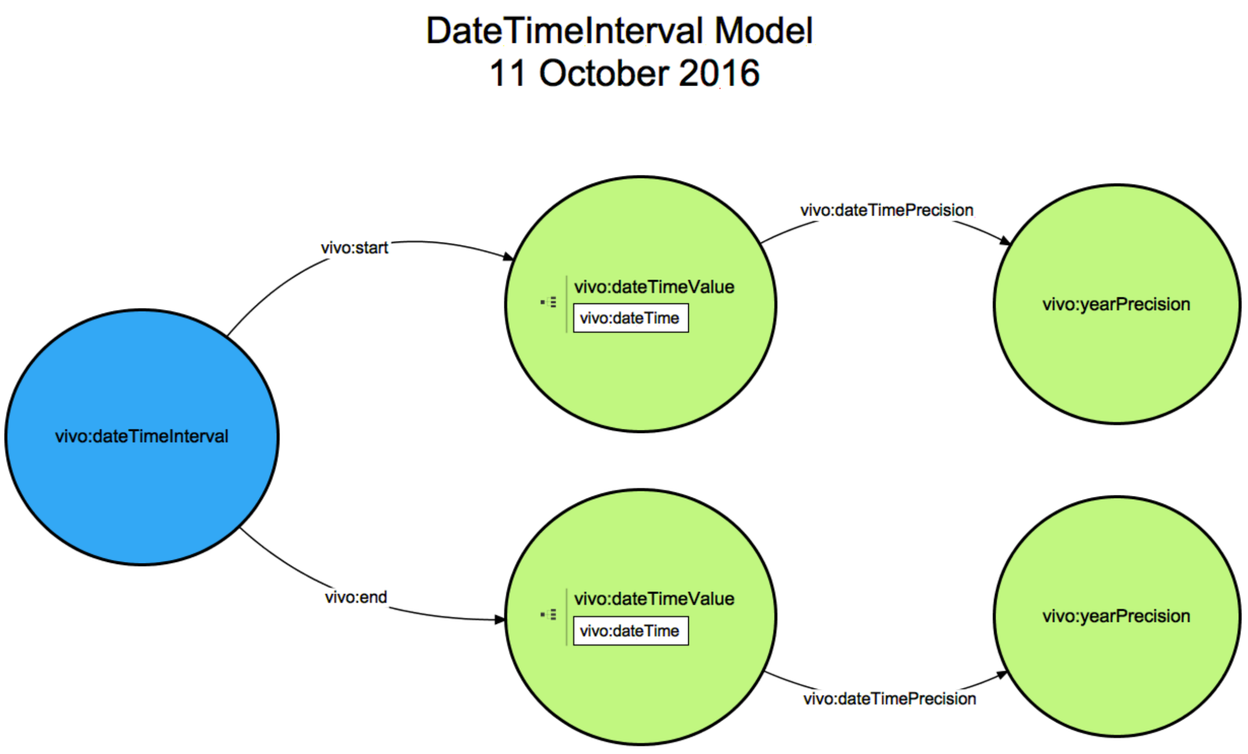
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

DateTimeInterval

The DateTimeInterval is an entity that references one or two DateTimeValues. Either reference could be missing. An interval might have a start date and no end date, for example. To create a DateTimeValue with a start and end takes the statements below, where the start and end objects exist and have the URI as shown.

```
<http://vivo.mydomain.edu/individual/n456> rdf:type vivo:DateTimeInterval .
<http://vivo.mydomain.edu/individual/n456> vivo:start <http://vivo.mydomain.edu/individual/n123> .
<http://vivo.mydomain.edu/individual/n456> vivo:end <http://vivo.mydomain.edu/individual/n124> .
```



Ontology Diagram Legend

Dark blue – the entity being modeled

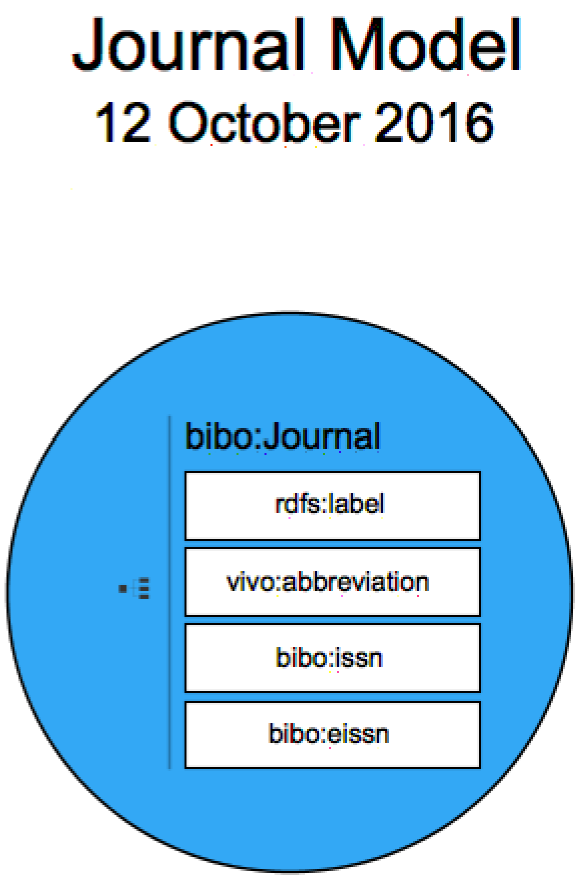
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.4 Journal Model

Notes

1. A journal in VIVO is an entity of type `bibo:Journal`.
2. The journal has a series of attributes, all are literals. Journal entities are quite simple.



Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

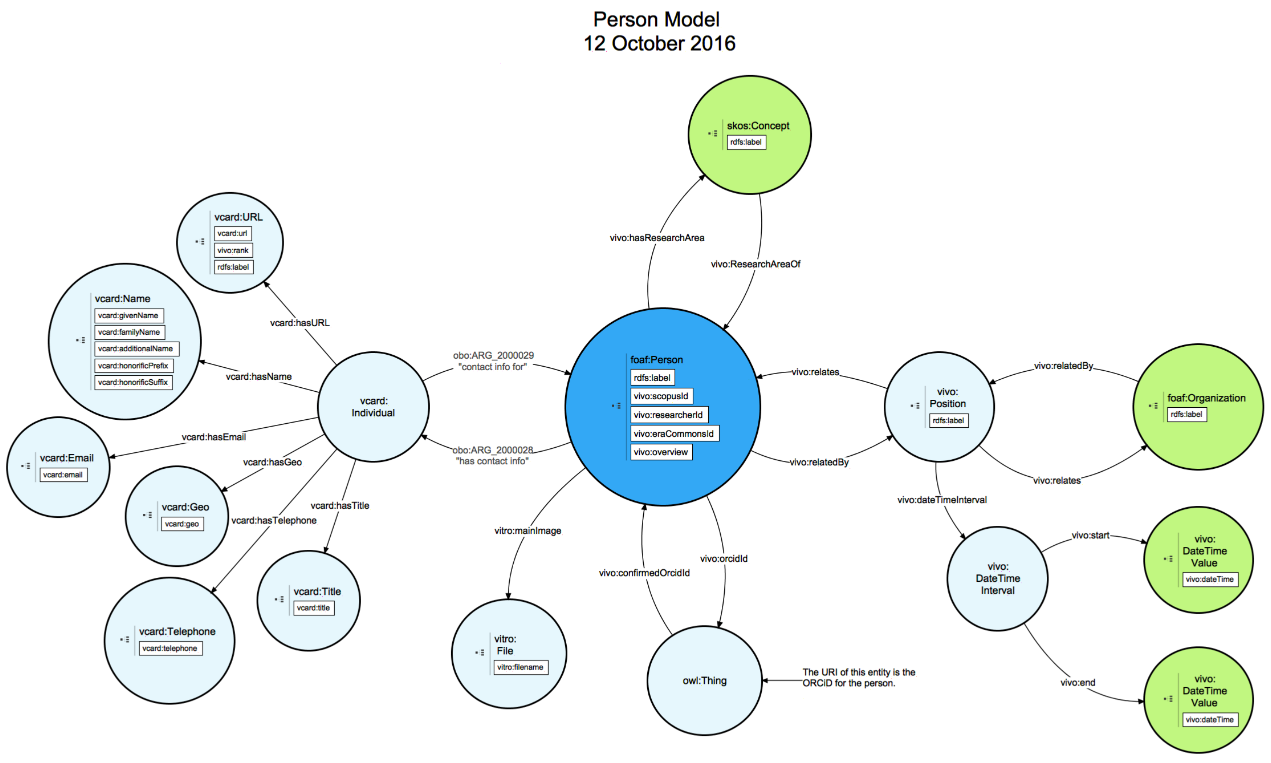
Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.5 Person Model

Notes

1. The Person entity has a large collection of literal values. The most common are shown in the Person entity below.
2. People are associated with research areas represented by skos:Concept entities. See Concept Model (see page 316).

3. Positions are relationships between a person and an organization. See Organization Model (see page 315) for detail regarding representation of organizations. The position may have an associated dateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 318) for details regarding the representation of these entities.
4. The ORCiD of a person is represented as an entity. The URI of the entity is the ORCiD of the person. See Managing Person Identifiers (see page 83) for additional details.
5. The photo of a person is stored in a file and referenced using triples associated with the person. See Image storage (see page 384)
6. Vcards are used to store contact information, name parts, URLs, and geolocation. The general pattern is that a person has a vcard, the vcard has a an intermediate related to the type of information to be stored, and the intermediate has references to literal values.
7. Additional details regarding the person – teaching (see page 323), grants (see page 325), publications (see page 324), advising (see page 333), educational training (see page 331), awards (see page 335), memberships (see page 336) – are shown in their respective models.



Ontology Diagram Legend

Dark blue – the entity being modeled

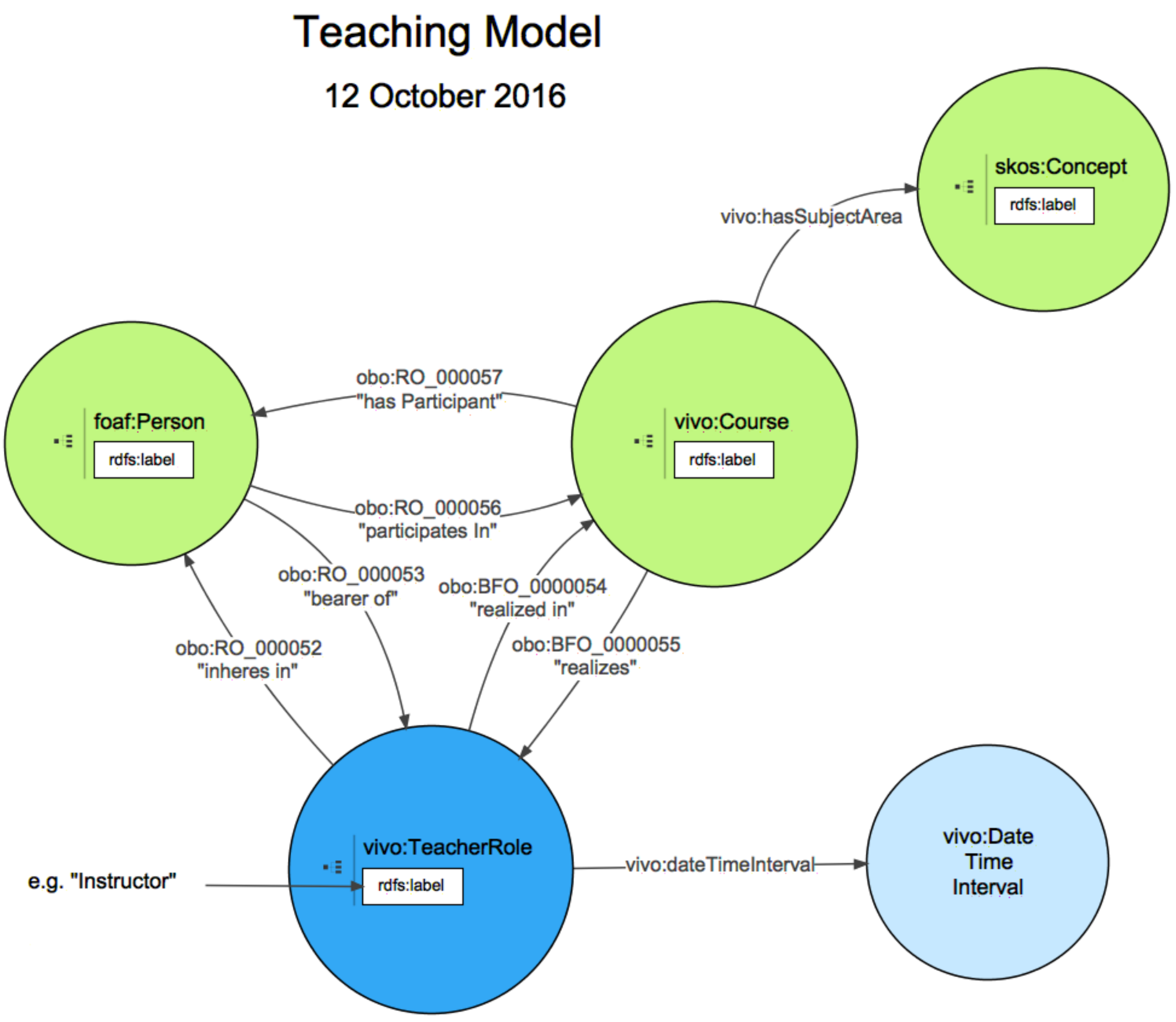
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.6 Teaching Model

Notes

1. Teaching is represented as a time limited role associating a person with a course.
2. The course may have optional concepts indicating subject area(s). See Concept Model (see page 316) for details.
3. The role typically has a DateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 318) for details.
4. The Role may have a label such as "Instructor" or "Team Lead" or other to further indicate the nature of the instructor's role.
5. The instructor is a person. See Person Model (see page 321) for details.
6. Any number of instructors may each have a role in a course. Each has their own role.



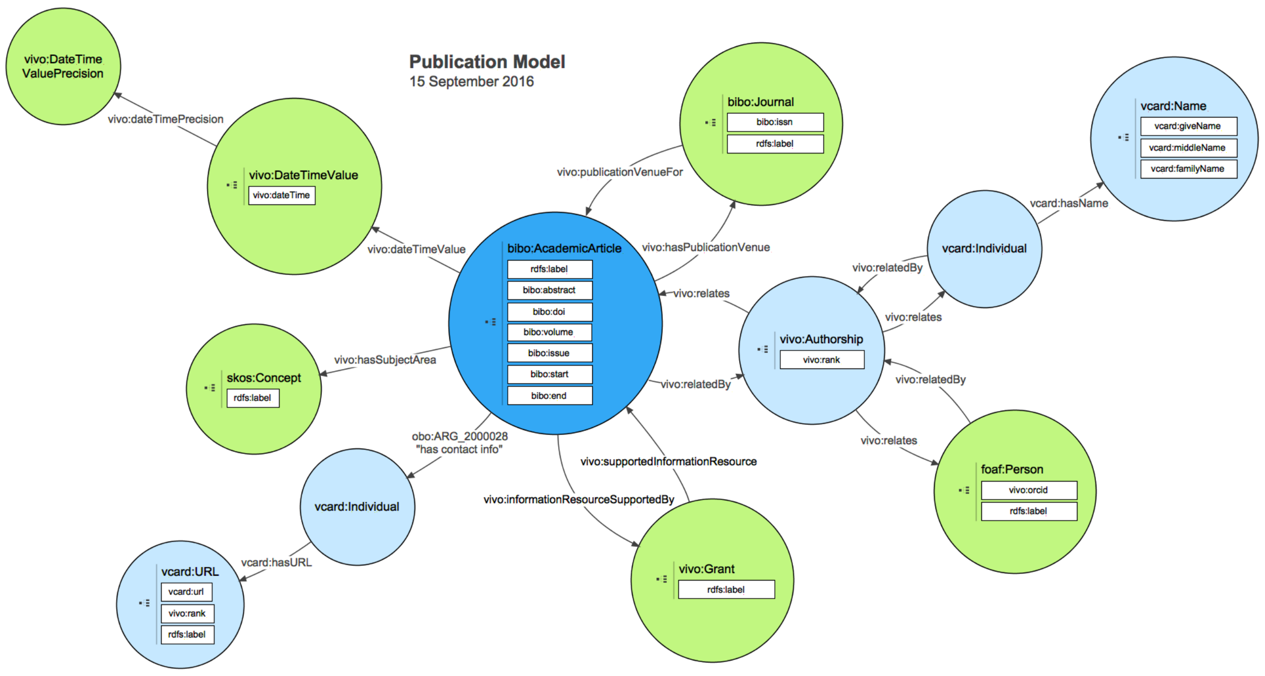
Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.7 Publication Model



Ontology Diagram Legend

Dark blue – the entity being modeled

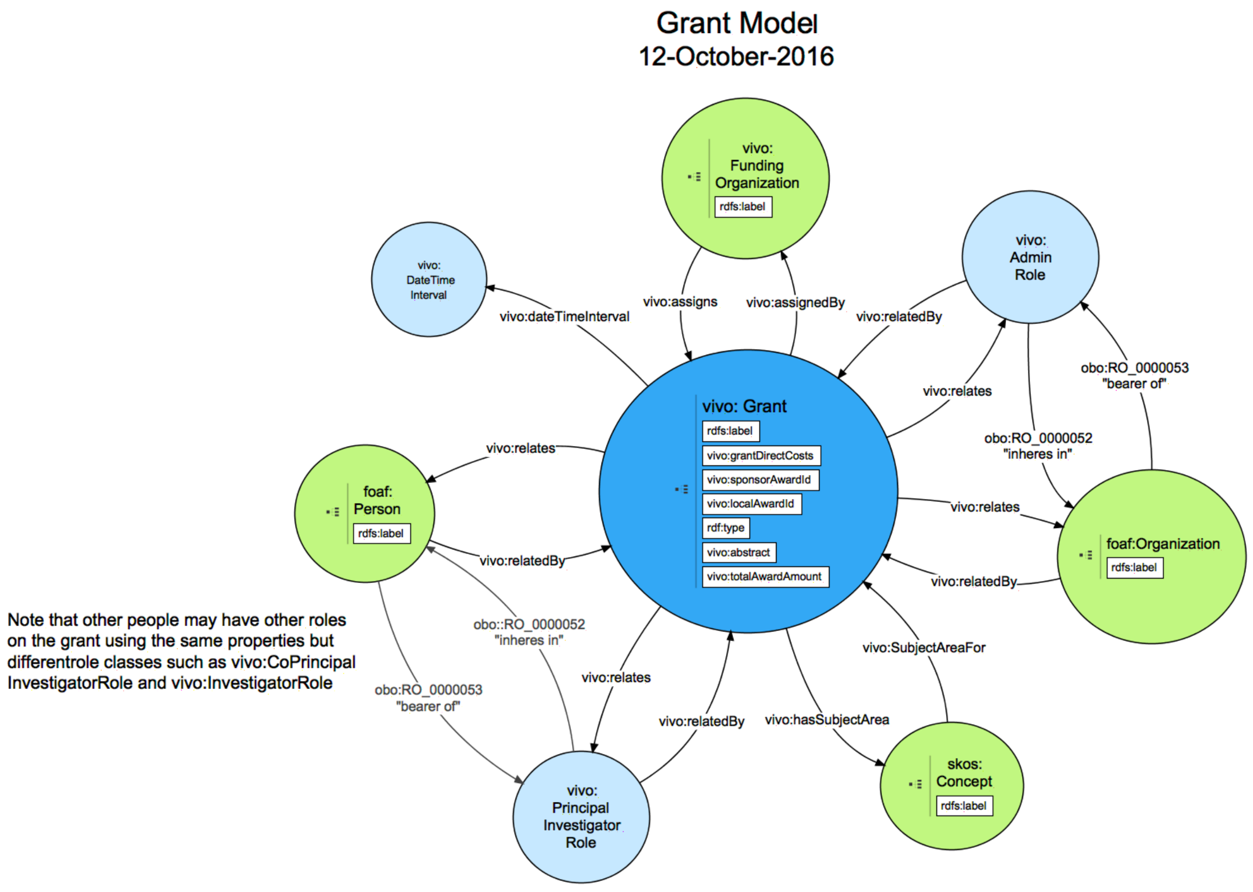
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.8 Grant Model

Notes

1. The Grant entity has attributes to record funding amounts, label, abstract, loca award ID (the ID as assigned by the administering organization), and sponsorAwardID (the ID of the grant as assigned by the funding organization)
2. The FundingOrganization is related to the grant through vivo:assigns and vivo:assignedBy. For additional details regarding modeling organizations, see Organization Model (see page 315)
3. An organization, often an academic department, typically has a role in administering the grant. This is modeling using an AdminRole which associated the grant, the role and the organization administering the grant.
4. The grant may have one or more subject areas, represented as skos:Concept. See Concept Model (see page 316).
5. One or more people will be associated with the grant through roles. There will be one role for each person. The role associates the person with the grant. For additional detail regarding the modeling of people, see Person Model (see page 321)
6. The grant has an associate dateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 318)



Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

Worked ontology example using Person, Role, and Project instead of Grant

A question has come up in the VIVO community about using Project instead of Grants – when a VIVO institution may not receive grants but does want to track projects.

In the VIVO-ISF ontology, a Grant is a subclass of vivo:Relationship, since it represents the agreement between a funding organization and a receiving organization, with the investigator roles usually also specified.

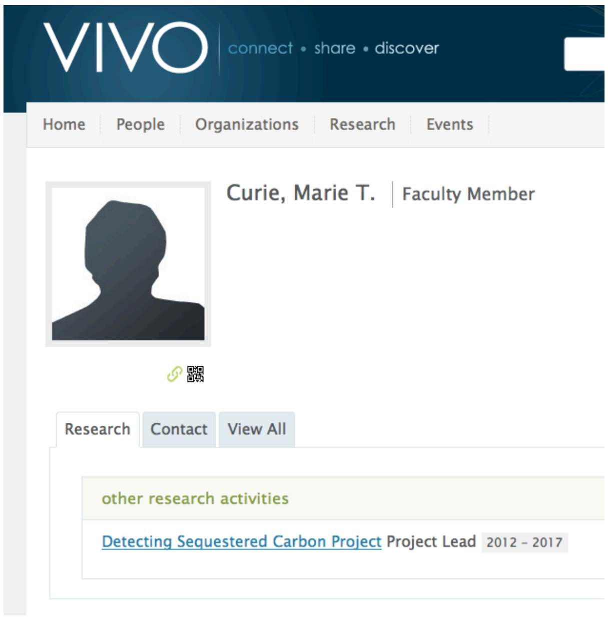
A Project, however, is a subclass of Project, which in turn is a subclass of bfo:Process. The project is the activity undertaken or the investigation, not just the agreement.

The properties used are therefore slightly different to connect a Person, Role, and Project vs. a Person, Role, and Grant, as indicated on the Grant Model (see page 325) page.

Example

We have a researcher, Marie Curie, who has the Project Lead role on a Project. In the VIVO front end display, there appears to be a direct relationship between the person and the project, and an inverse relationship in return. The role and date information appear as modifiers to the direct relationship, but are maintained through the ontology as an intermediate Role object bearing the title of the role ("Project Lead") and the date range.

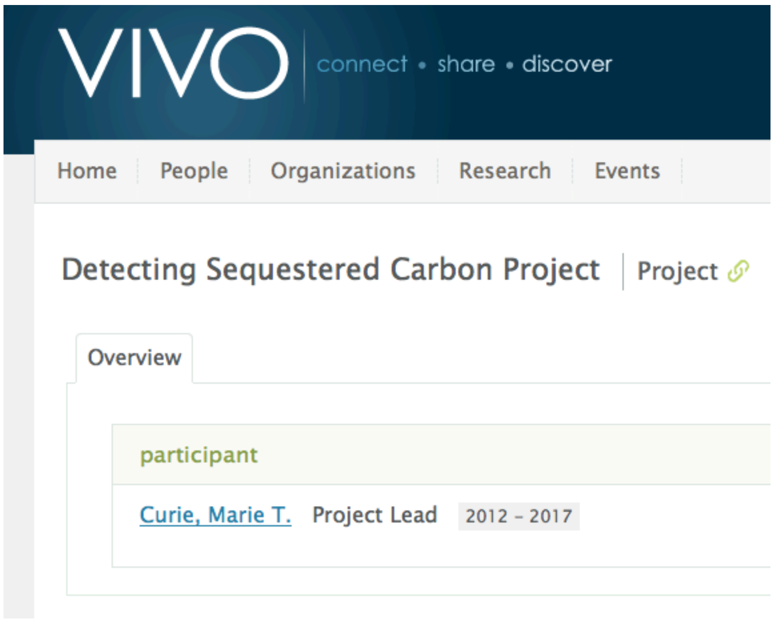
Public display view



Curie, Marie T. | Faculty Member

other research activities

Detecting Sequestered Carbon Project Project Lead 2012 - 2017



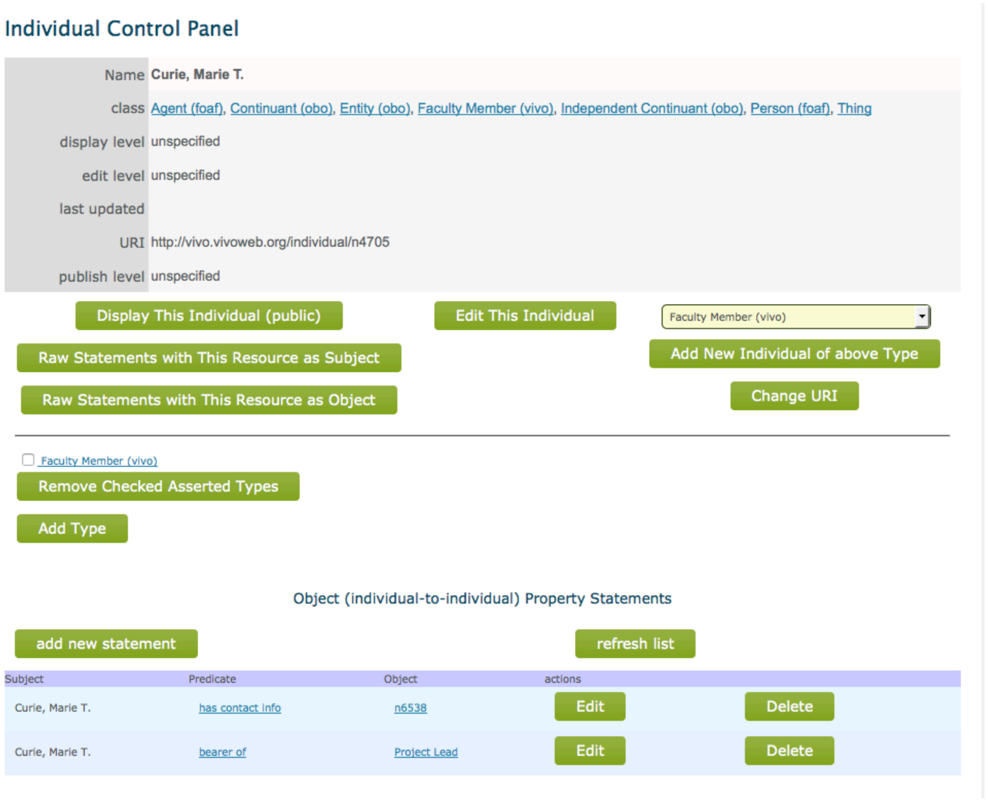
Detecting Sequestered Carbon Project | Project

participant

Curie, Marie T. Project Lead 2012 - 2017

Site admin view
This can be seen more clearly in the back-end editors view (when logged in with Site Admin privileges):

Note in the listing of object property statements at the bottom of the image that the Person has a "bearer of" relationship (http://purl.obolibrary.org/obo/RO_0000053) to the Role – and no direct relationship to the Project.


Individual Control Panel

Name	Curie, Marie T.
class	Agent (foaf), Continuant (obo), Entity (obo), Faculty Member (vivo), Independent Continuant (obo), Person (foaf), Thing
display level	unspecified
edit level	unspecified
last updated	
URI	http://vivo.vivoweb.org/individual/n4705
publish level	unspecified

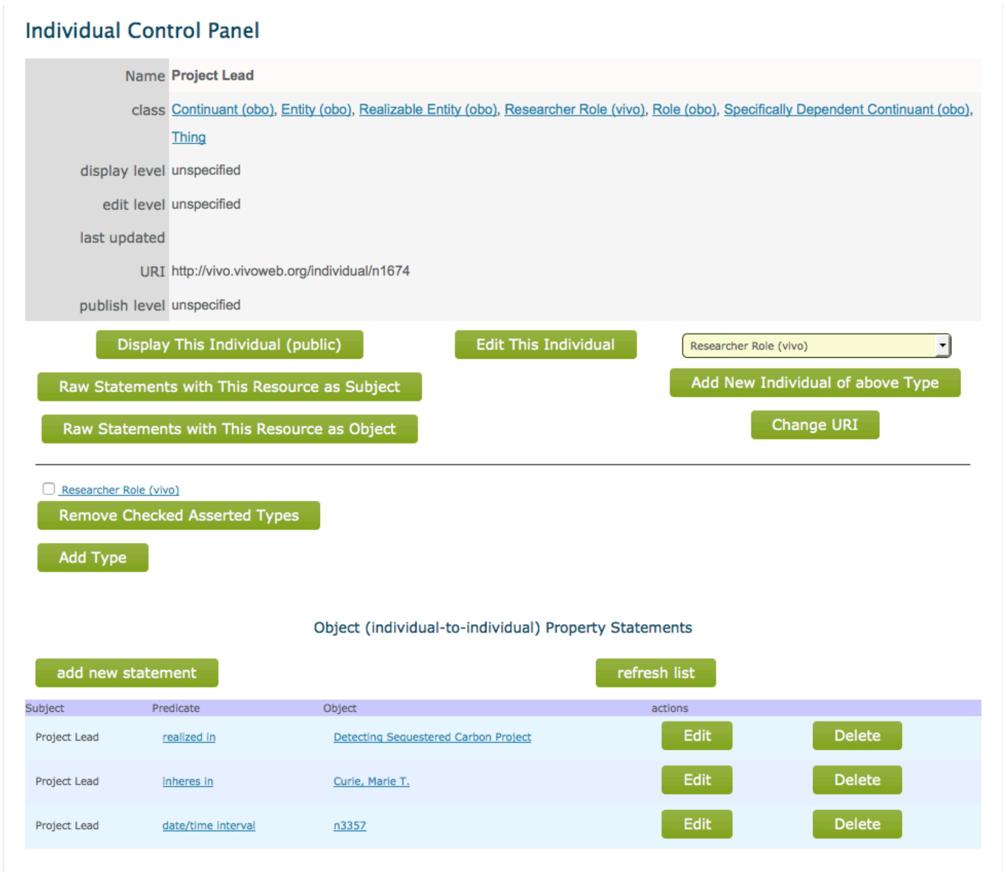
Display This Individual (public)
Edit This Individual
Faculty Member (vivo)
Raw Statements with This Resource as Subject
Add New Individual of above Type
Raw Statements with This Resource as Object
Change URI
Faculty Member (vivo)
Remove Checked Asserted Types
Add Type

Object (individual-to-individual) Property Statements

add new statement
refresh list

Subject	Predicate	Object	actions	
Curie, Marie T.	has contact info	n6538	Edit	Delete
Curie, Marie T.	bearer of	Project Lead	Edit	Delete

On the intermediate Role page, the relationships in both directions may be seen: the Role "inheres in" (http://purl.obolibrary.org/obo/RO_0000052) the Person and is "realized in" (http://purl.obolibrary.org/obo/BFO_0000054) the Project.



Individual Control Panel

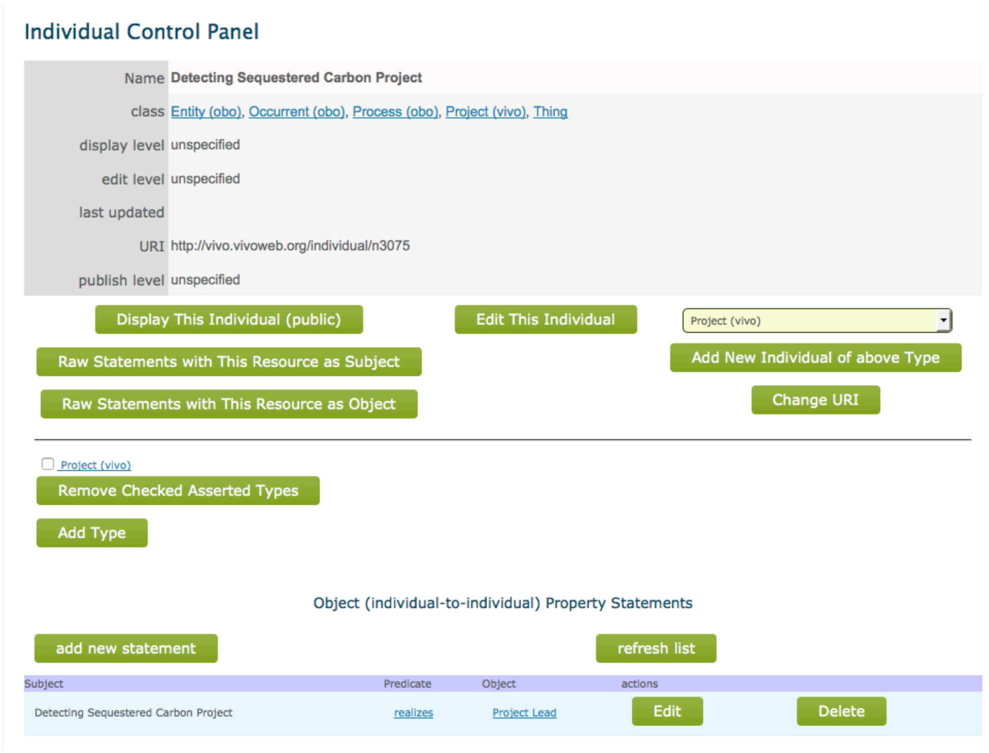
Name	Project Lead
class	Continuant (obo), Entity (obo), Realizable Entity (obo), Researcher Role (vivo), Role (obo), Specifically Dependent Continuant (obo), Thing
display level	unspecified
edit level	unspecified
last updated	
URI	http://vivo.vivoweb.org/individual/n1674
publish level	unspecified

Display This Individual (public)
Edit This Individual
Researcher Role (vivo)
Raw Statements with This Resource as Subject
Add New Individual of above Type
Raw Statements with This Resource as Object
Change URI
Researcher Role (vivo)
Remove Checked Asserted Types
Add Type

Object (individual-to-individual) Property Statements

Subject	Predicate	Object	actions	
Project Lead	realized in	Detecting Sequestered Carbon Project	Edit	Delete
Project Lead	inheres in	Curie, Marie T.	Edit	Delete
Project Lead	date/time interval	n3357	Edit	Delete

Finally, from the Project perspective, only the return (inverse) relationship to the Role is seen: the Project "realizes" (http://purl.obolibrary.org/obo/BFO_0000055) the Role.



Individual Control Panel

Name	Detecting Sequestered Carbon Project
class	Entity (obo), Occurrent (obo), Process (obo), Project (vivo), Thing
display level	unspecified
edit level	unspecified
last updated	
URI	http://vivo.vivoweb.org/individual/n3075
publish level	unspecified

Display This Individual (public)
Edit This Individual
Project (vivo)
Raw Statements with This Resource as Subject
Add New Individual of above Type
Raw Statements with This Resource as Object
Change URI
Project (vivo)
Remove Checked Asserted Types
Add Type

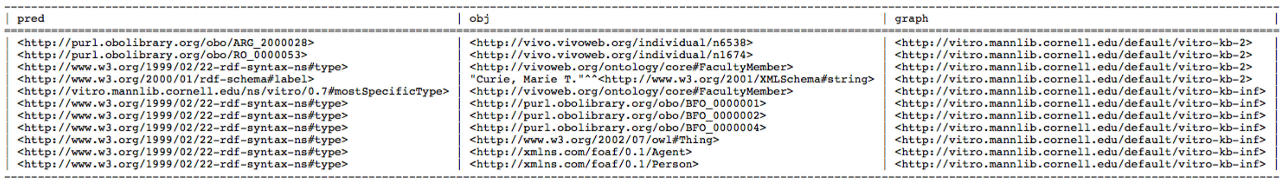
Object (individual-to-individual) Property Statements

add new statement
refresh list

Subject	Predicate	Object	actions	
Detecting Sequestered Carbon Project	realizes	Project Lead	Edit	Delete

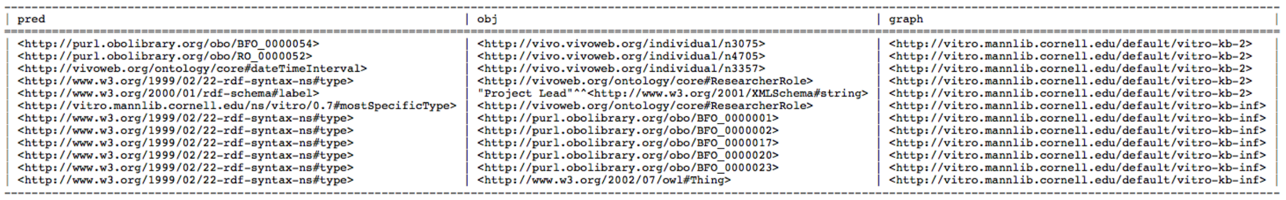
The triples underneath

For the Person (http://vivo.vivoweb.org/individual/n4705):



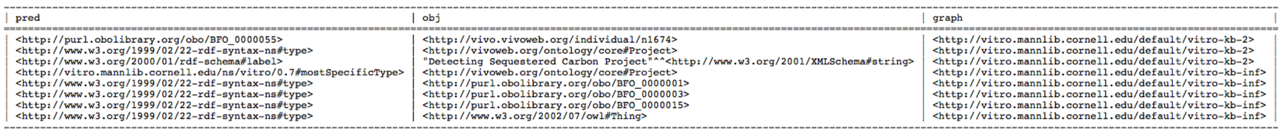
```
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
| pred                                                        | obj                                                          | graph                                            |
===================================================================================================================================================================
| <http://purl.obolibrary.org/obo/ARG_2000028>                | <http://vivo.vivoweb.org/individual/n6538>                   | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://purl.obolibrary.org/obo/RO_0000053>                 | <http://vivo.vivoweb.org/individual/n1674>                   | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://vivoweb.org/ontology/core#FacultyMember>             | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/2000/01/rdf-schema#label>                | "Curie, Marie T."^^<http://www.w3.org/2001/XMLSchema#string> | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#mostSpecificType> | <http://vivoweb.org/ontology/core#FacultyMember>        | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000001>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000002>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000004>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://www.w3.org/2002/07/owl#Thing>                        | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://xmlns.com/foaf/0.1/Agent>                            | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://xmlns.com/foaf/0.1/Person>                           | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

For the Role (http://vivo.vivoweb.org/individual/n1674):



```
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
| pred                                                        | obj                                                          | graph                                            |
===================================================================================================================================================================
| <http://purl.obolibrary.org/obo/BFO_0000054>                | <http://vivo.vivoweb.org/individual/n3075>                   | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://purl.obolibrary.org/obo/RO_0000052>                 | <http://vivo.vivoweb.org/individual/n4705>                   | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://vivoweb.org/ontology/core#dateTimeInterval>         | <http://vivo.vivoweb.org/individual/n3357>                   | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://vivoweb.org/ontology/core#ResearcherRole>            | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/2000/01/rdf-schema#label>                | "Project Lead"^^<http://www.w3.org/2001/XMLSchema#string>    | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#mostSpecificType> | <http://vivoweb.org/ontology/core#ResearcherRole>       | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000001>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000002>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000017>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000020>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000023>                 | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://www.w3.org/2002/07/owl#Thing>                        | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
-------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

And finally, for the Project (http://vivo.vivoweb.org/individual/n3075):



```
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
| pred                                                        | obj                                                                                  | graph                                            |
=======================================================================================================================================================================================
| <http://purl.obolibrary.org/obo/BFO_0000055>                | <http://vivo.vivoweb.org/individual/n1674>                                           | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://vivoweb.org/ontology/core#Project>                                           | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://www.w3.org/2000/01/rdf-schema#label>                | "Detecting Sequestered Carbon Project"^^<http://www.w3.org/2001/XMLSchema#string>    | <http://vitro.mannlib.cornell.edu/default/vitro-kb-2>   |
| <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#mostSpecificType> | <http://vivoweb.org/ontology/core#Project>                                      | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000001>                                         | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000003>                                         | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://purl.obolibrary.org/obo/BFO_0000015>                                         | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
| <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>           | <http://www.w3.org/2002/07/owl#Thing>                                                | <http://vitro.mannlib.cornell.edu/default/vitro-kb-inf> |
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
```

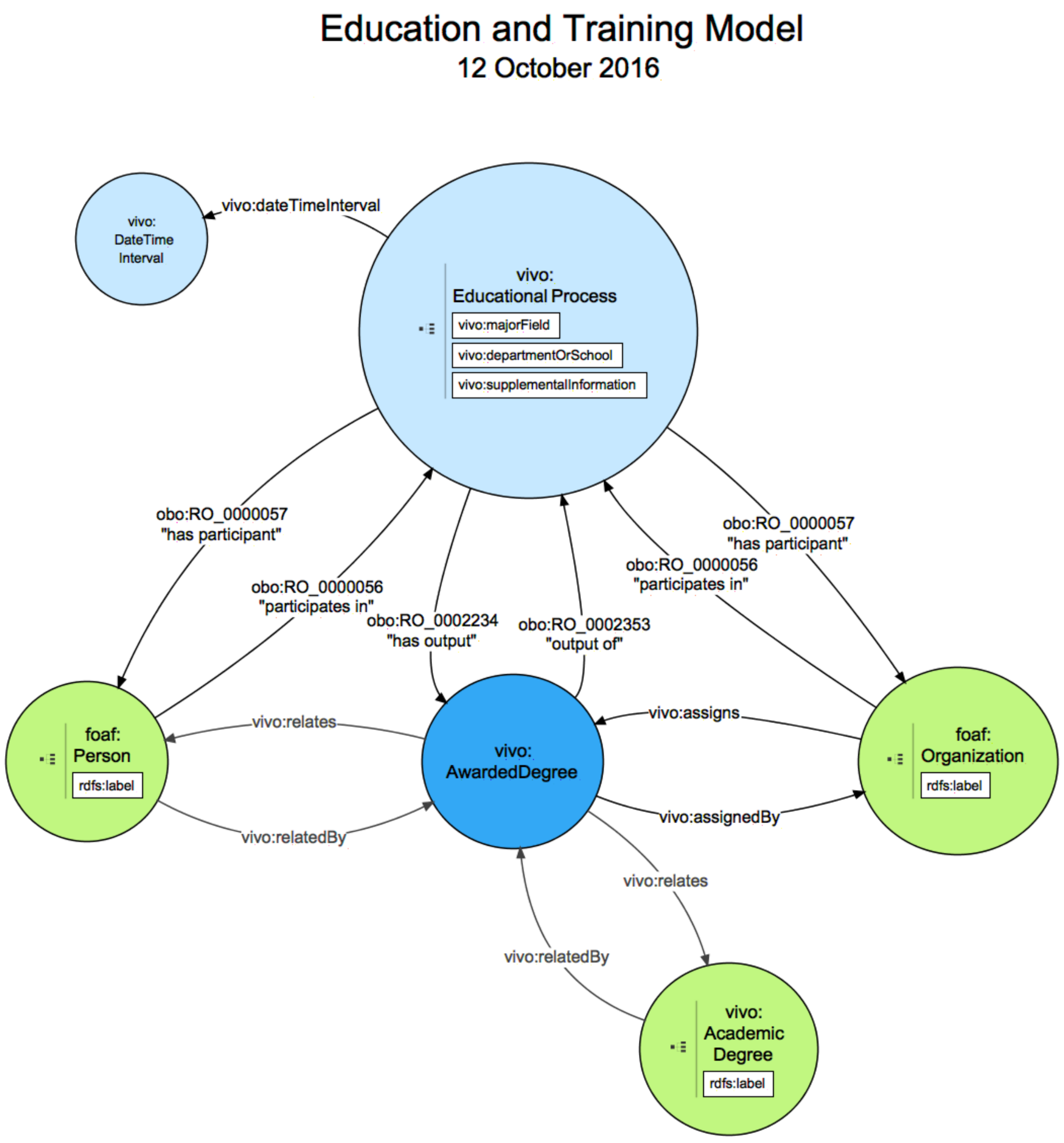
10.5.7.9 Education and Training Model

Notes

1. The entity of interest here is the AwardedDegree (dark blue in the center of the figure). The AwardedDegree is a relationship between a Person and an AcademicDegree. The AcademicDegree can be considered "abstract." The AwardedDegree is concrete – a person received the degree from a university at a particular time. VIVO provides a controlled vocabulary of AcademicDegrees. Note that the label for the degree is on the AcademicDegree.
2. The AwardedDegree has an associated EducationalProcess, which contains attributes of the AwardedDegree. The EducationalProcess has a DateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 318) for detail.
3. See Organization Model (see page 315) for details regarding the modeling of organizations
4. For a list of AcademicDegrees, use the SPARQL query below

```
SELECT ?s ?name
WHERE {
    ?s a vivo:AcademicDegree .
    ?s rdfs:label ?name .
}
ORDER BY ?name
```

5. See Person Model (see page 321) for details regarding the modeling of people



Ontology Diagram Legend

Dark blue – the entity being modeled

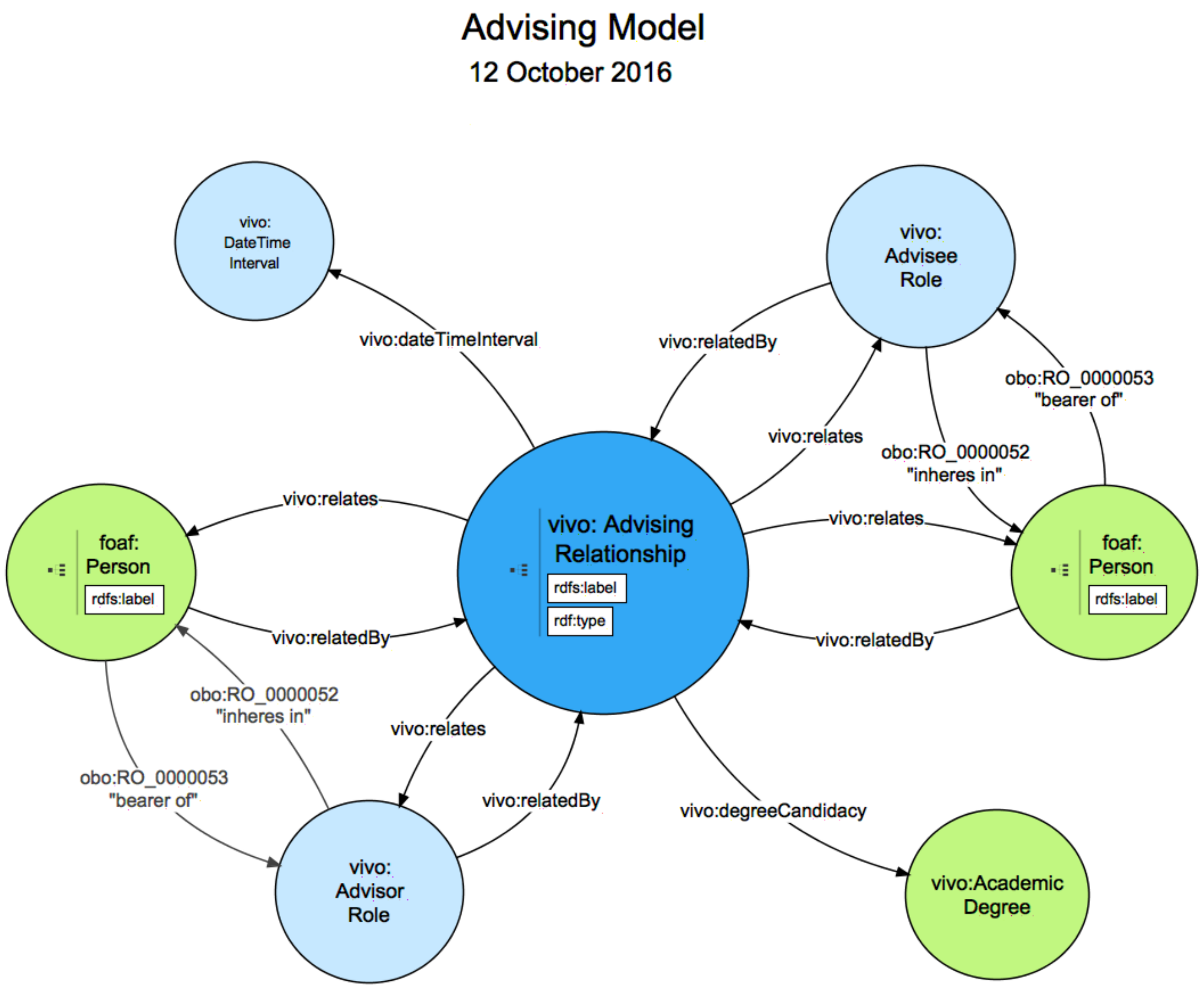
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.10 Advising Model

Notes

1. The label on the AdvisingRelationship is optional. VIVO constructs a label consisting of the Advisors label, the Advisee's label, some text representing the type of relationship
2. The AdvisingRelationship can optionally have one of the types below:
   `http://vivoweb.org/ontology/core#FacultyMentoringRelationship`
   `http://vivoweb.org/ontology/core#GaduateAdvisingRelationship`
   `http://vivoweb.org/ontology/core#UndergraduateAdvisingRelationship`
   `http://vivoweb.org/ontology/core#PostdocOrFellowAdvisingRelationship`
3. The AdviseeRole relates the AdvisingRelationship to the Advisee. This pattern is common for modeling roles and relationships.
4. The AcademicDegree is optional. It may be present for AdvisingRelationships leading to a degree.
5. The AdvisorRole relates the AdvisingRelationship to the Advisor. It uses the same pattern as the AdviseeRole.
6. The AdvisingRelationship may have an associated DateTimeInterval. See DateTimeValue and DateTimeInterval Models (see page 318) for details regarding modeling DateTimeIntervals.



Ontology Diagram Legend

Dark blue – the entity being modeled

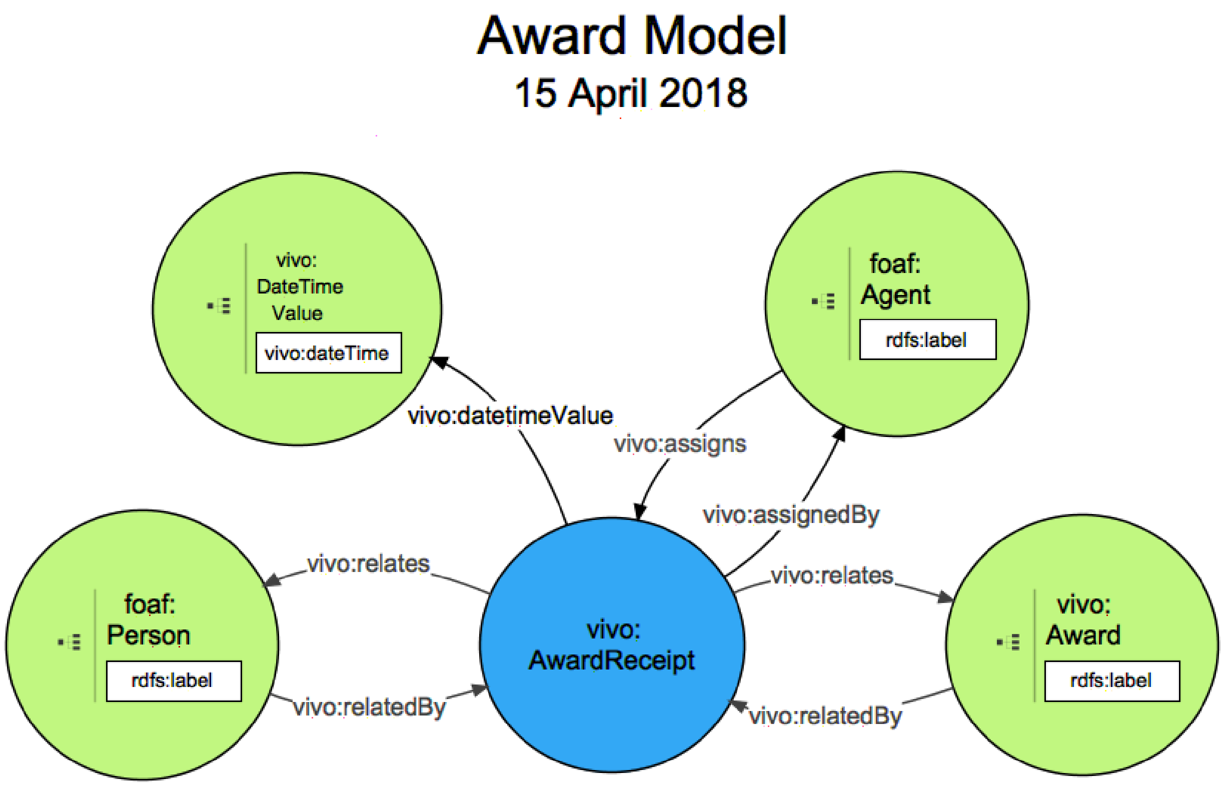
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.11 Award Model

Notes

1. The entity of interest here is the AwardReceipt. It is a relationship between a Person and an Award. The Award entity is generic, as in "The Nobel Prize in Physics." The AwardReceipt is specific, as in "Person x was awarded The Nobel Prize in Physics by the Royal Swedish Academy of Sciences on 10 October 2016"
2. The foaf:Agent is the entity making the award.



Ontology Diagram Legend

Dark blue – the entity being modeled

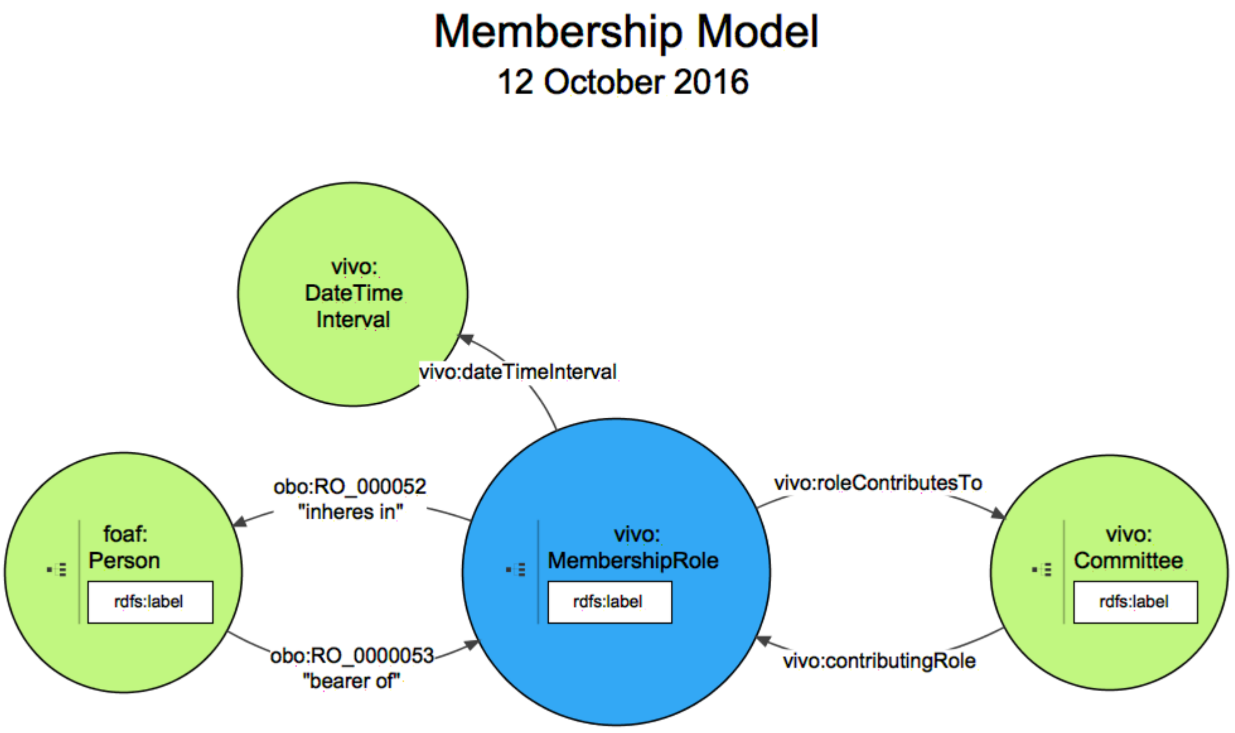
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.12 Membership Model

Notes

1. Membership is represented by using a MembershipRole to associate a person with an organization (a committee, or other organization).
2. The MembershipRole is associated with the organization using vivo:roleContributesTo and vivo:contributingRole
3. The MembershipRole has an optional label. The label is used to indicate the whether the person is "Chair" or "Member" or some other term that further describes the membership.



Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.13 Ontology Diagram Legend

Dark blue – the entity being modeled

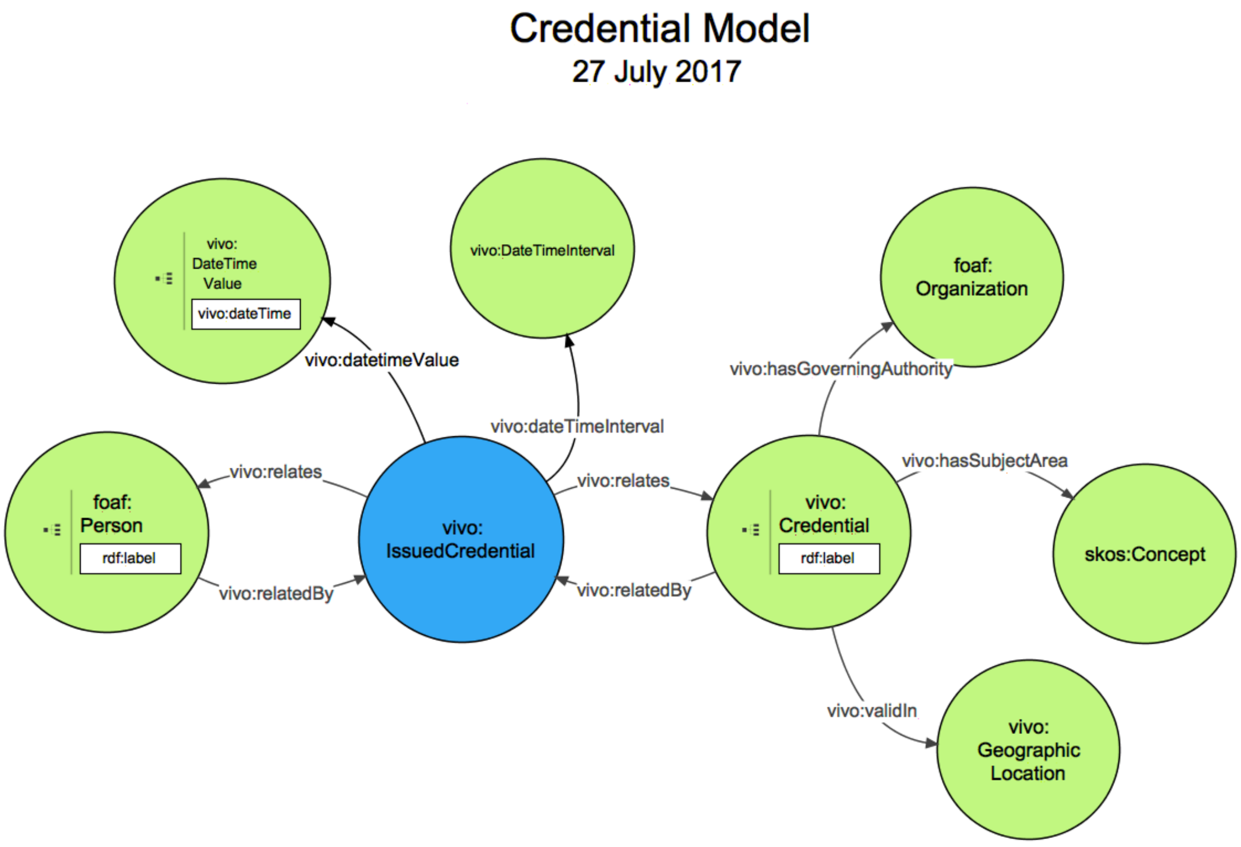
Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.7.14 Credential Model

Notes

1. The entity of interest here is the IssuedCredential. It is a relationship between a Person and a Credential. The Credential entity is generic, as in "Board Certified in Neurology." The IssuedCredential is specific, as in "Person x was credentialed as Board Certified in Neurology in 2017"
2. The Credential may have a type of License or Certificate.
3. This model is similar to the Award Model.



Ontology Diagram Legend

Dark blue – the entity being modeled

Light blue – entities dependent on the entity being modeled. These will typically be created along with the entity being modeled, and should be removed if the entity being modeled is removed.

Green – independent entities. These typically pre-exist in your VIVO when adding the entity being modeled. These should not be removed if the entity being modeled is removed.

10.5.8 Rich export SPARQL queries

VIVO's rich export queries are organized by typical sections of a curriculum vitae or CV.

They may be useful to supplement the examples of SPARQL queries[175] in the SPARQL Resources[176] section of this wiki.

Queries are grouped by directory in the /productMods/WEB-INF/rich-export section of the VIVO source code.

10.5.8.1 Rich export SPARQL queries: Address

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

address.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>


CONSTRUCT {
     ?address ?property ?object .
} WHERE {
     PERSON_URI obo:ARG_2000028 ?vcard .
     ?vcard vcard:hasAddress ?address .
     ?address ?property ?object .
}
```

locationOfAddress.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
```

175 https://wiki.duraspace.org/display/VIVO/Example+SPARQL+queries
176 https://wiki.duraspace.org/display/VIVO/SPARQL+Resources

```
CONSTRUCT {
     ?geographicLocation ?property ?object .
} WHERE {
     PERSON_URI obo:ARG_2000028 ?vcard .
     ?vcard vcard:hasAddress ?address .
     ?address obo:RO_0001025 ?geographicLocation .
     ?geographicLocation ?property ?object .
}
```

10.5.8.2 Rich export SPARQL queries: Advising

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

advisee.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?advisee ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee ?property ?object .
}
```

adviseesDegreeAlt.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?degree ?property ?object
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee core:relates ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0002234 ?awardedDegree .
     ?awardedDegree core:relates ?degree .
     ?degree a core:AcademicDegree .
     ?degree ?property ?object
}
```

adviseesEducationalInstitutionAlt.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {
     ?educationalInstitution rdfs:label ?label
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee core:relates ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0000057 ?educationalInstitution .
     ?educationalInstitution a foaf:Organization .
     ?educationalInstitution rdfs:label ?label
}
```

adviseesEducationalEndDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?dateTimeValue ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee core:relates ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:end ?dateTimeValue .
     ?dateTimeValue ?property ?object .
}
```

associatedDegree.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?degree ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:degreeCandidacy ?degree .
     ?degree ?property ?object .
}
```

associatedEducationalTraining.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?educationalTraining ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:advisingContributionTo ?educationalTraining .
     ?educationalTraining ?property ?object .
}
```

associatedEducationalTrainingAlt.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?educationalTraining ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?advisingRelationship .
     ?advisingRelationship a core:AdvisingRelationship .
     ?advisingRelationship core:relates ?advisee .
     ?advisee a foaf:Person .
     ?advisee obo:RO_0000053 ?adviseeRole .
     ?adviseeRole a core:AdviseeRole .
     ?adviseeRole core:relatedBy ?advisingRelationship .
     ?advisee core:relates ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining ?property ?object .
}
```

10.5.8.3 Rich export SPARQL queries: Award

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

award.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?award ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?awardReceipt .
     ?awardReceipt a core:AwardReceipt .
     ?awardReceipt core:relates ?award .
     ?award a core:Award .
     ?award ?property ?object .
}
```

conferringOrganization.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?organization ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?awardReceipt .
     ?awardReceipt a core:AwardReceipt .
     ?awardReceipt core:assignedBy ?organization .
     ?organization a foaf:Organization .
     ?organization ?property ?object .
}
```

sponsoringOrganization.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?organization ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?awardReceipt .
     ?awardReceipt a core:AwardReceipt .
     ?awardReceipt core:relates ?award .
     ?award a core:Award .
     ?award core:sponsoredBy ?organization .
     ?organization ?property ?object .
}
```

10.5.8.4 Rich export SPARQL queries: Credential

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

credential.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?credential ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:relates ?credential .
     ?credential a core:Credential .
     ?credential ?property ?object .
}
```

credentialGoverningAuthority.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?organization ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:relates ?credential .
     ?credential a core:Credential .
     ?credential core:hasGoverningAuthority ?organization .
     ?organization ?property ?object .
}
```

elibibleForCredential.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?credential ?property ?object .
} WHERE {
     PERSON_URI core:eligibleFor ?credential .
     ?credential ?property ?object .
}
```

issuedCredential.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
```

```
CONSTRUCT {
     ?issuedCredential ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential ?property ?object .
}
```

issuedCredentialExpirationDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:expirationDate ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

issuedCredentialIssueDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:dateIssued ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

issuedCredentialSubjectArea.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?subjectArea ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?issuedCredential .
     ?issuedCredential a core:IssuedCredential .
     ?issuedCredential core:hasSubjectArea ?subjectArea .
     ?subjectArea ?property ?object .
}
```

10.5.8.5 Rich export SPARQL queries: Educational Training

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

educationalTraining.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?educationalTraining ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining ?property ?object .
}
```

educationalTrainingDegree.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?degree ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0002234 ?awardedDegree .
     ?awardedDegree a core:AwardedDegree .
     ?awardedDegree core:relates ?degree .
     ?degree a core:AcademicDegree .
     ?degree ?property ?object .
}
```

educationalTrainingEndDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?dateTimeInterval core:end ?date .
     ?date ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:end ?date .
     ?date ?property ?object .
}
```

educationalTrainingLocation.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?geographicLocation ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0000057 ?organization .
     ?organization a foaf:Organization .
     ?organization obo:RO_0001025 ?geographicLocation .
     ?geographicLocation a core:GeographicLocation .
     ?geographicLocation ?property ?object .
}
```

educationalTrainingOrganization.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?organization ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining obo:RO_0000057 ?organization .
     ?organization a foaf:Organization .
     ?organization ?property ?object .
}
```

educationalTrainingStartDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?date ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000056 ?educationalTraining .
     ?educationalTraining a core:EducationalProcess .
     ?educationalTraining core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:start ?date .
     ?date ?property ?object .
}
```

10.5.8.6 Rich export SPARQL queries: Funding

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

grantAwardedBy.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {
     ?awardingOrganization rdfs:label ?label
} WHERE {
    {
      {PERSON_URI core:relatedBy ?investigatorRole .
       ?investigatorRole a core:PrincipalInvestigatorRole
      }
        union
      {PERSON_URI core:relatedBy ?investigatorRole .
       ?investigatorRole a core:CoPrincipalInvestigatorRole
      }
     }

     ?investigatorRole core:relatedBy ?grant .
     ?grant a core:Grant .
     ?grant core:assignedBy ?awardingOrganization .
     ?awardingOrganization a foaf:Organization .
     ?awardingOrganization rdfs:label ?label
}
```

grants.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT {
     ?grant ?property ?object .
     ?investigatorRole core:relatedBy ?grant .
} WHERE {
     {
       { PERSON_URI core:relatedBy ?investigatorRole .
         ?investigatorRole a core:PrincipalInvestigatorRole
       }
         union
       { PERSON_URI core:relatedBy ?investigatorRole .
         ?investigatorRole a core:CoPrincipalInvestigatorRole
       }
     }

     ?investigatorRole core:relatedBy ?grant .
     ?grant a core:Grant .
     ?grant ?property ?object
}
```

10.5.8.7 Rich export SPARQL queries: Membership

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

memberRoleIn.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?endeavor ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000053 ?memberRole .
     ?memberRole a core:MemberRole .
     ?memberRole core:roleContributesTo ?endeavor .
     ?endeavor ?property ?object .
}
```

10.5.8.8 Rich export SPARQL queries: Outreach

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

outreachRoleIn.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?endeavor ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000053 ?outreachRole .
     ?outreachRole a core:OutreachProviderRole .
     ?outreachRole core:roleContributesTo ?endeavor .
     ?endeavor ?property ?object .
}
```

10.5.8.9 Rich export SPARQL queries: Patent

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

assignee.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?assignee ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?patent core:assignee ?assignee .
     ?assignee ?property ?object .
}
```

inventors.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?person ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?authorship core:relates ?person .
     ?person a foaf:Person .
     ?person ?property ?object .
}
```

patent.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?patent ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?patent ?property ?object .
}
```

patentFiledDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?patent core:dateFiled ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

patentIssuedDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?patent .
     ?patent rdf:type bibo:Patent .
     ?patent core:dateIssued ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

10.5.8.10 Rich export SPARQL queries: Position

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

locationForPosition.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?organization core:hasGeographicLocation ?geographicLocation .
     ?geographicLocation rdfs:label ?label .
} WHERE {
     PERSON_URI core:relatedBy ?position .
     ?position a core:Position .
     ?position core:relates ?organization .
     ?organization a foaf:Organization .
     ?organization obo:RO_0001025 ?geographicLocation .
     ?geographicLocation rdfs:label ?label .
}
```

organizationForPosition.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?position core:positionInOrganization ?organization .
     ?organization rdfs:label ?label .
} WHERE {
     PERSON_URI core:relatedBy ?position .
     ?position a core:Position .
     ?position core:relates ?organization .
     ?organization a foaf:Organization .
     ?organization rdfs:label ?label .
}
```

subOrganizationForPosition.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?organization core:hasSubOrganization ?subOrganization .
     ?subOrganization rdfs:label ?label .
} WHERE {
     PERSON_URI core:relatedBy ?position .
     ?position a core:Position .
     ?position core:relates ?organization .
     ?organization a foaf:Organization .
     ?organization obo:BFO_0000050 ?subOrganization .
     ?subOrganization rdfs:label ?label .
}
```

superOrganizationForPosition.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>

CONSTRUCT {
     ?superOrganization ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?position .
     ?position a core:Position .
     ?position core:relates ?organization .
     ?organization a foaf:Organization .
     ?organization obo:BFO_0000051 ?superOrganization .
     ?superOrganization ?property ?object .
}
```

10.5.8.11 Rich export SPARQL queries: Presentation

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

meetingLocation.sparql

```
PREFIX vivo: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?location rdfs:label ?locationName .
} WHERE {
     PERSON_URI obo:RO_0000053 ?presenterRole .
     ?presenterRole a core:PresenterRole .
     ?presenterRole obo:BFO_0000054 ?presentation .
     ?presentation obo:BFO_0000050 ?containingEvent .
     ?containingEvent obo:RO_0001025 ?location .
     ?location rdfs:label ?locationName .
}
```

meetingName.sparql

```
PREFIX vivo: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {
     ?containingEvent rdfs:label ?containingEventName
} WHERE {
     PERSON_URI obo:RO_0000053 ?presenterRole .
     ?presenterRole a core:PresenterRole .
     ?presenterRole obo:BFO_0000054 ?presentation .
     ?presentation obo:BFO_0000050 ?containingEvent .
     ?containingEvent rdfs:label ?containingEventName
}
```

presenterRoleIn.sparql

```
PREFIX vivo: <http://vivoweb.org/ontology/core#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT {
     ?presentation rdfs:label ?presentationTitle .
     ?presenterRole rdfs:label ?roleLabel .
} WHERE {
     PERSON_URI obo:RO_0000053 ?presenterRole .
     ?presenterRole a core:PresenterRole .
     ?presenterRole rdfs:label ?roleLAbel .
     ?presenterRole obo:BFO_0000054 ?presentation .
     ?presentation rdfs:label ?presentationTitle .
}
```

10.5.8.12 Rich export SPARQL queries: Publication

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

associatedJournal.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publicationVenue ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication core:hasPublicationVenue ?publicationVenue .
     ?publicationVenue ?property ?object .
}
```

authors.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
     ?coAuthorship ?property1 ?object1 .
     ?person ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication core:relatedBy ?coAuthorship .
     ?coAuthorship a core:Authorship .
     ?coAuthorship ?property1 ?object1 .
     ?coAuthorship core:relates ?person .
     ?person a foaf:Person .
     ?person ?property2 ?object2 .
}
```

presentedAtEvent.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?event ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:presentedAt ?event .
     ?event ?property ?object .
}
```

presentedAtEventEndDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>PREFIX bibo: <http://purl.org/ontology/bibo/>

CONSTRUCT {
     ?endDate ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:presentedAt ?event .
     ?event ?property ?object .
     ?event core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:end ?endDate .
     ?endDate core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

presentedAtEventLocation.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?location rdfs:label ?locationName .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:presentedAt ?event .
     ?event obo:RO_0001025 ?location .
     ?location rdfs:label ?locationName .
}
```

presentedAtEventStartDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?startDate ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:presentedAt ?event .
     ?event ?property ?object .
     ?event core:dateTimeInterval ?dateTimeInterval .
     ?dateTimeInterval core:start ?startDate .
     ?startDate core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

publication.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publication ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication ?property ?object .
}
```

publicationDate.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?date ?property ?object .
     ?precision ?property2 ?object2 .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication ?dateTimeValue ?date .
     ?date ?property ?object .
     ?date core:dateTimePrecision ?precision .
     ?precision ?property2 ?object2 .
}
```

publicationPartOfInfoResource.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?informationResource ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication obo:BFO_0000050 ?informationResource .
     ?informationResource ?property ?object .
}
```

publicationReproducedIn.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?informationResource ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:reproducedIn ?informationResource .
     ?informationResource ?property ?object .
}
```

publicationStatus.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX bibo: <http://purl.org/ontology/bibo/>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publicationStatus ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication bibo:status ?publicationStatus .
     ?publicationStatus ?property ?object .
}
```

publicationURL.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>

CONSTRUCT {
     ?urllink ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication obo:ARG_2000028 ?vcard .
     ?vcard vcard:hasURL ?urllink .
     ?urllink ?property ?object .
}
```

publisher_variant1.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publisher ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication core:hasPublicationVenue ?publicationVenue .
     ?publicationVenue core:publisher ?publisher .
     ?publisher ?property ?object .
}
```

publisher_variant2.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?publisher ?property ?object .
} WHERE {
     PERSON_URI core:relatedBy ?authorship .
     ?authorship a core:Authorship .
     ?authorship core:relates ?publication .
     ?publication a obo:IAO_0000030 .
     ?publication core:publisher ?publisher .
     ?publisher ?property ?object .
}
```

10.5.8.13 Rich export SPARQL queries: Teaching

Note that these are CONSTRUCT queries designed to create a small Jena model for export as a whole after a series of queries has been run. The PERSON_URI variable is substituted by VIVO at runtime.

teacherRoleIn.sparql

```
PREFIX core: <http://vivoweb.org/ontology/core#>
PREFIX obo: <http://purl.obolibrary.org/obo/>

CONSTRUCT {
     ?course ?property ?object .
} WHERE {
     PERSON_URI obo:RO_0000053 ?teacherRole .
     ?teacherRole a core:TeacherRole .
     ?teacherRole obo:BFO_0000054  ?course .
     ?course ?property ?object .
}
```

10.5.9 VIVO-ISF deployment in VIVO

10.5.9.1

- ISF Development Source
- ISF in VIVO

10.5.9.2 ISF Development Source

The VIVO-ISF files can be found in the OpenRif repositories on GitHub. https://github.com/openrif/vivo-isf-ontology

10.5.9.3 ISF in VIVO

- TBox filegraph directory largely mirrors /source directory from ISF repository
- selective (manual) removal of certain files and parts of files
    - no anatomy.owl
    - smaller clinical.owl
    - no research-resource-phenotype-mp.owl
    - no sharecenter.owl
    - vastly smaller research-resource.owl
    - smaller object-properties.owl and data-properties.owl
- additional VIVO-specific content:
    - personTypes.n3
    - object-properties3.owl
- additional axioms primarily for application control purposes
    - appControls-temp.n3
    - classes-additional.owl
    - dataDomains.rdf
    - objectDomains.rdf
    - objectRanges.rdf
- labels removed from ontology files and stored elsewhere for editing in the interface
- additional PropertyConfig.n3 file to set up "faux properties," e.g.:
    - "relatedBy" when used between a Person and Position is called "positions" and configured separately
    - "bearer of" when used between a Person and a ServiceProviderRole is called "has service provider role" and configured separately

10.6 Freemarker Template Variables and Directives

Template variables are made available to render dynamic content within the application. To print a variable's value in FreeMarker, use the following syntax:
${variableName}

Some variables have methods which can be used to return a value or perform a task such as adding a stylesheet or script to the <head> element.
${stylesheets.add('<link rel="stylesheet" href="mystylesheet.css" />')}
${headScripts.add(<script type="text/javascript" src="myscript.js"></script>)}

Special template directives provide debugging features that assist in template development.
<@describe var="stylesheets" />
(describe the methods callable on a template variable)

<@dump var ="stylesheets" />
(dump the contents of a template variable)

<@dumpAll />
(dump the contents of the template data model)

A sample page at *

```
http://yourLocalInstance.com/freemarkersamples
```

* demonstrates most of the methods and directives available within a template. The template file responsible for this page is vitro-core/webapp/web/templates/freemarker/body/samples.ftl.

10.7 Architecture

10.7.1 Overview

VIVO is an enterprise class software system relying on numerous open source software components. Fundamentally, VIVO relies on Vitro (see below). VIVO adds a collection of ontologies (see Ontology Reference (see page 310)) to represent data about scholarship.

10.7.2 Vitro

Vitro is an open source, general purpose, semantic web engine. It is the application development platform underlying VIVO. Vitro has no domain knowledge. Given ontologies regarding a domain, Vitro supports the editing of the ontology, creation of individuals, management of individuals on "pages" which it generates, organization of individuals into "class groups," indexing, search, faceted browsing, query, import, and export. Vitro has been used to manage collections of clinical trials, spaceships, library catalogs, datasets, and many more.

VIVO is Vitro with an ontology for representing scholarship, and a set of displays and visualizations that support the use of data for expert finding, team building, assessment, and other VIVO use cases.

Vitro can be built an operated independently of VIVO. VIVO is completely dependent on Vitro.

10.7.3 VIVO

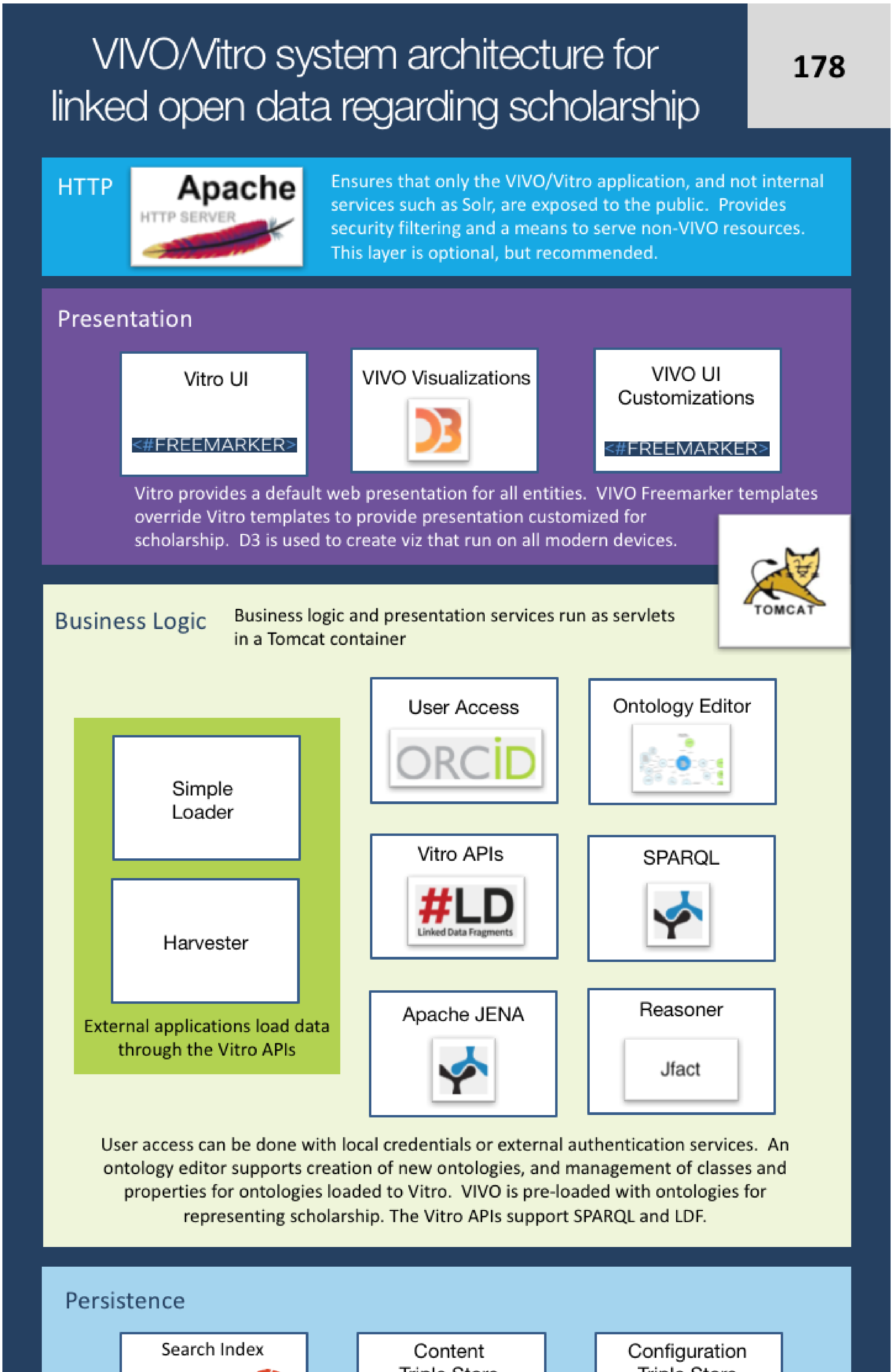
VIVO is a customized Vitro. The table below shows how VIVO compares to Vitro.

	Vitro	**VIVO**
Purpose	General-purpose tool for working with Semantic Data	Specialized tool for Research Networking
Ontology	No ontology	Includes an ontology (VIVO-ISF) for Research Networking

	Vitro	**VIVO**
Theme	Minimal theme	Elaborate theme, display and editing are customized for the ontology
Display Rules	Default display rules	Annotations are used to: • Assign data properties to groups • Arrange property groups on the page
Form editing	Default editing forms	Editing is customized to the ontology
Search Index	Default search index	Search index contains additional fields specific to VIVO
Function ality	Default functionality	Additional functionality: visualizations, interface to Harvester, QR codes, etc.

10.7.4 Component View

VIVO, with Vitro, as "made" out of components, including other open source software components. The figure below shows the various software components that are used in a VIVO/Vitro system.



VIVO/Vitro system architecture for linked open data regarding scholarship

178

Ensures that only the VIVO/Vitro application, and not internal services such as Solr, are exposed to the public. Provides security filtering and a means to serve non-VIVO resources. This layer is optional, but recommended.

Vitro provides a default web presentation for all entities. VIVO Freemarker templates override Vitro templates to provide presentation customized for scholarship. D3 is used to create viz that run on all modern devices.

Business logic and presentation services run as servlets in a Tomcat container

External applications load data through the Vitro APIs

User access can be done with local credentials or external authentication services. An ontology editor supports creation of new ontologies, and management of classes and properties for ontologies loaded to Vitro. VIVO is pre-loaded with ontologies for representing scholarship. The Vitro APIs support SPARQL and LDF.

10.7.5 Additional Resources

- Vitro (see page 363)
- VIVO and Vitro (see page 363)
- Software Architecture Overview (see page 364)
- VIVO Data Models (see page 368)
- VIVO and the Solr search engine (*) (see page 378)
- Image storage (see page 384)

10.7.6 Vitro

Vitro is an open source, general purpose, semantic web engine. It is the application development platform underlying VIVO. Vitro has no domain knowledge. Given ontologies regarding a domain, Vitro supports the editing of the ontology, creation of individuals, management of individuals on "pages" which it generates, organization of individuals into "class groups," indexing, search, faceted browsing, query, import, and export. Vitro has been used to manage collections of clinical trials, spaceships, library catalogs, datasets, and many more.

VIVO is Vitro with an ontology for representing scholarship, and a set of displays and visualizations that support the use of data for expert finding, team building, assessment, and other VIVO use cases.

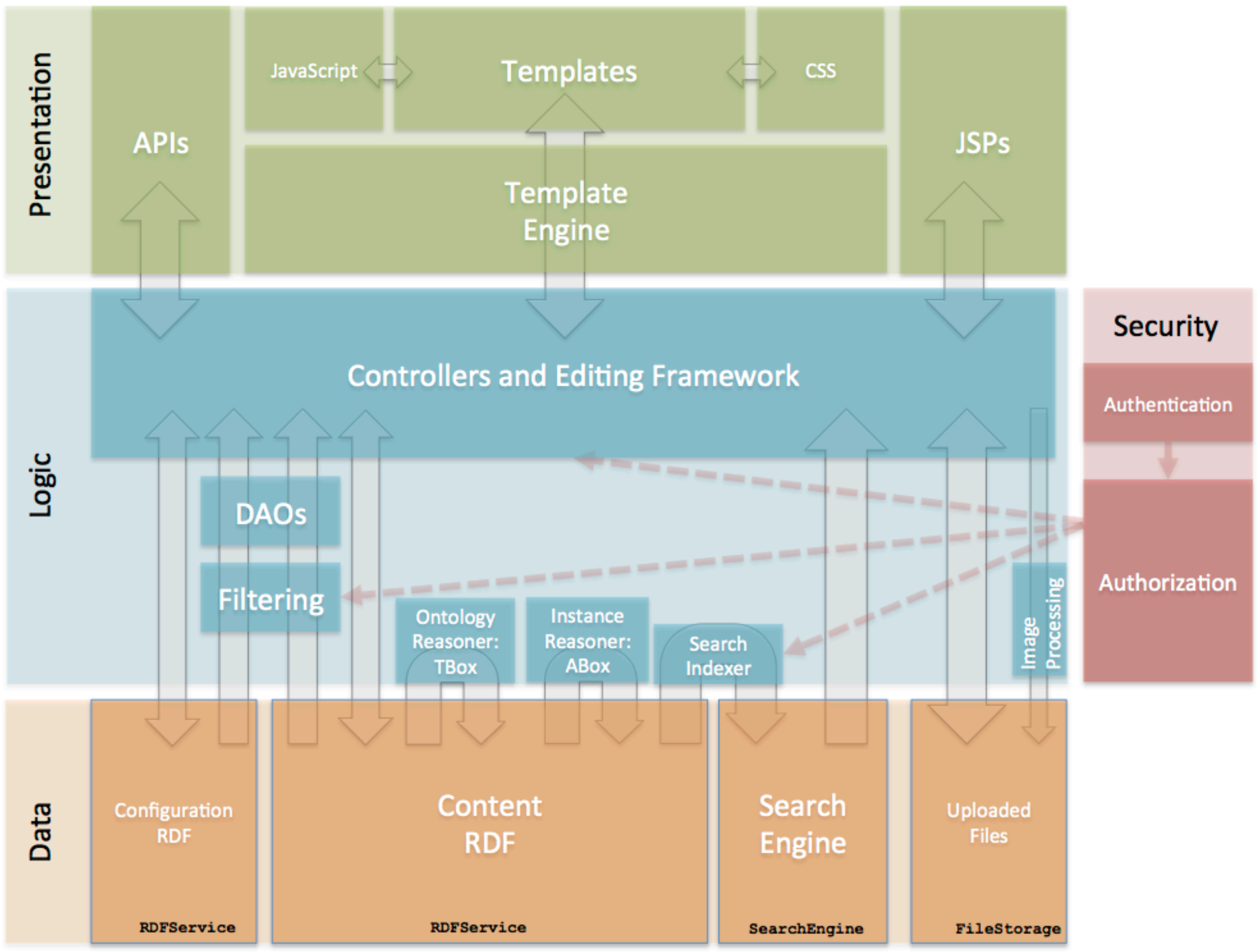
10.7.7 VIVO and Vitro

VIVO itself is a customization of a more generic product called Vitro. Here is how VIVO has been customized from Vitro.

	Vitro	**VIVO**
Purpose	General-purpose tool for working with Semantic Data	Specialized tool for Research Networking
Ontology	No ontology	Includes an ontology (VIVO-ISF) for Research Networking
Theme	Minimal theme	Elaborate theme, display and editing are customized for the ontology
Display Rules	Default display rules	Annotations are used to: • Assign data properties to groups • Arrange property groups on the page
Form editing	Default editing forms	Editing is customized to the ontology
Search Index	Default search index	Search index contains additional fields specific to VIVO

	Vitro	VIVO
Functionality	Default functionality	Additional functionality: visualizations, interface to Harvester, QR codes, etc.

10.7.8 Software Architecture Overview



10.7.8.1 Data

VIVO has four data stores. When copying, backing up, or restoring a VIVO installation, all four data stores should be considered.

Content RDF

This is where most of VIVO's information is stored. Names of individuals, relationships between individuals, types of individuals (for example, Person or Organization), are all stored in the Content RDF

Content RDF uses a triple-store or other SPARQL endpoint. Usually, the triple-store is a Jena SDB implementation, with a MySQL database.

The interface is specified by `RDFService.java`.

Configuration RDF

This is where VIVO's parameters are stored, like which templates are used to display what types of data. Other parameters describe how the custom editing screens are applied to complex data structures. The Configuration RDF is also the storage for VIVO's user accounts.

Configuration RDF uses a triple-store or other SPARQL endpoint. The triple-store is a Jena TDB implementation, with files kept in the home directory of the VIVO application.

The interface is specified by `RDFService.java`.

Search Engine

In theory, all of the search operations in VIVO could be performed using SPARQL queries against the RDF. In practice, however, a dedicated search engine gives a much faster response. The search engine is available to VIVO's users, to assist in finding pages of interest. The search engine is also used internally, to provide prompt response to requests for auto-completion, indexes, counts, and other data.

The search engine permits queries that yield faceted results, for a more successful search. Usually, it is implemented with a Solr web application. By default, Solr is installed in the same web server as VIVO. However, it is easy to move Solr to a different web server, to improve performance.

The interface is specified by `SearchEngine.java`

Uploaded Files

VIVO allows individuals to upload images for their profile pages. VIVO also generates a thumbnail image for more compact display. These images are kept in the Uploaded Files storage. Each file is assigned a URI, so it can be distinguished from other files of the same name. Currently this is only used for images, but VIVO could be customized to store other types of files here as well.

The default implementation uses a storage system similar to PairTree[177].

The interface is specified by `FileStorage.java`

10.7.8.2 Logic

VIVO adds a layer of "business logic" to the data storage. It uses inference to add to the data. It applies policies to determine which users are authorized to see which pieces of data.

Controllers and Editing Framework

The controllers contain the top-level logic, determining how to respond to web requests. This includes fetching data, making decisions based on that data, and displaying the results.

The Editing Framework provides the user with the tools needed to edit the RDF data. In some cases, a simple default screen will suffice. For more elaborate data structures, the Editing Framework creates related groups of data objects, and enforces the relationships among them.

DAOs

The DAOs, or Data Access Objects, form a layer of secondary logic. They provide a large number of utility subroutines, to take the repetitive processing tasks away from the Controllers and Editing Framework.

The DAOs also provide a framework for the filtering layer.

There are a large number of interfaces that define the DAO layer.

177 https://wiki.ucop.edu/display/Curation/PairTree

Filtering

Data within VIVO can be public or private, or shades of gray. The Filtering layer works with the Authentication system to determine which pieces of data may be displayed to a particular user.

The Filtering layer means that the Controllers don't need to include logic for this sort of decision. The Controller asks the Filtered DAOs for data, and receives as much data as the current user is authorized to see.

The interface is specified by `VitroFilters.java`

Ontology Reasoners: ABox and TBox

One of the principal strengths of RDF is that we can infer additional data from the data at hand. However, the logic involved can be complicated and time-consuming.

Currently VIVO applies two different reasoners to the Content RDF. The TBOX - or Ontology models - are small enough that extensive reasoning can be applied. Currently, the Pellet reasoner is used. Applying that same level of inference to the ABOX - or Assertions models - would take a prohibitive amount of time. VIVO uses its own reasoner for this, applying only those logical inferences that VIVO requires to function.

Search Indexer

The Search Indexer reacts to changes in the Content RDF, updating the search index to reflect those changes. Several types of logic are employed to determine which individuals are affected by the RDF changes, and how to build the search records for those individuals. Sometimes a single change requires that several search records be rebuilt.

Image Processor

When images are uploaded through the GUI, the Image Processor creates a thumbnail image, cropped and sized as the user requests. Currently, the image processor is based on the Java Advanced Imaging library.

10.7.8.3 Presentation

The presentation layer is where the web pages of VIVO are created. Most of the web pages are created using the Freemarker template engine. However, a number of pages are still created by JSPs.

Template Engine and Templates

VIVO uses the Freemarker Template Engine to construct the HTML for its web pages. The templates describe the format and structure of the pages, and the template engine inserts relevant data each time the template is used.

JavaScript

VIVO relies heavily on JQuery to create a rich and responsive user interface. Other scripts are used also.

CSS

Like any web application, VIVO uses CSS files to produce a consistent style across the user interface.

JSPs

Early releases of VIVO were built almost entirely using JSPs. The change to Freemarker was an attempt to insure better separation between the Logic layer and the Presentation layer.

Some JSPs are still used in VIVO. In general these are restricted to administrative pages, including advanced data manipulation and "back-end editing".

APIs

VIVO supports a collection APIs for importing and exporting data. The APIs have no presentation layer, per se. The format of their responses is determined by the nature of the request, and usually does not involve HTML. Responses to API requests are constructed entirely by the Controllers.

10.7.8.4 Security

The security system determines what data a user may see, what data they may modify, and what functions they may perform.

Authentication

VIVO includes its own authentication system, including user accounts with email addresses as identifiers and passwords as credentials. VIVO can be configured to use an external authentication system also. In this case, VIVO still maintains a user account for each user, but no passwords are stored. If the external authentication system asserts that a user has properly logged in, VIVO accepts that assertion.

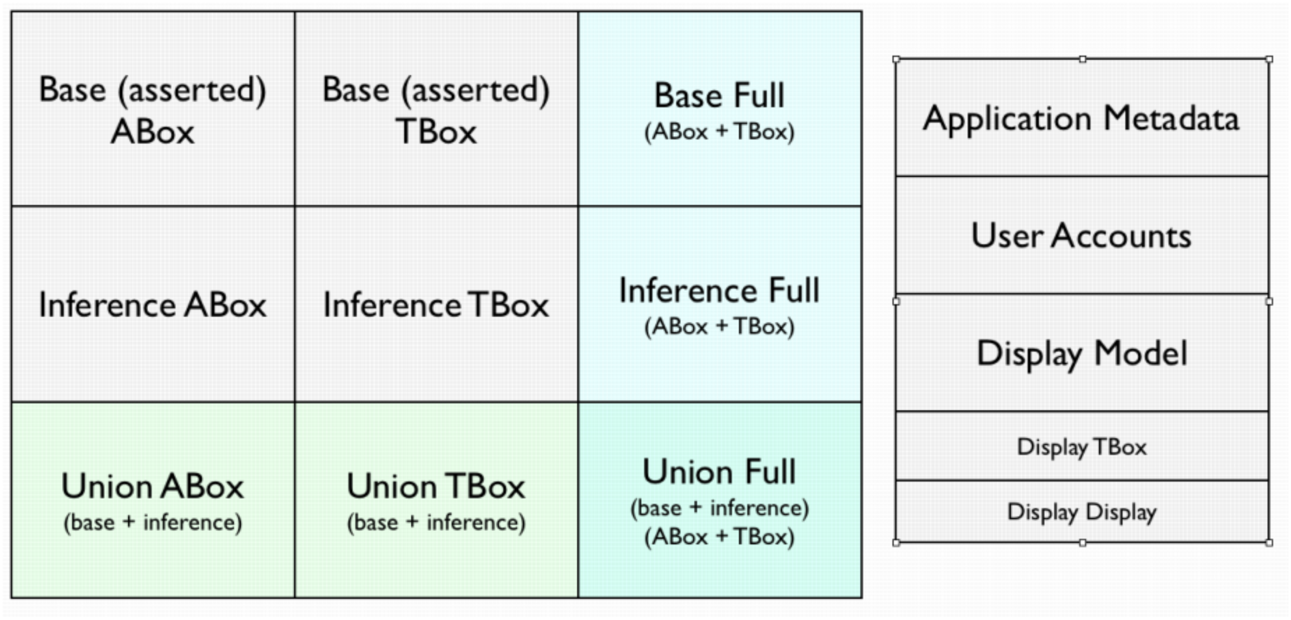
Authorization

The Authorization system relies on a list of Policy objects to determine what a user may do. Before the Controllers or the Editing Framework or the Filtering layer take any action, they consult the Policy list to determine whether that action is authorized for the current user.

This very flexible set of Policies permits VIVO to classify some data as public or private, while other data is private except to the user who owns it.

10.7.9 VIVO Data Models

10.7.9.1 Concepts



Frequently, we talk about "the data model" in VIVO. But this is an over-simplification which can be useful at times, but misleading at other times. In fact, VIVO contains a matrix of data models and sub-models, graphs, datasets and other constructs.

It might be more accurate to talk about the union of these data models as "the knowlege base". However, the terminology of "the data model" is firmly entrenched.

Beginning in VIVO release 1.6, we are attempting to simplify this complex collection of models, and to produce a unified access layer. This is a work in progress. Regardless of how clean the design might eventually become, this will remain an area with complex requirements which cannot be satisfied by simplistic solutions.

Divisions in the knowledge base

Depending on what you want to do with the data, it can be useful to sub-divide it by one or more of the following criteria:

Types of statements

An RDF model is often divided into ABox (assertions) and TBox (terminology). In RDF, there is no technical distinction between TBox and ABox data. They are stored separately because they are used for different purposes. The combination of the two is informally called the Full model.

	Data type	**Example data**

TBox	"Terminological data"
	
	Defines classes, properties, and relationships in your ontology.
		`foaf:Person`
		`   a owl:Class ;`
		`   rdfs:subClassOf owl:Thing ;`
		`   rdfs:label "Person"@en .`
		`ex:preferredName`
		`   a owl:DatatypeProperty ;`
		`   rdfs:subPropertyOf`
		`skos:prefLabel,`
		`                      foaf:name,`
		`                      rdfs:label ;`
		`   rdfs:domain foaf:Person ;`
		`   rdfs:label "preferred name"@en .`

ABox	"Assertion data"
	
	Enumerates the individual instances of your classes and describes them.
		`local:tobyink`
		`   a foaf:Person ;`
		`   ex:preferredName "Toby`
		`Inkster" .`

Full	The TBox and the ABox together, treated as a single model. For example, when you use the RDF tools to remove statements, you want them removed regardless of whether they are found in the TBox or the ABox.	

Source of statements

An RDF model can also be divided into Assertions and Inferences. The combination of the two is informally called the Union.

Statement type	Meaning	Example data
Assertions	Statements that you explicitly add to the model, either through setup, ingest, or editing.	`local:tobyink`[178] `rdfs:type`[179] `core:FacultyMember .`

Inferences	Statements that the semantic reasoner adds to the model, by reasoning about the assertions, or about other inferences.	`local:tobyink rdfs:type foaf:Person .`
		
		`local:tobyink rdfs:type foaf:Agent .`
		
		`local:tobyink rdfs:type owl:Thing .`

Union	The combination of Assertions and Inferences. For most purposes, this is the desired model. You want to know what statements are available, without regard to whether they were asserted or inferred.	

"Content" vs. "Configuration"

We sometimes distinguish between the data that VIVO is serving (Content) and the data that VIVO itself uses (Configuration). The Content is available for display, for searching, for serving as Linked Open Data. The Configuration controls how the content is displayed, who can access the data, and what VIVO itself looks like.

Model type	Purpose	Examples
Configuration	Data about the VIVO application itself.	Application parameters User Accounts Display options
Content	The payload - the data that VIVO is intended to distribute.	People data Publications data Grant data etc.

Model scope

The knowledge base exists for as long as VIVO is running. However, subsets or facets of the knowledge base are often used to satisfy a particular HTTP request, or through the length of a VIVO session for a particular user. These

178 http://localtobyink
179 http://rdfstype

subsets are created dynamically from the full knowledge base, used for as long as they are useful, and then discarded.

Scope	Purpose	Example	Discarded when...
Application (Servlet Context)	Created for the life of VIVO.		Never discarded.
Session	Created for a particular logged-in user	Data that is filtered by what the user is permitted to view.	When the user logs out, or the session times out.
Request	Created for a single HTTP request	Data that is organized by the languages that are preferred by the browser.	When the individual request has been satisfied.

At present, the Session lifespan is almost never used. However, potential use cases do exist for it.

The Request lifespan is used extensively, since it provides a convenient way to manage database connections and minimize contention for resources.

Purpose vs. scope

It is tempting to think of the models of the Servlet Context as equivalent to the unfiltered models of the Request. They may even represent the very same data. However, they have different scope, which makes them very different in practice.

The unfiltered models in the Request go out of scope when the Request has been satisfied. The resources required by these models have short lifetimes and are very easily managed. The models of the Servlet Context never go out of scope until VIVO is shut down. It is difficult to reclaim resources such as database connections or processor memory from these models.

Filtering

⚠ TBD: talk about language filters and policy filters. What do we mean by "unfiltered?"

10.7.9.2 The Data Models

This is a summary of the data models:

The basic content	Base ABox, Base TBox, Inferred ABox, Inferred TBox	Named graphs from the RDF Service (optionally with sub-graphs).
Views of the content	Base Full, Inferred Full, Union ABox, Union TBox, Union Full	Views of the 4 basic content graphs in different combinations.

The configuration	Application Metadata, User Accounts, Display Model, Display TBox, DisplayDisplay	Named graphs from the application datasource.

10.7.9.3 Increasing complexity

The structure of the data models has grown as VIVO has developed. New models, new structures, and new means of accessing the data have been added as required by the growing code. The resulting data layer has grown more complex and more error-prone.

The `RDFService` interface, increases the flexibility of data sources, and promises to allow a more unified view of the knowledge base. However, the transition to `RDFService` is not complete, and so this adds another layer of complexity to the data issues. New structures have been added, but none removed.

Beyond the models

There is an incredible variety of ways to access all of these models. Some of this variety is because the models are accessed in different ways for different purposes. Additional variety stems from the evolution of VIVO in which new mechanisms were introduced without taking the time and effort to phase out older mechanisms.

Here are some of the ways for accessing data models:

Attributes on Context, Session, or Request

Previously, it was common to assign a model to the ServletContext, to the HTTP Session, or to the HttpSessionRequest like this:

```
OntModel ontModel = (OntModel) getServletContext().getAttribute("jenaOntModel");

Object sessionOntModel = request.getSession().getAttribute("jenaOntModel");

ctx.setAttribute("jenaOntModel", masterUnion);
```

Occasionally, conditional code was inserted, to retrieve a model from the Request if available, and to fall back to the Session or the Context as necessary. Such code was sporadic, and inconsistent. This sort of model juggling also involved inversions of logic, with some code acting so a model in the Request would override one in the Session, while other code would prioritize the Session model over the one in the Request. For example:

```
public OntModel getDisplayModel(){
    if( _req.getAttribute("displayOntModel") != null ){
        return (OntModel) _req.getAttribute(DISPLAY_ONT_MODEL);
    } else {
        HttpSession session = _req.getSession(false);
        if( session != null ){
            if( session.getAttribute(DISPLAY_ONT_MODEL) != null ){
                return (OntModel) session.getAttribute(DISPLAY_ONT_MODEL);
            }else{
                if( session.getServletContext().getAttribute(DISPLAY_ONT_MODEL) != null){
                    return (OntModel)session.getServletContext().getAttribute(DISPLAY_ONT_MODEL);
                }
            }
        }
    }
    log.error("No display model could be found.");
    return null;
}
```

This mechanism has been removed in 1.6, being subsumed into the `ModelAccess` class (see below). Now, the `ModelAccess` attributes on Request, Session and Context are managed using code that is private to `ModelAccess` itself. Similarly, the code which gives priority to a Request model over a Session model is uniformly implemented across the models.

It remains to be seen whether this uniformity can satisfy the various needs of the application. If not, at least the changes can all be made within a single point of access.

The DAO layer

This mechanism is pervasive through the code, and remains quite useful. In it, a `WebappDaoFactory` is created, with access to particular data models. This factory then can be used to create DAO objects which satisfy interfaces like `IndividualDao`, `OntologyDAO`, or `UserAccountsDAO`. Each of these object implements a collection of convenience methods which are used to manipulate the backing data models.

Because the factory and each of the DAOs is an interface, alternative implementations can be written which provide

- Optimization for Jena RDB models
- Optimization for Jena SDB models
- Filtering of restricted data
- and more...

Initially, the `WebappDaoFactory` may have been used only with the full Union model. But what if you want to use these DAOs only against asserted triples? Or only against the ABox? This led to the `OntModelSelector`.

OntModelSelectors

An `OntModelSelector` provides a way to collect a group of Models and construct a `WebappDaoFactory`. With slots for ABox, TBox, and Full model, an `OntModelSelector` could provide a consistent view on assertions, or on inferences, or on the union. The `OntModelSelector` also holds references to a display model, an application metadata model, and a user accounts model, but these are more for convenience than flexibility.

Prior to release 1.6, `OntModelSelectors`, like `OntModels`, were stored in attributes of the Context, Session, and Request. They have been subsumed into the `ModelAccess` class.

Further, the semantics of the "standard" `OntModelSelectors` have changed, so they only act as facades before the Models store in `ModelAccess`. In this way, if we make this call:

```
ModelAccess.on(session).setOntModel(ModelID.BASE_ABOX, someWeirdModel)
```

Then both of the following calls would return the same model:

```
ModelAccess.on(session).getOntModel(ModelID.BASE_ABOX);
ModelAccess.on(session).getBaseOntModelSelector().getABoxModel();
```

Again, this is a change in the semantics of OntModelSelectors. It insures a consistent representation of `OntModels` across `OntModelSelectors`, but it is certainly possible that existing code relies on an inconsistent model instead.

The RDF Service

⚠ TBD

Model makers and Model sources

10.7.9.4 The ModelAccess class

⚠ TBD - Show how it represents all of these distinctions. Describe the scope searching and masking, wrt set and get. Include the OntModelSelectors and WADFs.

10.7.9.5 Initializing the Models

When VIVO starts up, `OntModel` objects are created to represent the various data models. The configuration models are created from the datasource connection, usually to a MySQL database. The content models are created using the new RDFService layer. By default this also uses the datasource connection, but it can be configured to use any SPARQL endpoint for its data.

Some of the smaller models are "memory-mapped" for faster access. This means that they are loaded entirely into memory at startup. Any changes made to the memory image will be replicated in the original model.

The data in each model persists in the application datasource (usually a MySQL database), or in the RDFService. Also, data from disk files may be loaded into the models. This may occur:

- the first time that VIVO starts up,
- if a model is found to be empty,
- every time that VIVO starts up.

depending on the particular model.

Where are the RDF files?

In the distribution, the RDF files appear in `[vivo]/rdf` and in `[vitro]/webapp/rdf`. These directories are merged during the build process in the usual way, with files in VIVO preferred over files in Vitro.

During the VIVO build process, the RDF files are copied to the VIVO home directory, and at runtime VIVO will read them from there.

The "first time"

For purposes of initialization, "first time" RDF files are loaded if the relevant data model contains no statements. Content models may also load "first time" files if the RDFService detects that its SDB-based datastore has not been initialized.

Initializing Configuration models

Application metadata

Function: Describes the configuration of VIVO at this site. Many of the configuration options are obsolete.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-applicationMetadata

Source: the application Datasource (MySQL database) (memory-mapped)

If this is the first startup, read the files in rdf/applicationMetadata/firsttime.

- In Vitro, there are none
- In VIVO, `initialSiteConfig.rdf, classgroups.rdf and propertygroups.rdf`

User Accounts

Contains login credentials and assigned roles for VIVO users.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-userAccounts

Source: the application Datasource (MySQL database) (memory-mapped)

If this model is empty, read the files in rdf/auth/firsttime.

- In Vitro, there are none (except during Selenium testing)
- In VIVO, there are none

Every time, read the files in rdf/auth/everytime

- In Vitro, `permission_config.n3`
- In VIVO, there are none.

The Display model

This is the ABox for the display model, and contains the RDF statements that define managed pages, custom short views, and other items.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadata

Source: the application Datasource (MySQL database) (memory-mapped)

If this model is empty, read the files in `rdf/display/firsttime`

- In Vitro, `application.owl, menu.n3, profilePageType.n3, pageList_editableStatements.n3`
- VIVO contains its own copy of menu.n3, which overrides the one in Vitro `aboutPage.n3 menu.n3 PropertyConfig.n3 PropertyConfigSupp.n3`

Every time, read the files in rdf/display/everytime

- in Vitro, `dataGetterLabels.n3   permissions.n3 displayModelListViews.rdf  searchIndexerConfigurationVitro.n3 pageList.n3    vitroSearchProhibited.n3`
- In VIVO `homePageDataGetters.n3   vivoConceptDataGetters.n3 localeSelectionGUI.n3   vivoListViewConfig.rdf n3ModelChangePreprocessors.n3  vivoOrganizationDataGetters.n3 orcidInterfaceDataGetters.n3  vivoQrCodeDataGetter.n3 searchIndexerConfigurationVivo.n3 vivoSearchProhibited.n3`

Display TBox

The TBox for the display model.

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadataTBOX

Source: the application Datasource (MySQL database) (memory-mapped)

Every time, read the files in rdf/displayTbox/everytime.

- In Vitro, `displayTBOX.n3`
- In VIVO, there are none

DisplayDisplay

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-displayMetadata-displayModel

Source: the application Datasource (MySQL database) (memory-mapped)

Every time, read the files in rdf/displayDisplay/everytime

- In Vitro, `displayDisplay.n3`
- In VIVO, there are none.

Initializing Content models

base ABox

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-2

Source: named graph from the RDFService

If first setup, read the files in `rdf/abox/firsttime`

- In Vitro, there are none
- In VIVO, `geopolitical.ver1.1-11-18-11.individual-labels.rdf`

Every time, read the files in `rdf/abox/filegraph`, and create named models in the RDFService. Add them as sub-models to the base ABox. If these files are changed or deleted, update the RDFService accordingly.

- In Vitro, there are none
- In VIVO `documentStatus.owl`
  `academicDegree.rdf   geopolitical.abox.ver1.1-11-18-11.owl    us-states.rdf`
  `continents.n3    validation.n3 dateTimeValuePrecision.owl  vocabularySource.n3`
- Plus whatever data packages you may have added. See Managing Data Packages (see page 89)

base TBox

Name: http://vitro.mannlib.cornell.edu/default/asserted-tbox

Source: named graph from the RDFService (memory-mapped)

If first setup, read the files in rdf/tbox/firsttime (without subdirectories)

- In Vitro, there are none
- In VIVO, additionalHiding.n3 initialTBoxAnnotations.n3

Every time, read the files in `rdf/tbox/filegraph`, and create named models in the RDFService. Add them as sub-models to the base TBox. If these files are changed or deleted, update the RDFService accordingly.

- In Vitro, `vitro-0.7.owl, vitroPublic.owl`
- In VIVO `education.owl personTypes.n3 agent.owl event.owl process.owl appControls-temp.n3 geo-political.owl publication.owl bfo-bridge.owl grant.owl relationship.owl bfo.owl linkSuppression.n3 relationshipAxioms.n3 classes-additional.owl location.owl research-resource-iao.owl clinical.owl object-properties.owl research-resource.owl contact-vcard.owl object-properties2.owl research.owl contact.owl object-properties3.owl role.owl data-properties.owl objectDomains.rdf sameAs.n3 dataDomains.rdf objectRanges.rdf service.owl dataset.owl ontologies.owl skos-vivo.owl date-time.owl orcid-interface.n3 teaching.owl dateTimeValuePrecision.owl other.owl vitro-0.7.owl documentStatus.owl outreach.owl vitroPublic.owl`
- Plus whatever ontology extensions you may have added

base Full

Source: a combination of base ABox and base TBox

inference ABox

Name: http://vitro.mannlib.cornell.edu/default/vitro-kb-inf

Source: named graph from the RDFService

inference TBox

Name: http://vitro.mannlib.cornell.edu/default/inferred-tbox

Source: named graph from the RDFService (memory-mapped)

inference Full

Source: a combination of inference ABox and inference TBox

union ABox

Source: a combination of base ABox and inference ABox

union TBox

Source: a combination of base TBox and inference TBox

union Full

Source: a combination of union ABox and union TBox

10.7.10 VIVO and the Solr search engine (*)

10.7.10.1 What is Solr?

Solr is an open-source, enterprise level search platform, available from Apache. It is based on the popular Lucene search engine. VIVO uses a standard instance of Solr, without modification. You can learn more about Solr at the Apache Solr home page[180].

VIVO maintains its data in a semantic triple-store. A triple-store is very well suited for expressing a complex, flexible web of data relationship. It is not very well suited for text-based searches. Solr provides fast searching with features like

- weighted results by field,
- searching by the stems of words, rather than exact matches,
- faceted search results,
- and much more.

Solr provides these features much more efficiently than a triple-store would.

Solr maintains its own index of data, which reflects the contents of the triple-store. As the data in VIVO changes, the contents of the Solr index must change also. In most cases this happens automatically, but not always. Sometimes the search index must be rebuilt to bring it into synchronization with the triple-store. See the section below called "How is the index kept up to date" (see page 0) for more information.

Solr is implemented as a self-contained web application, separate from VIVO. At most VIVO sites, Solr and VIVO run on the same machine, in the same instance of Tomcat, but this is not the only possible configuration. It is possible to put Solr in a different servlet container or even on a different computer from VIVO.

In a typical VIVO installation, Solr is hidden behind VIVO, and the users cannot access it directly. In general, they don't know that Solr exists as an application.

10.7.10.2 How does VIVO use Solr?

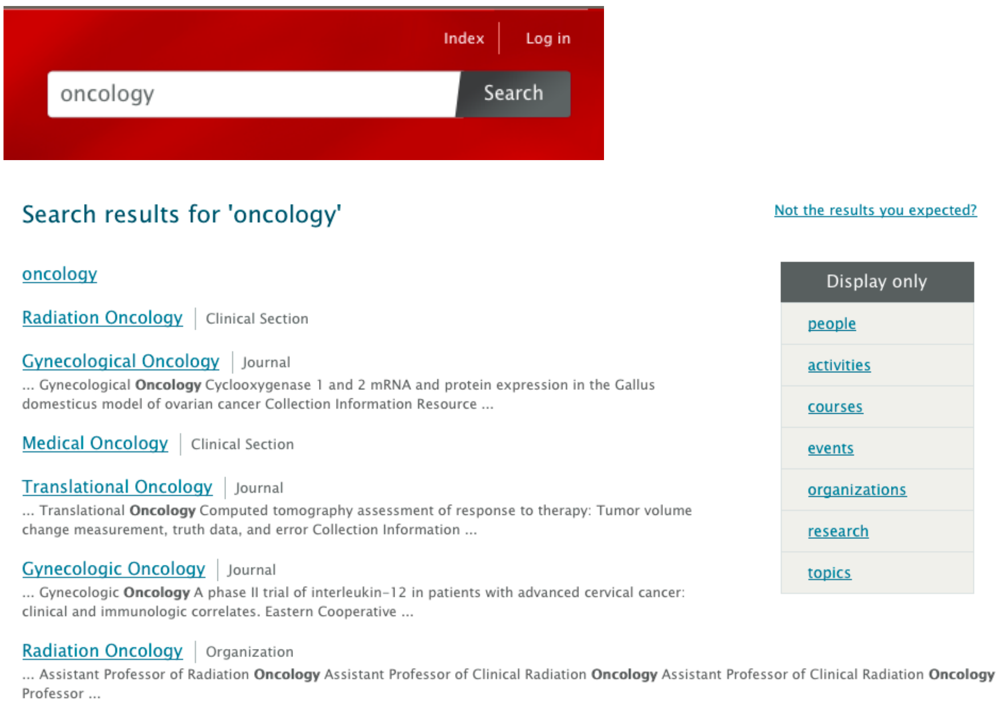
VIVO uses the Solr search engine in two ways:

- as a service to the end user,
- as a tool within the structure of the application.

Solr for the end user

Like many web sites, VIVO includes a search box on every page. The person using VIVO can type a search term, and see the results. This search is conducted by Solr, and the results are formatted and displayed by VIVO.

180 https://lucene.apache.org/solr/



Search results for 'oncology'

oncology

Radiation Oncology | Clinical Section

Gynecological Oncology | Journal
... Gynecological **Oncology** Cyclooxygenase 1 and 2 mRNA and protein expression in the Gallus domesticus model of ovarian cancer Collection Information Resource ...

Medical Oncology | Clinical Section

Translational Oncology | Journal
... Translational **Oncology** Computed tomography assessment of response to therapy: Tumor volume change measurement, truth data, and error Collection Information ...

Gynecologic Oncology | Journal
... Gynecologic **Oncology** A phase II trial of interleukin-12 in patients with advanced cervical cancer: clinical and immunologic correlates. Eastern Cooperative ...

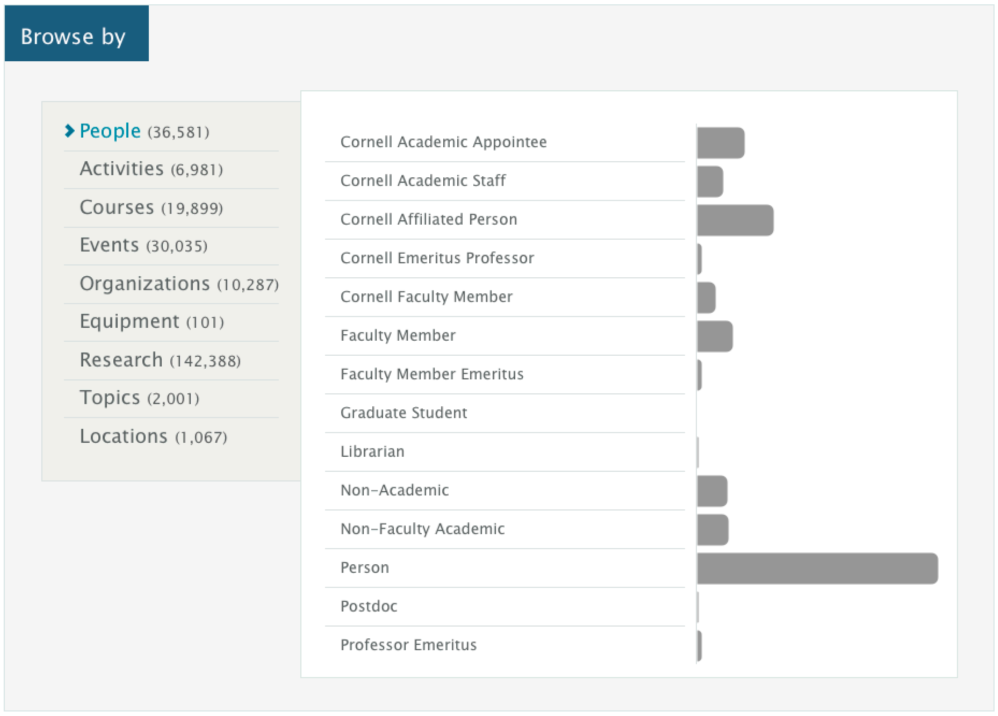
Radiation Oncology | Organization
... Assistant Professor of Radiation **Oncology** Assistant Professor of Clinical Radiation **Oncology** Assistant Professor of Clinical Radiation **Oncology** Professor ...

Solr allows for a "faceted" search, and VIVO displays the facets on the right side of the results page. These allow the user to filter the search results, showing only entries for people, or for organizations, etc.

Solr within VIVO

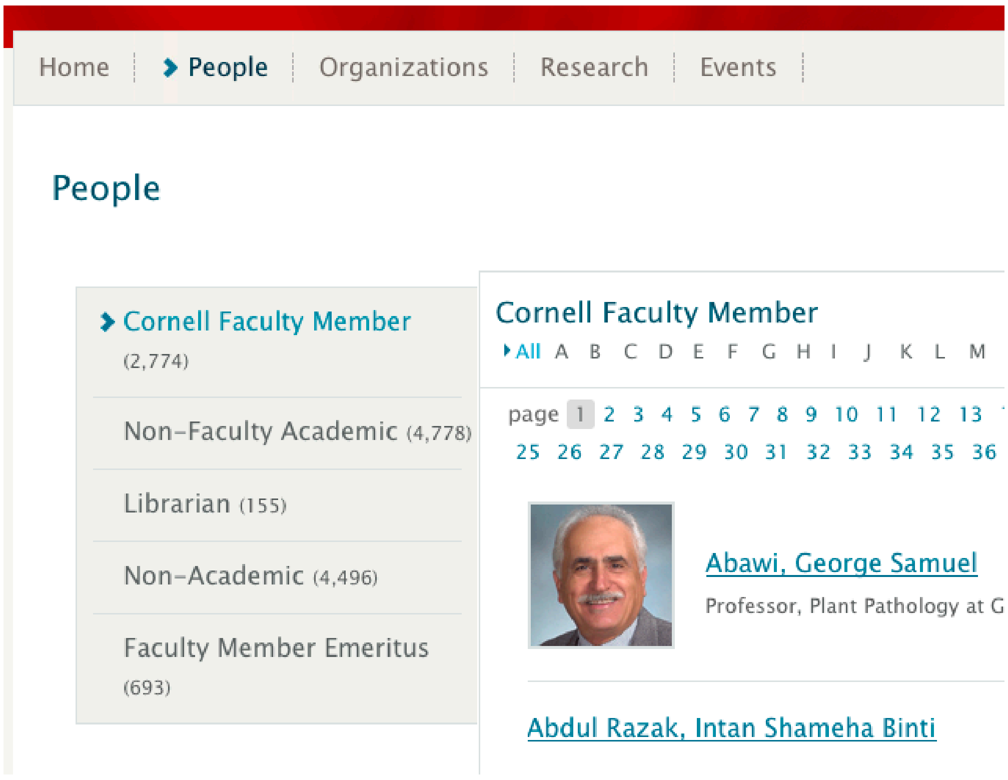
VIVO is based around an RDF triple-store, which holds all of its data. However, there are some tasks that a search engine can do much more quickly than a triple-store. Some of the fields in the Solr search index were put there specifically to help with these tasks.

For example, the browse area on the home page shows how many individuals VIVO holds for each class group.



VIVO could produce this data by issuing a SPARQL query against its data model. However, this would take several seconds for a large site, and we do not want the user to wait that long to see the home page. To avoid this delay, the class group of each individual is stored in the Solr record for that individual. Solr can count these fields very quickly, so VIVO issues a Solr query against the index, and displays the results on the home page.

Record counts on VIVO's index pages are obtained using the same type of Solr query.


Home
People
Organizations
Research
Events
People
Cornell Faculty Member (2,774)
Non-Faculty Academic (4,778)
Librarian (155)
Non-Academic (4,496)
Faculty Member Emeritus (693)
Cornell Faculty Member
All A B C D E F G H I J K L M
page 1 2 3 4 5 6 7 8 9 10 11 12 13
25 26 27 28 29 30 31 32 33 34 35 36
Abawi, George Samuel
Professor, Plant Pathology at G
Abdul Razak, Intan Shameha Binti

10.7.10.3 How is Solr created and configured?

The VIVO distribution includes a copy of Solr WAR file. When VIVO is installed the Solr WAR file is deployed to Tomcat as a web application.

The behavior of Solr depends extensively on its configuration files. These are stored in a directory that is called the Solr Home directory. In the standard VIVO installation, this is the `solr` sub-directory in the VIVO Home directory. When VIVO is installed, the Solr configuration files are copied to the Solr Home directory. When VIVO runs, the Solr search index is built inside the Solr Home directory.

If you are installing VIVO in a servlet container other than Tomcat, or if you are installing Solr in a separate servlet container, you will need to tell Solr how to find its home directory. See the instructions in Building a VIVO distribution for other servlet containers[181].

10.7.10.4 The search index

What is in the index?

The Solr search index contains one record for each Individual in VIVO, unless that individual is explicitly excluded from the index. Exclusions are usually made for individuals that represent "context nodes" in the VIVO data model.

For example, if a professor teaches a course, the search index will contain:

- a record for the professor
- a record for the course

The VIVO data model also contains an individual that represents this teaching activity. That individual will be excluded from the index, since users would almost certainly prefer to find the teacher or the course in their search results, rather than the concept that connects the two.

What is in each record?

Each record in the search index contains several fields (see the chart below). The most commonly used field is `alltext`, In the record for a faculty member, `alltext` will contain her name, the name of her department, the names of her classes, the names of her papers and grants, etc. So, if you search for "Carpenter", you might see results for people named Carpenter, people in the Carpentry department, people who have written papers about carpentry, or have worked on grants about carpentry. You would also see results for the department itself, for the papers, and for the grants.

Solr index fields, VIVO 1.6

DocId	nameRaw	PREFERRED_TITLE
URI	nameText	siteURL
ALLTEXT	nameLowercase	siteName

181 https://wiki.duraspace.org/display/VIVOARC/Building+a+VIVO+distribution+for+other+servlet+containers

ALLTEXTUNSTEMMED	nameLowercaseSingleValued	THUMBNAIL
classgroup	nameUnstemmed	THUMBNAIL_URL
type	nameStemmed	indexedTime
mostSpecificTypeURIs	acNameUntokenized	timestamp
BETA	acNameStemmed	etag
PROHIBITED_FROM_TEXT_RESULTS	NAME_PHONETIC	

When is the index updated?

During normal operation

When an individual is added, edited, or delete through VIVO's user interface, Solr is given the new information and the index is updated.

VIVO administrators may also make changes to the data using the Advanced Data Tools, which are accessible from the Site Administration page. These tools also pass the data changes to Solr, so the index is kept current with the data.

Finally, data can be modified using the The SPARQL Update API (see page 378). Again, Solr receives the changes and the index remains current.

On demand

Some tools, such as the VIVO Harvester, bypass VIVO and write directly to the data store. Solr is not notified when these tools are used, and the data becomes out of sync with the search index.

Other circumstances can cause issues with the search index. Perhaps a problem required you to restore your database to a backup, but you did not restore your search index at the same time. Perhaps you are developing a modification for VIVO, and you have emptied your database in order to test it. Perhaps VIVO crashed while data was being ingested.

In any of these circumstances, the solution is to log in to VIVO as an administrator, navigate to the `Site Administration` page and click on `Rebuild search index`.

The existing search index remains in place while the new index is being built. When the rebuild is complete, the new index replaces the old one, and the old index is deleted.

Customizing the index

In progress

- Building the record
- Exclusions

10.7.10.5 How does VIVO contact Solr?

> ⚠ In progress

- Need to tell VIVO how to contact Solr
  - Authorization tests, now obsolete
- VIVO may start before Solr does. Usually does.

10.7.11 Image storage

The uploaded image files are identified by a combination of URI and filename. The URI is used as the principal identifier so we don't need to worry about collisions if two people each upload an image named "image.jpg". The filename is retained so the user can use their browser to download their image from the system and it will be named as they expect it to be.

We wanted a way to store thousands of image files so they would not all be in the same directory. We took our inspiration from the PairTree[182] folks, and modified their algorithm to suit our needs. The general idea is to store files in a multi-layer directory structure based on the URI assigned to the file.

Let's consider a file with this information:

URI	http://vivo.mydomain.edu/individual/n3156
Filename	`lily1.jpg`

> ⚠ In this example, we assume that VIVO's home directory is at `/usr/local/vivo`.

We want to turn the URI into the directory path, but the URI contains prohibited characters. Using a PairTree-like character substitution, we might store it at this path:

```
/usr/local/vivo/uploads/file_storage_root/http+==vivo.mydomain.edu=individual=n3156/lily1.jpg
```

Using that scheme would mean that each file sits in its own directory under the storage root. At a large institution, there might be hundreds of thousands of directories under that root.

By breaking this into PairTree-like groupings, we insure that all files don't go into the same directory. Limiting to 3-character names will insure a maximum of about 30,000 files per directory. In practice, the number will be considerably smaller. So then it would look like this:

```
/usr/local/vivo/uploads/file_storage_root/htt/p+=/=vi/vo./myd/oma/in./edu/=in/div/idu/al=/n31/56/lily1.jpg
```

182 https://wiki.ucop.edu/display/Curation/PairTree

But almost all of our URIs will start with the same namespace, so the namespace just adds unnecessary and unhelpful depth to the directory tree. We assign a single-character prefix to that namespace, using the `file_storage_namespaces.properties` file in the uploads directory, like this:

```
a = http://vivo.mydomain.edu/individual/
```

And our URI now looks like this:

```
a~n3156
```

Which translates to:

```
/usr/local/vivo/uploads/file_storage_root/a~n/315/6/lily1.jpg
```

So what we hope we have implemented is a system where:

- Files are stored by URI and filename.
- File paths are constructed to limit the maximum number of files in a directory.
- "Illegal" characters in URIs or filenames will not cause problems.
    - even if a character is legal on the client and illegal on the server.
- Frequently-used namespaces on the URIs can be collapsed to short prefix sequences.
- URIs with unrecognized namespaces will not cause problems.

By the way, almost all of this is implemented in
`edu.cornell.mannlib.vitro.webapp.filestorage.backend.FileStorageHelper`
and illustrated in
`edu.cornell.mannlib.vitro.webapp.filestorage.backend.FileStorageHelperTest`

10.7.11.1 Access images after changing the default namespace

If you are moving images from one server to another, with no change in the URL, it should be sufficient to just move the VIVO home directory with no changes. VIVO will find `file_storage_namespaces.properties` and `file_storage_root` in `[home]/uploads`, and everything still works.

If you are changing to a new URL, I presume that you are changing to a new default namespace. Have you used the "Change Namespace of Resources" tool? (http://localhost:8082/vivo/ingest?action=renameResource)

So your file individual has changed from the old URI
`http://localhost:8082/vivo/individual/n187`
to the new URI
`http://logics.emap.fgv.br:8080/vivo/individual/n187`

However, `file_storage_namespaces.properties` does not know how to translate this new namespace.

One way to cope with this is to edit `file_storage_namespace.properties` accordingly, adding this line:
`b = http://logics.emap.fgv.br:8080/vivo/individual/`
and rename your
`[home]/uploads/file_storage_root/a~n`
directory to

`[home]/uploads/file_storage_root/b~n`

The new URI now translates to `b~n187`, and the file which is now stored at
`[home]/uploads/file_storage_root/b~n/187/servletrecuperafoto.jpeg`
is accessible by its new URI.

10.7.11.2 How are Images represented in the Model?

When an image file is uploaded via the GUI, the process is something along these lines:

- upload the image file, and store in a temporary location.
- ask the user for a cropping square to be used in producing the thumbnail.
- create a URI for the image file surrogate object, and a URI for the image file bytestream object.
- create a URI for the thumbnail surrogate object, and a URI for the thumbnail bytestream object.
- hand the image file bytestream URI and the temporary file to the File Storage system, which will create a permanent storage.
- generate a 115 by 115 JPEG thumbnail image from the main image and the cropping square.
- hand the thumbnail image stream and the thumbnail bytestream URI to the File Storage system, which will create a permanent storage.
- create a thumbnail bytestream object in the model.
- create a thumbnail surrogate object in the model, storing the filename of the thumbnail, the mime type of the thumbnail, and the URI of the thumbnail bytestream.
- create a main image bytestream object in the model.
- create a main image surrogate object in the model, storing the filename of the main image, the mime type of the main image, and the URI of the main image bytestream.
- link the main image surrogate object to the person object.

These are no more than a multitude of technical details, except: how do you find an appropriate region of the image to use as the thumbnail?

Generating the thumbnail itself can be quite problematic if the initial image is a GIF or PNG with transparency.

For an individual on my test installation (in N3, if I remember how to write it)

INDIVIDUAL

```
<http://vivo.mydomain.edu/individual/n1451>
   http://vitro.mannlib.cornell.edu/ns/vitro/public#mainImage
         http://vivo.mydomain.edu/individual/n1674.
```

IMAGE FILE SURROGATE

```
<http://vivo.mydomain.edu/individual/n1674>
    http://vitro.mannlib.cornell.edu/ns/vitro/public#thumbnailImage
            http://vivo.mydomain.edu/individual/n5863;
    http://vitro.mannlib.cornell.edu/ns/vitro/public#downloadLocation
            http://vivo.mydomain.edu/individual/n3156;
    http://vitro.mannlib.cornell.edu/ns/vitro/public#mimeType
            "image/jpeg";
    http://vitro.mannlib.cornell.edu/ns/vitro/public#filename
            "lily1.jpg";
    http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
            "2010-10-18T09:51:58";
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://vitro.mannlib.cornell.edu/ns/vitro/public#File;
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://www.w3.org/2002/07/owl#Thing.
```

IMAGE FILE BYTESTREAM

```
<http://vivo.mydomain.edu/individual/n3156>
    http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
            "2010-10-18T09:51:57";
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://vitro.mannlib.cornell.edu/ns/vitro/public#FileByteStream;
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://www.w3.org/2002/07/owl#Thing.
```

THUMBNAIL SURROGATE

```
<http://vivo.mydomain.edu/individual/n5863>
    http://vitro.mannlib.cornell.edu/ns/vitro/public#downloadLocation
            http://vivo.mydomain.edu/individual/n5889;
    http://vitro.mannlib.cornell.edu/ns/vitro/public#mimeType
            "image/jpeg";
    http://vitro.mannlib.cornell.edu/ns/vitro/public#filename
            "thumbnail_lily1.jpg";
    http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
            "2010-10-18T09:52:12";
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://vitro.mannlib.cornell.edu/ns/vitro/public#File;
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://www.w3.org/2002/07/owl#Thing.
```

THUMBNAIL BYTESTREAM

```
<http://vivo.mydomain.edu/individual/n5889>
    http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
            "2010-10-18T09:52:12";
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://vitro.mannlib.cornell.edu/ns/vitro/public#FileByteStream;
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://www.w3.org/2002/07/owl#Thing;
```

The file system looks something like this:

File storage properties file: /usr/local/vivo/uploads/file_storage_namespace.properties

```
a = http://vivo.mydomain.edu/individual/
```

Main image:

```
/usr/local/vivo/uploads/file_storage_root/a~n/315/6/lily1.jpg
```

Thumbnail:

```
/usr/local/vivo/uploads/file_storage_root/a~n/588/9/thumbnail_lily1.jpg
```

Note: The file storage system does "laundering" on the filenames, in order to allow files with special characters to be stored in a portable manner (e.g., Linux or Windows).

Jim

A summary from Eliza Chan

Excerpted from a message[183] in the vivo-dev-all archive, by Eliza Chan, dated 2010-11-04 16:08

As an experiment tested on localhost, when the pictures were uploaded using a "non-traditional" method, i.e. copying directly to the folder /usr/local/vivo/data/uploads/file_storage_root/a~n, the content under primary tab became blank (see attachment localhost_vivo_mainTab.tiff). Pictures did show up but only when the primary tab content was clicked (see attachment localhost_vivo_tabContent.tiff). The reason for copying directly to the folder was to save the work for doing manual upload of about 1000 photos.

The way it was done was as follows:

1. Create RDF for images and add to the VIVO site, e.g.

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/cwid-gwa2001">
<j.2:mainImage rdf:resource="http://localhost:8080/vivo/individual/mainImage-gwa2001"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
<rdf:type rdf:resource="http://vivoweb.org/ontology/core#FacultyMember"/>
<rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Agent"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/mainImage-gwa2001">
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<j.2:downloadLocation rdf:resource="http://localhost:8080/vivo/individual/n1229119954939"/>
<j.2:thumbnailImage rdf:resource="http://localhost:8080/vivo/individual/thumbnailImage-gwa2001"/>
<j.5:modTime xml:lang="en">2010-11-04T10:44:04</j.5:modTime>
<j.2:mimeType xml:lang="en">image/jpeg</j.2:mimeType>
<j.2:filename xml:lang="en">_main_image_gwa2001.jpg</j.2:filename>
<rdf:type rdf:resource="http://vitro.mannlib.cornell.edu/ns/vitro/public#File"/>
</rdf:Description>
```

183 https://sourceforge.net/mailarchive/message.php?msg_id=26544669

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/n1229119954939">
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<j.5:modTime xml:lang="en">2010-11-04T10:44:04</j.5:modTime>
<rdf:type rdf:resource="http://vitro.mannlib.cornell.edu/ns/vitro/public#FileByteStream"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/thumbnailImage-gwa2001">
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<j.2:downloadLocation rdf:resource="http://localhost:8080/vivo/individual/n12291199549391"/>
<j.5:modTime xml:lang="en">2010-11-04T10:44:04</j.5:modTime>
<j.2:mimeType xml:lang="en">image/jpeg</j.2:mimeType>
<j.2:filename xml:lang="en">gwa2001.jpg</j.2:filename>
<rdf:type rdf:resource="http://vitro.mannlib.cornell.edu/ns/vitro/public#File"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://localhost:8080/vivo/individual/n12291199549391">
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<j.5:modTime xml:lang="en">2010-11-04T10:44:04</j.5:modTime>
<rdf:type rdf:resource="http://vitro.mannlib.cornell.edu/ns/vitro/public#FileByteStream"/>
</rdf:Description>
```

2. Copy images to the following folders:

```
/usr/local/vivo/data/uploads/file_storage_root/a~n/122/911/995/493/9/_main_image_gwa2001.jpg
/usr/local/vivo/data/uploads/file_storage_root/a~n/122/911/995/493/91/gwa2001.jpg
```

Update on "alias URL" and "directDownloadUrl" property

You can retrieve an image file by asking for the Individual page of its FileByteStream. For example,

```
http://localhost:8080/vivo/individual/n4898
```

VIVO will see that this particular individual is a FileByteStream, and will redirect your browser to the "alias URL" for that image. In this case:

```
http://localhost:8080/vivo/file/n4898/john_doe.jpg
```

This redirection means that the image shown in your browser has a name that you will recognize, with an appropriate file type. If you choose "Save Image" in your browser, the default filename will be suitable for the image.

However, this redirection implies additional overhead. Pages local to VIVO calculated the alias URL and used it as the "src" property on the image, avoiding the redirection. But because the "alias URL" was not present in the RDF, it was not available to external applications, which resulted in excessive load times for pages that displayed dozens of images.

The directDownloadUrl" property of FileByteStream objects contains the "alias URL", is created when the image is ingested, and is used both by VIVO and by external applications when displaying images.

Accordingly, the FileByteStream examples shown above must now look like this instead:

IMAGE FILE BYTESTREAM

```
 <http://vivo.mydomain.edu/individual/n3156>
    http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
            "2010-10-18T09:51:57";
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://vitro.mannlib.cornell.edu/ns/vitro/public#FileByteStream;
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://www.w3.org/2002/07/owl#Thing.
    http://vitro.mannlib.cornell.edu/ns/vitro/public#directDownloadUrl
            "/file/n3156/lily1.jpg"
```

THUMBNAIL BYTESTREAM

```
 <http://vivo.mydomain.edu/individual/n5889>
    http://vitro.mannlib.cornell.edu/ns/vitro/0.7#modTime
            "2010-10-18T09:52:12";
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://vitro.mannlib.cornell.edu/ns/vitro/public#FileByteStream;
    http://www.w3.org/1999/02/22-rdf-syntax-ns#type
            http://www.w3.org/2002/07/owl#Thing;
```

```
http://vitro.mannlib.cornell.edu/ns/vitro/public#directDownloadUrl
    "/file/n5889/thumbnail_lily1.jpg"
```

10.8 URL Reference

10.8.1 Overview

VIVO has several pages that can be reached by adding one of the words below to the end of your VIVO URL. For example, if the URL of your VIVO home page is `http://vivo.myschool.edu`, then you can access a page regarding revision information by accessing `http://vivo.myschool.edu/revisionInfo`.

10.8.2 sitemap.xml

See the XML VIVO generates for your site's sitemap.

10.8.3 `SearchIndex`

Show search index status and access to rebuild the VIVO search index

10.8.4 `RecomputeInferences`

Review all triples in the triple store and add inferences to the inference graph as needed.

10.8.5 `revisionInfo`

Show a page of version information including date and time of most recent build, as shown below.

Revision Information

Levels:

name	release	revision
VIVO	1.9.1	066d5a0

Build date:

Thursday, October 6, 2016 5:00:03 PM EDT

10.8.6 `freemarkersamples`

Displays a page of Freemarker widget results. The template for this page can be found here vitro-core/webapp/web/templates/freemarker/body/samples.ftl.

10.8.7 `vivosolr`

Display the VIVO Solr search index control panel

10.9 VIVO APIs

The VIVO APIs are HTTP end-points that can be used to read or write data, or to manage VIVO's operation. They have no user interface, and are intended to be called by external applications that are cooperating with VIVO.

The end-points include:

Public Services	• **available without restriction** • **provide filtered results, allowing restrictions on data**
Linked Open Data (see page 394)	Information about an individual, its types, its data values, incoming and outgoing links.
ListRDF (see page 403)	Lists of individuals that belong to a particular class in the ontology. For example, a list of all People, or all Organizations.

Access Controlled Services	• **require account credentials on each request** • **credentials are for an internal VIVO with sufficient authorization** • **results are not filtered, and may return data that should be kept private**
SPARQL Query API (see page 406)	Submit a SPARQL query to get information from VIVO. Supports `SELECT`, `ASK`, `CONSTRUCT`, and `DESCRIBE` query types.

Access Controlled Services	• **require account credentials on each request** • **credentials are for an internal VIVO with sufficient authorization** • **results are not filtered, and may return data that should be kept private**
SPARQL Update API (see page 411)	Submit a SPARQL query to `INSERT` new triples or `DELETE` existing triples. Also, `LOAD` triples from a web-accessible file.
Search Indexing API (see page 416)	Submit a list of URIs that may have stale data in the search index. The search data for each of these URIs will be rebuilt.

10.9.1 Linked Open Data - requests and responses

10.9.1.1 Overview

Linked Open Data is one of the fundamental concepts of the Semantic Web. It consists of asking a server for the RDF relating to an individual. If the response includes object properties that link to other individuals, those individuals can be queried also. For more information on Linked Open Data, see Concept: Linked Data (see page 394).

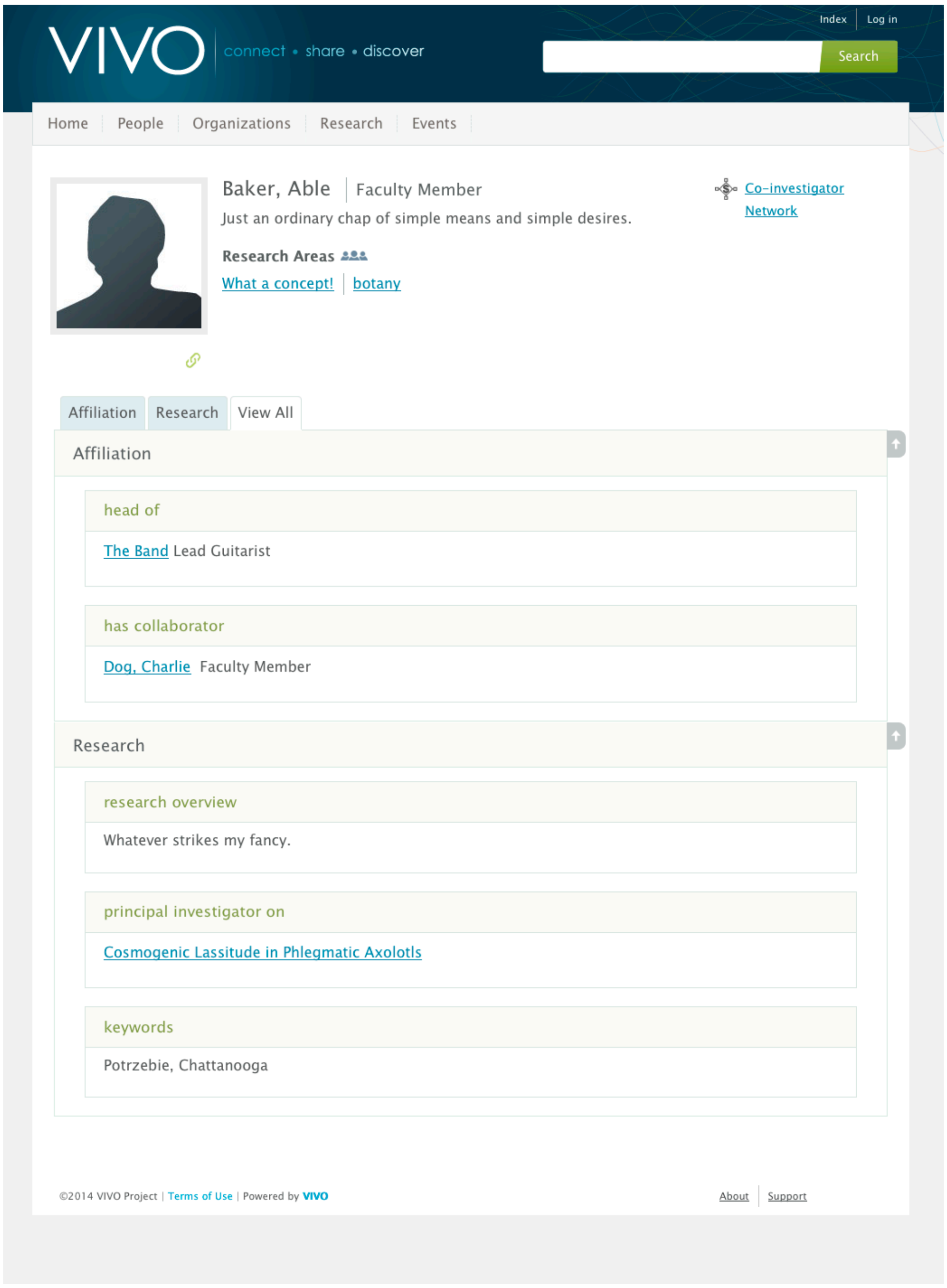
VIVO accepts standard requests for Linked Open Data and some non-standard ones. The contents of the response are in accordance with those suggested by the in their tutorial How to Publish Linked Data on the Web[184].

VIVO will provide Linked Open Data in several formats. The semantic content remains the same; only the syntax differs among formats.

An example

The examples on this page are based on a fictitious individual named "Able Baker", with a URI of `http://vivo.mydomain.edu/individual/n3639`. To keep the examples simple, this person has just a few items in his VIVO profile. His profile page looks like this:

184 http://wifo5-03.informatik.uni-mannheim.de/bizer/pub/LinkedDataTutorial/



Baker, Able | Faculty Member

Just an ordinary chap of simple means and simple desires.

Research Areas

What a concept! | botany

Affiliation

head of

The Band Lead Guitarist

has collaborator

Dog, Charlie Faculty Member

Research

research overview

Whatever strikes my fancy.

principal investigator on

Cosmogenic Lassitude in Phlegmatic Axolotls

keywords

Potrzebie, Chattanooga

©2014 VIVO Project | Terms of Use | Powered by VIVO

About | Support

10.9.1.2 Requesting Linked Open Data from VIVO

Available formats

VIVO will serve Linked Open Data in these formats:

- RDF/XML[185]
- N3[186]
- Turtle[187]
- JSON-LD[188]

Specifications for each of the formats are provided by the World Wide Web Consortium (W3C).

Types of requests

The standard way of requesting Linked Open data is an HTTP request to the URI of the individual in question, with the `Accept` header on the request indicating the desired format. If there is no `Accept` header, it is assumed to be `text/html`, and the standard profile page is returned.

URL	**Accept header**	**Response format**	**Response MIME type**
`http://vivo.mydomain.edu/individual/n3639`	`application/rdf+xml`	RDF/XML	`application/rdf+xml`
`http://vivo.mydomain.edu/individual/n3639`	`text/n3`	N3	`text/n3`
`http://vivo.mydomain.edu/individual/n3639`	`text/turtle`	Turtle	`text/turtle`
`http://vivo.mydomain.edu/individual/n3639`	`application/json`	JSON-LD	`application/json`

The different responses may also be explicitly requested by URL. In fact, the requests listed above will simply redirect the browser to these URLs:

URL	**Response format**	**Response MIME type**
`http://vivo.mydomain.edu/individual/n3639/n3639.rdf`	RDF/XML	`application/rdf+xml`
`http://vivo.mydomain.edu/individual/n3639/n3639.n3`	N3	`text/n3`
`http://vivo.mydomain.edu/individual/n3639/n3639.ttl`	Turtle	`text/turtle`

185 http://www.w3.org/TR/REC-rdf-syntax/
186 http://www.w3.org/TeamSubmission/n3/
187 http://www.w3.org/TeamSubmission/turtle/
188 http://www.w3.org/TR/json-ld/

URL	Response format	Response MIME type
`http://vivo.mydomain.edu/individual/n3639/n3639.jsonld`	JSON-LD	`application/json`

Finally, VIVO allows you to request Linked Open Data in a way that is not specified by the standard. You can make an HTTP GET request to the URI of the individual, and include a `format` parameter that specifies the format of the response.

URL	Response format	Response MIME type
`http://vivo.mydomain.edu/individual/n3639?format=rdfxml`	RDF/XML	`application/rdf+xml`
`http://vivo.mydomain.edu/individual/n3639?format=n3`	N3	`text/n3`
`http://vivo.mydomain.edu/individual/n3639?format=ttl`	Turtle	`text/turtle`
`http://vivo.mydomain.edu/individual/n3639?format=jsonld`	JSON-LD	`application/json`

10.9.1.3 What is included in the response?

When you get request the public RDF about an individual in VIVO, the result is a set of RDF statements, or triples. These triples state:

- The data properties of the individual.
- The object properties that relate this individual to other individuals.
- The object properties of other individuals that relate to this individual
- The labels and types of these related individuals.
- Some triples that describe the RDF document itself.

This statement over-simplifies slightly. In VIVO, object properties and data properties can be public, or restricted to some extent. The RDF for an individual will contain only public properties.

An example response

Here is the RDF produced for the example, in N3 format.

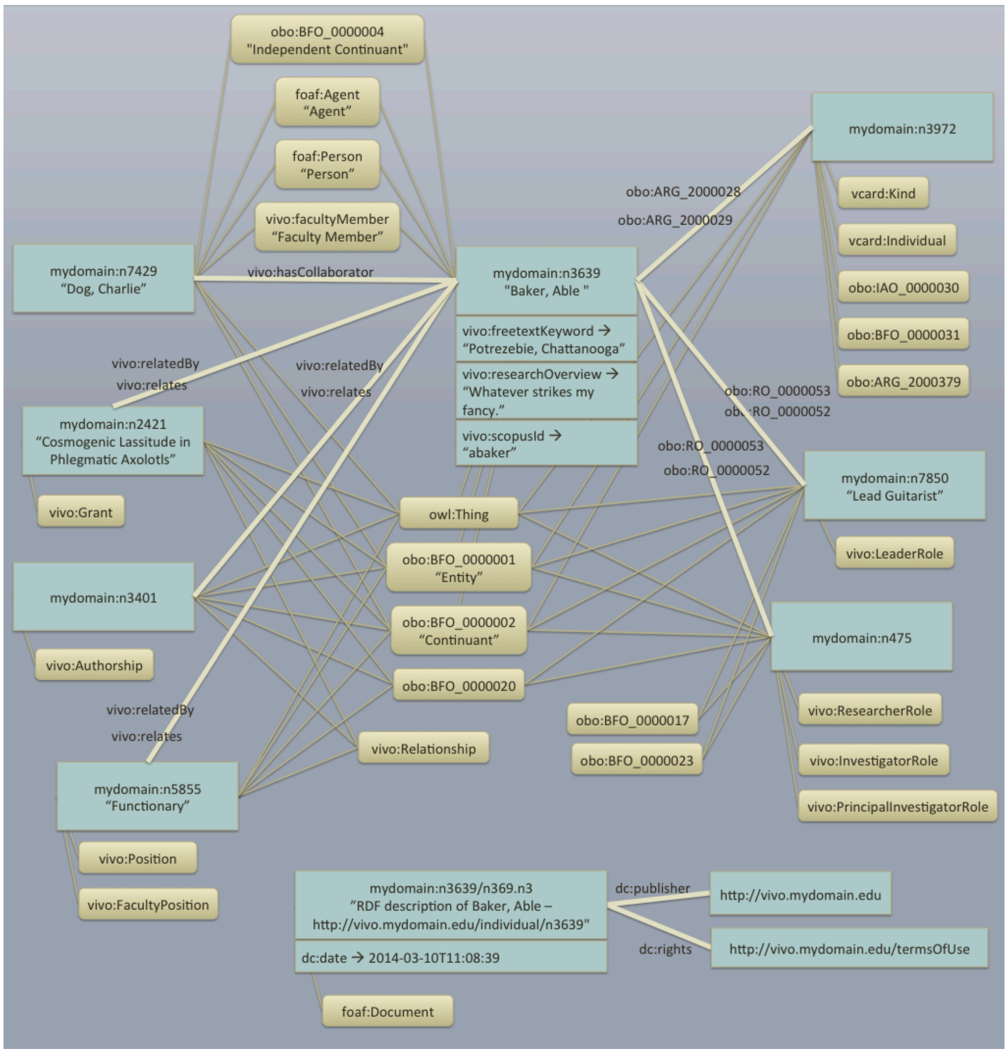
```
@prefix foaf:     <http://xmlns.com/foaf/0.1/> .
@prefix vcard:    <http://www.w3.org/2006/vcard/ns#> .
@prefix obo:      <http://purl.obolibrary.org/obo/> .
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .
@prefix vitro:    <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#> .
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix owl:      <http://www.w3.org/2002/07/owl#> .
```

```
@prefix vivo:    <http://vivoweb.org/ontology/core#> .

<http://vivo.mydomain.edu/individual/n3639>
      a       vivo:FacultyMember ,
              foaf:Person ,
              owl:Thing ,
              foaf:Agent ,
              obo:BFO_0000002 ,
              obo:BFO_0000001 ,
              obo:BFO_0000004 ;
      rdfs:label "Baker, Able "^^xsd:string ;
      obo:ARG_2000028 <http://vivo.mydomain.edu/individual/n3972> ;
      obo:RO_0000053 <http://vivo.mydomain.edu/individual/n475> ,
                     <http://vivo.mydomain.edu/individual/n7850> ;
      vitro:mostSpecificType
              vivo:FacultyMember ;
      vivo:freetextKeyword
              "Potrezebie, Chattanooga" ;
      vivo:hasCollaborator
              <http://vivo.mydomain.edu/individual/n7429> ;
      vivo:relatedBy <http://vivo.mydomain.edu/individual/n3401> ,
                     <http://vivo.mydomain.edu/individual/n5855> ,
                     <http://vivo.mydomain.edu/individual/n2421> ;
      vivo:researchOverview
              "Whatever strikes my fancy." ;
      vivo:scopusId "abaker" .

<http://vivo.mydomain.edu/individual/n3972>
      a       vcard:Kind ,
              obo:BFO_0000031 ,
              owl:Thing ,
              obo:ARG_2000379 ,
              obo:IAO_0000030 ,
              obo:BFO_0000002 ,
              obo:BFO_0000001 ,
              vcard:Individual ;
      obo:ARG_2000029 <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n475>
      a       owl:Thing ,
              obo:BFO_0000023 ,
              vivo:InvestigatorRole ,
              obo:BFO_0000002 ,
              obo:BFO_0000017 ,
              vivo:PrincipalInvestigatorRole ,
              obo:BFO_0000020 ,
              obo:BFO_0000001 ,
              vivo:ResearcherRole ;
      obo:RO_0000052 <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n7850>
      a       owl:Thing ,
              obo:BFO_0000023 ,
              obo:BFO_0000017 ,
              obo:BFO_0000002 ,
              obo:BFO_0000020 ,
              obo:BFO_0000001 ,
              vivo:LeaderRole ;
      rdfs:label "Lead Guitarist"^^xsd:string ;
      obo:RO_0000052 <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n7429>
      a       foaf:Person ,
              vivo:FacultyMember ,
              foaf:Agent ,
              owl:Thing ,
              obo:BFO_0000002 ,
              obo:BFO_0000001 ,
              obo:BFO_0000004 ;
      rdfs:label "Dog, Charlie" .

<http://vivo.mydomain.edu/individual/n3401>
      a       owl:Thing ,
              vivo:Relationship ,
              obo:BFO_0000002 ,
              obo:BFO_0000020 ,
              obo:BFO_0000001 ,
              vivo:Authorship ;
      vivo:relates <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n5855>
      a       vivo:FacultyPosition ,
              owl:Thing ,
              vivo:Relationship ,
              obo:BFO_0000002 ,
              obo:BFO_0000020 ,
              obo:BFO_0000001 ,
              vivo:Position ;
      rdfs:label "Functionary"^^xsd:string ;
      vivo:relates <http://vivo.mydomain.edu/individual/n3639> .

<http://vivo.mydomain.edu/individual/n2421>
      a       owl:Thing ,
              vivo:Relationship ,
              obo:BFO_0000002 ,
              obo:BFO_0000020 ,
              obo:BFO_0000001 ,
              vivo:Grant ;
      rdfs:label "Cosmogenic Lassitude in Phlegmatic Axolotls" ;
      vivo:relates <http://vivo.mydomain.edu/individual/n3639> .

obo:BFO_0000001
      a       owl:Class ;
      rdfs:label "Entity" .

obo:BFO_0000002
      a       owl:Class ;
      rdfs:label "Continuant" .
```

```
obo:BFO_0000004
      a       owl:Class ;
      rdfs:label "Independent Continuant"@en-US .

vivo:FacultyMember
      a       owl:Class ;
      rdfs:label "Faculty Member"@en-US .

foaf:Person
      a       owl:Class ;
      rdfs:label "Person"@en-US .

foaf:Agent
      a       owl:Class ;
      rdfs:label "Agent"@en-US .

owl:Thing
      a       owl:Class .

<http://vivo.mydomain.edu/individual/n3639/n3639.n3>
      a       foaf:Document ;
      rdfs:label "RDF description of Baker, Able  - http://vivo.mydomain.edu/individual/n3639" ;
      <http://purl.org/dc/elements/1.1/date> "2014-03-10T11:08:39"^^xsd:dateTime ;
      <http://purl.org/dc/elements/1.1/publisher> <http://vivo.mydomain.edu> ;
      <http://purl.org/dc/elements/1.1/rights> <http://vivo.mydomain.edu/termsOfUse> .
```

A graphic summary

The RDF can be expressed graphically like this:


obo:BFO_0000004 "Independent Continuant"
foaf:Agent "Agent"
foaf:Person "Person"
vivo:facultyMember "Faculty Member"
mydomain:n7429 "Dog, Charlie"
vivo:hasCollaborator
mydomain:n3639 "Baker, Able "
vivo:freetextKeyword → "Potrezebie, Chattanooga"
vivo:researchOverview → "Whatever strikes my fancy."
vivo:scopusId → "abaker"
obo:ARG_2000028
obo:ARG_2000029
mydomain:n3972
vcard:Kind
vcard:Individual
obo:IAO_0000030
obo:BFO_0000031
obo:ARG_2000379
obo:RO_0000053
obo:RO_0000052
vivo:relatedBy
vivo:relates
mydomain:n2421 "Cosmogenic Lassitude in Phlegmatic Axolotls"
vivo:Grant
owl:Thing
obo:BFO_0000001 "Entity"
obo:BFO_0000002 "Continuant"
obo:BFO_0000020
mydomain:n7850 "Lead Guitarist"
vivo:LeaderRole
mydomain:n3401
vivo:Authorship
mydomain:n475
obo:BFO_0000017
obo:BFO_0000023
vivo:Relationship
vivo:ResearcherRole
vivo:InvestigatorRole
vivo:PrincipalInvestigatorRole
mydomain:n5855 "Functionary"
vivo:Position
vivo:FacultyPosition
mydomain:n3639/n369.n3 "RDF description of Baker, Able – http://vivo.mydomain.edu/individual/n3639"
dc:date → 2014-03-10T11:08:39
dc:publisher
dc:rights
http://vivo.mydomain.edu
http://vivo.mydomain.edu/termsOfUse
foaf:Document

10.9.1.4 Restricting properties

Editing the property

You can exclude a property from Linked Open Data, or include it, by editing the property within VIVO. Perhaps the easiest way to edit a property is to log in as a VIVO administrator, navigate to an individual's profile page, and turn on the verbose display:

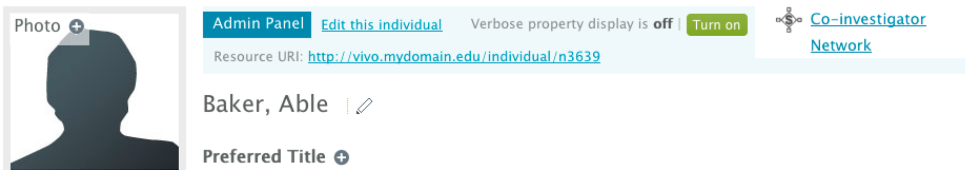
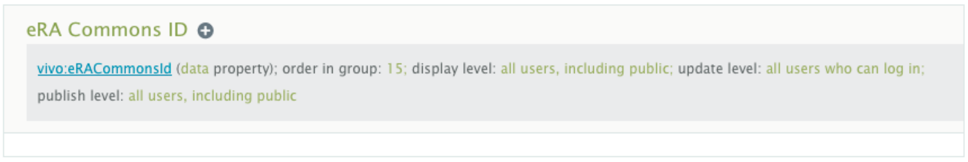


Photo
Admin Panel
Edit this individual
Verbose property display is off | Turn on
Co-investigator Network
Resource URI: http://vivo.mydomain.edu/individual/n3639
Baker, Able
Preferred Title

Once the verbose display is turned on, scroll through the profile page to find the property you are interested in. You can see what it's current restriction levels are for display, update, and publishing. You also have a link to the control panel for that property:

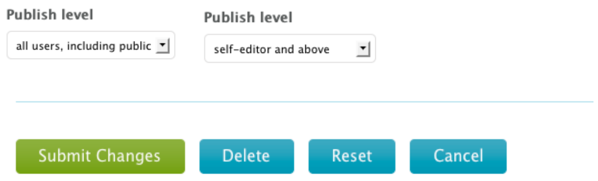

eRA Commons ID

vivo:eRACommonsId (data property); order in group: 15; display level: all users, including public; update level: all users who can log in; publish level: all users, including public

Note that all Linked Open Data requests are treated as public, so any setting other than `all users, including public` will exclude the property.

Navigate to the control panel for the property, and then to the editing form for the property.

Set the `Publish level` as you like, and submit the changes.


Publish level: all users, including public

Publish level: self-editor and above

Setting triples in the display model

Properties in VIVO can be restricted from Linked Open Data, by attaching the `vitro:hiddenFromPublishBelowRoleLevelAnnot` annotation to the property.

For example, this triple in VIVO's display model would mean that the `eRACommonsId` property would not be published in Linked Open Data

```
<http://vivoweb.org/ontology/core#eRACommonsId>
      <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#hiddenFromPublishBelowRoleLevelAnnot>
              <http://vitro.mannlib.cornell.edu/ns/vitro/role#nobody> .
```

Note, however, that the standard VIVO distribution includes this triple in the display model:

```
<http://vivoweb.org/ontology/core#:eRACommonsId>
      <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#:hiddenFromPublishBelowRoleLevelAnnot>
              <http://vitro.mannlib.cornell.edu/ns/vitro/role#public> .
```

You would need to remove this triple in order for the more restrictive triple to take effect.

An exception to the restrictions

VIVO uses the same permissions model to restrict Linked Open Data that it uses to restrict displays or updates. So if you are logged in to VIVO as the root user, and you request Linked Open Data, no restrictions would be applied.

This is consistent with VIVO's authorization model.

An external application could take advantage of this fact to obtain full RDF about individuals. Since there is no authorization parameter on the Linked Open Data request, the client application would need to begin by logging in to VIVO as an administrator, and then retain the session cookie to submit with subsequent requests.

10.9.1.5 Error handling

If you ask for Linked Open Data for a non-existent individual, regardless of the form you use, VIVO will return a response code of `404 not found`.

If you ask for an unsupported format, either in the `Accept` header or the `format` parameter, VIVO will treat your request as a request for HTML, and will return the standard profile page for the individual. The response code will be `200 OK`.

10.9.2 ListRDF API

10.9.2.1 Overview

Purpose

Permits external applications to obtain a list of all Individuals in VIVO that belong to a specified class. For example, a list of all Persons, or a list of all Organizations.

This API complements the Linked Open Data API. The Linked Open Data standard describes a way to get data about any Individual, but it does not provide a way to get a list of Individuals to begin with.

Filtered results

The results of this query is filtered by the same VIVO policies that control Linked Open Data. Individuals may be omitted from the results, if those policies restrict access to those Individuals.

Use Cases

Harvesting data from VIVO

Data in VIVO is available to other applications via Linked Open Data - requests and responses[189]. A list of Individuals from this API may provide a starting point for such applications.

Multi-site search index

If an external application chooses to build a compendium from several VIVO sites, it will need to know what Individuals are present in each site.

10.9.2.2 Specification

URL

`[vivo]/listrdf`

Examples:

```
http://vivo.cornell.edu/listrdf
```

```
http://localhost:8080/vivo/listrdf
```

HTTP Method

The API supports HTTP GET or POST calls.

Parameters

name	value
`vclass`	the URI of the class to be listed.

Response Codes

Code	Reason
`200 OK`	SPARQL query was successful.
`400 Bad Request`	HTTP request did not include a `vclass` parameter.

189 https://wiki.duraspace.org/display/VIVOARC/Linked+Open+Data+-+requests+and+responses

Code	Reason
`406 Not Acceptable`	The Accept header does not include any available content types.
`500 Internal Server Error`	VIVO could not execute the request; internal code threw an exception.

Content of the response

The response will contain RDF triples. Each triple asserts that an Individual is an instance of the requested class.

Available content types

The request may include an `Accept` header, to specify the preferred content type of the response. If no `Accept` header is provided, the preferred content type is assumed to be `text/plain`.

MIME type in the Accept header	Response format	Format description
`text/plain`	N-Triples	http://www.w3.org/2001/sw/RDFCore/ntriples/
`application/rdf+xml`	RDF/XML	http://www.w3.org/TR/rdf-syntax-grammar/
`text/n3`	N3	http://www.w3.org/TeamSubmission/n3/
`text/turtle`	Turtle	http://www.w3.org/TeamSubmission/turtle/
`application/json`	JSON-LD	http://www.w3.org/TR/json-ld/

10.9.2.3 Examples

These examples use the UNIX `curl` command to issue queries to the API.

Continents as N-Triples example

This example requests a list of `vivo:Continent` Individuals, in N-triples format.

```
curl -i -d 'vclass=http://vivoweb.org/ontology/core#Continent' -H 'Accept:text/plain' 'http://localhost:
8080/vivo/listrdf'
```

The response looks like this:

```
<http://aims.fao.org/aos/geopolitical.owl#Africa> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#Europe> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#Antarctica> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#Continent> .
```

```
<http://aims.fao.org/aos/geopolitical.owl#northern_America> <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> <http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#South_America> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#Australia_and_New_Zealand> <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://vivoweb.org/ontology/core#Continent> .
<http://aims.fao.org/aos/geopolitical.owl#Asia> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://
vivoweb.org/ontology/core#Continent> .
```

Faculty Members as JSON-LD example

This example requests a list of `vivo:FacultyMember` Individuals, in JSON.

```
curl -i -d 'vclass=http://vivoweb.org/ontology/core#FacultyMember' -H 'Accept:application/json' 'http://
localhost:8080/vivo/listrdf'
```

The response (for a very small VIVO) looks like this:

```
[{"@id":"http://vivo.mydomain.edu/individual/n4295","@type":["http://vivoweb.org/ontology/
core#FacultyMember"]},{"@id":"http://vivo.mydomain.edu/individual/n5056","@type":["http://vivoweb.org/
ontology/core#FacultyMember"]},{"@id":"http://vivo.mydomain.edu/individual/n7630","@type":["http://
vivoweb.org/ontology/core#FacultyMember"]},{"@id":"http://vivoweb.org/ontology/core#FacultyMember"}]
```

10.9.3 SPARQL Query API

10.9.3.1 Purpose

Permits external applications to obtain data from the VIVO data model.

The results of the queries are not filtered, so access to the service should remain restricted if the VIVO instance contains any data which should remain private. Queries can be performed against the entire data model, or against specific graphs. .

⚠ By default, the SPARQL Query API is disabled in VIVO, for security reasons. See Enabling the API (see page 0)

10.9.3.2 Use Cases

Reusing data from VIVO

Data in VIVO is available to other applications via Linked Open Data - requests and responses[190]. But some applications may work better with the sort of data sets that can be obtained from SPARQL queries.

Writing a VIVO "face" application

Various VIVO sites have written applications, in Drupal or other such frameworks, that display data from VIVO, and allow the user to edit their data. This API, used in conjunction with SPARQL Update API (see page 411), allows such an application to freely read or modify VIVO data.

10.9.3.3 Specification

URL

`[vivo]/api/sparqlQuery`

Examples:

```
http://vivo.cornell.edu/api/sparqlQuery
```

```
http://localhost:8080/vivo/api/sparqlQuery
```

HTTP Method

The API supports HTTP GET or POST calls.

Parameters

name	**value**
`email`	the email address of a VIVO administrator account

190 https://wiki.duraspace.org/display/VIVOARC/Linked+Open+Data+-+requests+and+responses

name	value
`password`	the password of the VIVO administrator account
`query`	A SPARQL query

The syntax of the SPARQL query is described on the World Wide Web Consortium site at http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

Response Codes

Code	Reason
`200 OK`	SPARQL query was successful.
`400 Bad Request`	HTTP request did not include a `query` parameter.
	The SPARQL query was syntactically incorrect.
	The type of the SPARQL query was not `SELECT`, `ASK`, `CONSTRUCT`, or `DESCRIBE`
`403 Forbidden`	HTTP request did not include an `email` parameter.
	HTTP request did not include a `password` parameter.
	The combination of `email` and `password` is not valid.
	The selected VIVO account is not authorized to use the SPARQL Query API.
`406 Not Acceptable`	The Accept header does not include any available content types.
`500 Internal Server Error`	VIVO could not execute the request; internal code threw an exception.

Available content types

The request may include an `Accept` header, to specify the preferred content type of the response. If no `Accept` header is provided, the preferred content type is assumed to be `text/plain`.

For `SELECT` or `ASK` queries

`SELECT` queries return rows of results, and each row may include an arbitrary number of values, depending on the query.

`ASK` queries return a single result, which is either `true` or `false`.

MIME type in the Accept header	Response format	Format description
`text/plain`	text	

MIME type in the Accept header	Response format	Format description
`text/csv`	CSV	http://www.w3.org/TR/2013/REC-sparql11-results-csv-tsv-20130321
`text/tab-separated-values`	TSV	
`application/sparql-results+xml`	XML	http://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321
`application/sparql-results+json`	JSON	http://www.w3.org/TR/2013/REC-sparql11-results-json-20130321

For `CONSTRUCT` or DESCRIBE queries

`CONSTRUCT` and `DESCRIBE` queries return RDF.

MIME type in the Accept header	Response format	Format description
`text/plain`	N-Triples	http://www.w3.org/2001/sw/RDFCore/ntriples/
`application/rdf+xml`	RDF/XML	http://www.w3.org/TR/rdf-syntax-grammar/
`text/n3`	N3	http://www.w3.org/TeamSubmission/n3/
`text/turtle`	Turtle	http://www.w3.org/TeamSubmission/turtle/
`application/json`	JSON-LD	http://www.w3.org/TR/json-ld/

Limitation

Queries can be performed against specific graphs. However, the graphs that hold application data are not accessible to the API. "Application data" means data that controls the functioning of the VIVO application, such as user accounts, page definitions, or display parameters.

10.9.3.4 Examples

These examples use the UNIX `curl` command to issue queries to the API.

SELECT to JSON example

This example reads 5 arbitrary triples from the data model, returning the result as JSON.

```
curl -i -d 'email=testAdmin@mydomain.edu' -d 'password=Password' -d 'query=SELECT ?s ?p ?o WHERE {?s ?p ?o}
LIMIT 5' -H 'Accept: application/sparql-results+json' 'http://localhost:8080/vivo/api/sparqlQuery'
```

The response looks like this:

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "bnode" , "value": "b0" } ,
        "p": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#rest" } ,
        "o": { "type": "bnode" , "value": "b1" }
      } ,
      {
        "s": { "type": "bnode" , "value": "b0" } ,
        "p": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#first" } ,
        "o": { "type": "uri" , "value": "http://purl.obolibrary.org/obo/ERO_0000006" }
      } ,
      {
        "s": { "type": "bnode" , "value": "b2" } ,
        "p": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#rest" } ,
        "o": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil" }
      } ,
      {
        "s": { "type": "bnode" , "value": "b2" } ,
        "p": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#first" } ,
        "o": { "type": "bnode" , "value": "b3" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://vivoweb.org/ontology/core#FacultyMember" } ,
        "p": { "type": "uri" , "value": "http://vitro.mannlib.cornell.edu/ns/vitro/
0.7#hiddenFromDisplayBelowRoleLevelAnnot" } ,
        "o": { "type": "uri" , "value": "http://vitro.mannlib.cornell.edu/ns/vitro/role#public" }
      }
    ]
  }
}
```

DESCRIBE to N3 example

This example reads all of the properties for a particular individual in the model, returning the result as N3.

```
curl -i -d 'email=vivo_root@mydomain.edu' -d 'password=Password' -d 'query=DESCRIBE <http://dbpedia.org/
resource/Connecticut>' -H 'Accept: text/n3' 'http://localhost:8080/vivo/api/sparqlQuery'
```

The response looks like this:

```
@prefix vitro:   <http://vitro.mannlib.cornell.edu/ns/vitro/0.7#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://dbpedia.org/resource/Connecticut>
```

```
        a       <http://vivoweb.org/ontology/core#StateOrProvince> ,
                <http://purl.obolibrary.org/obo/BFO_0000006> ,
                <http://vivoweb.org/ontology/core#Location> ,
                owl:Thing ,
                <http://vivoweb.org/ontology/core#GeopoliticalEntity> ,
                <http://purl.obolibrary.org/obo/BFO_0000002> ,
                <http://vivoweb.org/ontology/core#GeographicRegion> ,
                <http://purl.obolibrary.org/obo/BFO_0000001> ,
                <http://purl.obolibrary.org/obo/BFO_0000141> ,
                <http://vivoweb.org/ontology/core#GeographicLocation> ,
                <http://purl.obolibrary.org/obo/BFO_0000004> ;
        <http://www.w3.org/2000/01/rdf-schema#label>
                "Connecticut"@en ;
        <http://purl.obolibrary.org/obo/BFO_0000050>
                <http://aims.fao.org/aos/geopolitical.owl#United_States_of_America> ;
        vitro:mostSpecificType
                <http://vivoweb.org/ontology/core#StateOrProvince> .
```

10.9.3.5 Enabling the SPARQL Query API

Before enabling the SPARQL Query API, you should secure the URL `api/sparqlQuery` with HTTPS. Otherwise, email/password combinations will be sent across the network without encryption. Methods for securing the URL will depend on your site's configuration.

By default, the SPARQL Query API is disabled in VIVO for all users except the root user. To enable it for non-root users, you must edit the RDF file `[vivo]/home/rdf/auth/everytime/permission_config.n3` to authorize your site administrators to use the API. Find the permissions for `auth:ADMIN` and include the following permission:

permission_config.n3

```
auth:hasPermission simplePermission:UseSparqlQueryApi;
```

After editing this file you need to restart tomcat.

10.9.4 SPARQL Update API

10.9.4.1 Purpose

Permits external applications to add or remove specific triples from the VIVO data model. These changes use the standard data channels in VIVO, so the search index will be updated as appropriate, and the reasoner will add or remove inferences as needed.

⚠ By default, the SPARQL Update API is disabled in VIVO, for security reasons. See Enabling the API (see page 416).

10.9.4.2 Use Cases

Harvester

Previous implementations of the Harvester and similar tools have written directly to the VIVO triple-store, bypassing the usual data channels in VIVO. After ingesting, it was necessary to rebuild the search index, and to run the reasoner to add or remove inferences. Since the search index and the reasoner were not aware of the exact changes, the entire data model was re-indexed and re-inferenced.

When the Harvester and other tools have been modified to use the SPARQL Update API, VIVO will ensure that the search index and inferences are kept in synchronization with the data.

Other ingest tools

This API permits ingest tools such as Karma to programmatically insert data into VIVO without requiring knowledge of VIVOs internal data structures.

VIVO "face" applications

Linked Open Data requests have permitted people to write Drupal applications (for example) that display data from VIVO. This API will permit such applications to accept user edits, and apply them back to VIVO.

10.9.4.3 Specification

URL

`[vivo]/api/sparqlUpdate`

Examples:

```
http://vivo.cornell.edu/api/sparqlUpdate
```

```
http://localhost:8080/vivo/api/sparqlUpdate
```

HTTP Method

The API supports only HTTP POST calls. GET, HEAD, and other methods are not supported, and will return a response code of `405 Method Not Allowed`.

Parameters

name	**value**
`email`	the email address of a VIVO adminstrator account
`password`	the password of the VIVO administrator account
`update`	A SPARQL Update request

The syntax for a SPARQL Update request is described on the World Wide Web Consortium site at http://www.w3.org/TR/2013/REC-sparql11-update-20130321/

Limitation

The API requires that you specify a GRAPH in your SPARQL update request. Insertions or deletions to the default graph are not supported.

Response Codes

Code	**Reason**
`200 OK`	SPARQL Update was successful.
`400 Bad Request`	HTTP request did not include an `update` parameter.
	The SPARQL Update request did not specify a GRAPH.
	The SPARQL Update request was syntactically incorrect.
`403 Forbidden`	HTTP request did not include an `email` parameter.
	HTTP request did not include a `password` parameter.
	The combination of `email` and `password` is not valid.
	The selected VIVO account is not authorized to use the SPARQL Update API.

Code	Reason
`405 Method Not Allowed`	Incorrect HTTP method; only POST is accepted.
`500 Internal Server Error`	VIVO could not execute the request; internal code threw an exception.

10.9.4.4 Examples

These examples use the UNIX `curl` command to insert and delete data using the API.

Insert example

This example inserts a single RDF statement into the data model.

```
curl -i -d 'email=testAdmin@mydomain.edu' -d 'password=Password' -d '@insert.sparql' 'http://localhost:
8080/vivo/api/sparqlUpdate'
```

insert.sparql

```
update=INSERT DATA {
   GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> {
      <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Fundamentals of Compiler Design" .
    }
}
```

Modify example

This example removes the previous statement, and inserts a replacement.

```
curl -i -d 'email=testAdmin@mydomain.edu' -d 'password=Password' -d '@modify.sparql' 'http://localhost:
8080/vivo/api/sparqlUpdate'
```

modify.sparql

```
update=DELETE DATA {
   GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> {
      <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Fundamentals of Compiler Design" .
    }
}
INSERT DATA {
   GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> {
      <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
```

```
            "Design Patterns" .
     }
}
```

Delete example

This example removes the modified statement.

```
 curl -i -d 'email=testAdmin@mydomain.edu' -d 'password=Password' -d '@delete.sparql' 'http://localhost:
8080/vivo/api/sparqlUpdate'
```

delete.sparql

```
update=DELETE DATA {
   GRAPH <http://vitro.mannlib.cornell.edu/default/vitro-kb-2> {
      <http://test.domain/ns#book1>
          <http://purl.org/dc/elements/1.1/title>
          "Design Patterns" .
    }
}
```

Large Files

For large files one can also use the SPARQL LOAD[191] command.

For this, you have to first create the RDF file with the triples that you want to add, and make the file accessible at a URL. In the example below, the RDF file containing the triples is called `data.rdf`, and is available in the root directory of the web server at `myserver.address.xxx`.

Like the previous commands, this one references a data file, in this case called `import.sparql`. That file contains the `LOAD` command which references the actual data.

```
curl -d 'email=USER' -d 'password=PASSWORD' -d '@import.sparql' 'http://localhost:8080/vivo/api/
sparqlUpdate'
```

import.sparql

```
update=LOAD <http://myserver.address.xxx/data.rdf> into graph <http://vitro.mannlib.cornell.edu/default/
vitro-kb-2>
```

Increase the default Tomcat `maxPostSize`

By default, Tomcat sets the default maximum of a POST request to 2 megabytes. If you want to increase this to be able to POST larger sets of triples to VIVO, you can use the `maxPostSize` attribute in `server.xml`. The example below would increase the maximum to 10 MB. See the Tomcat documentation[192] for more details.

191 http://www.w3.org/TR/sparql11-update/#load
192 https://tomcat.apache.org/tomcat-7.0-doc/config/http.html

server.xml

```
<Connector port="8080" protocol="HTTP/1.1"
    connectionTimeout="20000"
   URIEncoding="UTF-8"
   redirectPort="8443"
   maxPostSize="10485760"/>
```

Enabling the API

Before enabling the SPARQL update handler, you should secure the URL `api/sparqlUpdate` with HTTPS. Otherwise, email/password combinations will be sent across the network without encryption. Methods for securing the URL will depend on your site's configuration.

By default, the SPARQL Update handler is enabled for only the root user in VIVO. To enable it for other user groups, you can either:

- uncomment the line references "UseSparqlUpdateAPI" in [vitro]/rdf/auth/everytime/permission_config.n3 or
- create an RDF file in the `[vitro]/rdf/auth/everytime` directory that will authorize your site administrators to use the API. Below is an example of such a file, using N3 syntax.

authorizeSparqlUpdate.n3

```
@prefix auth: <http://vitro.mannlib.cornell.edu/ns/vitro/authorization#> .
@prefix simplePermission: <java:edu.cornell.mannlib.vitro.webapp.auth.permissions.SimplePermission#> .

# Authorize the ADMIN role to use the SPARQL Update API
auth:ADMIN auth:hasPermission simplePermission:UseSparqlUpdateApi .
```

10.9.5 Search indexing service

- Securing the API (see page 419)

10.9.5.1 Purpose

Permits external applications to request specific updates to the VIVO search index, by providing a list of URIs whose search records may be out of date.

When the VIVO triple-store is updated in a way that bypasses VIVO's internal data channels, the search index will not reflect the updates.

With this service, you can provide a list of URIs whose contents have changed, and request that only those search records be updated. This is usually faster than rebuilding the entire index.

10.9.5.2 Use Cases

Use with ingest tools

The Harvester and similar tools write directly to the VIVO triple-store, bypassing the usual data channels in VIVO. After ingesting, it has been necessary to rebuild the search index so it will reflect the changes in the data. With this service, you can rebuild only part of the index.

Note: when the Harvester and other tools have been modified to use the SPARQL Update API, VIVO will ensure that the search index and inferences are kept in synchronization with the data.

Loading the triple-store

Some sites use two VIVO instances: a staging instance and a production instance. All ingests occur on the staging instance. Periodically, the triple-store is copied from staging to production. When this is done, you have 3 options:

- Copy the search index files from staging to production to keep it consistent with the triple-store
- Rebuild the search index in production
- Use the Search Indexing service to update specific records in the search index.

10.9.5.3 Indexing and Reasoning

The concerns that apply to the search index will also apply to the state of the inferred triples in the data model. When bypassing the data channels in VIVO, you bypass the semantic reasoner. To compensate for this, you must either

- Request that the reasoner rebuild all of the inferences, using `Recompute Inferences` from the `Site Administration` page, or
- Ensure that the ingested RDF contains all of the triples that you want VIVO to contain, including those that would be provided by the reasoner

In most cases, the time required to re-inference the model is greater than the time required to rebuild the search index. Unfortunately, the reasoning process is not easy to partition. To date, VIVO has no service that would allow you to update the inferences on a limited set of data.

10.9.5.4 Specification

URL

`[vivo]/searchService/updateUrisInSearch`

Examples:

```
http://vivo.cornell.edu/searchService/updateUrisInSearch
```

```
http://localhost:8080/vivo/searchService/updateUrisInSearch
```

HTTP Method

The API supports only HTTP POST requests with a content type of `multipart/form-data`.

If the request does not specify an encoding, UTF-8 is assumed.

Parameters

name	value
`email`	the email address of a VIVO administrator account
`password`	the password of the VIVO administrator account
`other`	One or more content parts, containing URIs to be indexed, separated by white space and/or commas

The name of the file content is unimportant. The API will examine all parts of the request and add any URIs to the list to be indexed. It is common, however, to put the entire list of URIs into a single content part.

Response Codes

Code	Reason
`200 OK`	Search indexing request was successful.
`403 Forbidden`	HTTP request did not include an `email` parameter.
	HTTP request did not include a `password` parameter.
	The combination of `email` and `password` is not valid.
	The selected VIVO account is not authorized to use the SPARQL Update API.
`500 Internal Server Error`	VIVO could not execute the request; internal code threw an exception.

10.9.5.5 Examples

This example uses the UNIX `curl` command to request updates to the search records of 3 individuals.

```
curl -v --form 'email=testAdmin@mydomain.edu' --form 'password=Password' --form 'uris=@uriList.txt'
'http://localhost:8080/vivo/searchService/updateUrisInSearch'
```

uriList.txt

```
http://vivo.mydomain.edu/individual/n6724
http://vivo.mydomain.edu/individual/n90987
http://vivo.mydomain.edu/individual/n32
```

10.9.5.6 Securing the API

⚠ ⚠ The Search Indexing service is enabled by default. However, it is recommended that you secure the URL `/searchService` with HTTPS. Otherwise, email/password combinations will be sent across the network without encryption. Methods for securing the URL will depend on your site's configuration.

10.10 Resource Links

The resources below should be helpful for anyone seeking additional information on topics related to VIVO, Vitro, ontologies, and the Semantic Web.

VIVO website	http://vivoweb.org/
VIVO project Wiki	https://wiki.duraspace.org/display/VIVO
VIVO project Facebook page	http://www.facebook.com/VIVOcollaboration
VIVO project Twitter	http://twitter.com/vivocollab
VIVO project LinkedIn group	https://www.linkedin.com/groups/2905369

Semantic Web technologies and Standards published by W3C	http://www.w3.org/2001/sw/wiki/Main_Page
Resource Description Framework is a standard model for data interchange on the web, to learn more about RDF	http://www.w3.org/2001/sw/wiki/RDF
More information on Web Ontology Language (OWL)	http://www.w3.org/2001/sw/wiki/OWL
SPARQL Query Language for RDF	http://www.w3.org/2001/sw/wiki/SPARQL
References to events, news, personal pages on the community	Semanticweb.org[193]
Semantic Web related conferences	the Semantic web "dogfood"[194]
Supporting the OWLED Workshop series and task forces	http://webont.org/owled/
Semantic Web portal dedicated to ontology design patterns (ODPs)	Ontology Design Pattern Wiki[195]
Books available on the Semantic web development	http://www.w3.org/2001/sw/wiki/Books#Books_on_Semantic_Web:_Intro
Protégé is a lightweight web-based ontology editor supporting OWL	http://protege.stanford.edu

193 http://semanticweb.org/
194 http://data.semanticweb.org/
195 http://ontologydesignpatterns.org/wiki/

11 About This Documentation

VIVO documentation is created by the users of VIVO.

If you find something here that is incorrect or confusing or incomplete, please let us know by posting on vivo-tech@googlegroups.com[196]

If you would like to write, rewrite or otherwise improve this documentation, please contact Graham Triggs[197]

11.1 Maintaining release-specific info on the Wiki

11.1.1 Goals

Recognize that some documentation applies only to a particular release, or set of releases.

Recognize that documentation will most likely be found by web searches, not by walking the wiki.

Information about the current release should be the easiest to find.

Information about older releases should be available somewhere.

There should be an area for documenting new features before they are included in a release.

It should be easy to tell whether the information you are viewing is correct for the code you are using.

Documentation should not be frozen at release time - It should remain available for improvements.

The most basic instructions should be included in the release.

Novice users should be able to find what they need; expert users should be able to find more

11.1.2 Two types of wiki pages

11.1.2.1 Release-specific pages

- Apply to a specific release, or range of releases.
- Incorrect if used with an inappropriate release.
- Examples: installation instructions, customization guides, ontology details

196 mailto:vivo-tech@googlegroups.com
197 https://wiki.duraspace.org/display/~grahamtriggs

11.1.2.2 Release-neutral pages

- Equally relevant to all releases (or to none)
- Examples: tutorials, meeting agendas, glossary, outreach events and resources, presentations.
- Most wiki pages are not release specific

11.1.3 Approach

11.1.3.1 VIVO (main wiki, also known as the project wiki, also known as the community wiki)

- Located at http://wiki.duraspace.org/display/VIVO
- The main wiki holds all sorts of information, as it does now.
- Includes governance, task forces, interest groups, background material, community support materials
- Does not include release specific information describing the product

11.1.3.2 VIVO Release specific wikis, also known as the documentation

- The release-specific information will be collected in release specific wikis. These wikis will be copied forward to create spaces for new releases. The next release wiki will be available before the release of the software to document the next release.
- Contains release-specific pages for older releases.
- The styling indicates that the wiki id documentation for a specific release
- Release specific wikis exist for releases prior to the current release, for the current release and for the next release
- The release specific wikis use a documentation template and a documentation PDF export template optimized for the production of a single PDF document from the documentation wiki.

11.1.3.3 Minimal documentation in the Git repository

- The release specific documentation wiki is the definitive documentation for the current release
- The project wiki is the definitive source of material regarding the project
- The Git README.md refers to the project wiki and the release specific wiki and describes each

11.1.4 Between releases

- All community processes continue in the project wiki
- Release specific wikis are available for improvement
- A next release wiki is created from the current release wiki when there are new features to document.

11.2 VIVO documentation style guide

Each VIVO document consists of a large number of wiki pages. These pages must work well together, both on the wiki and when exported to a PDF file. Here are some suggestions to help that happen.

11.2.1 Page sizes

Keep each page to a manageable size, and focused on a particular topic. If you find that the page is too complex or too diverse, break it into smaller pages. Within each group of pages, the parent should contain introductory material or an overview, while the child pages explore individual topics.

11.2.2 Start with a Table of Contents

Start the page with a call to the "Table of Contents" macro. The Table of Contents will include all of the headings in the current page. It will also include a top-level heading for each child page, thanks to the "Children Display" macro.

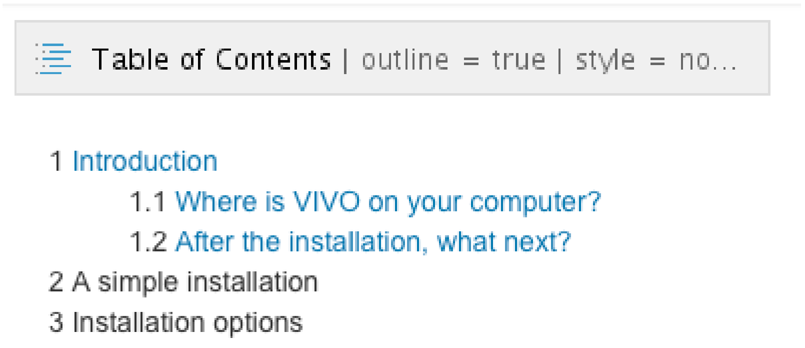


Table of Contents | outline = true | style = no...

1 Introduction
    1.1 Where is VIVO on your computer?
    1.2 After the installation, what next?
2 A simple installation
3 Installation options

When the document is exported to a PDF file, the "Table of Contents" macros are not included. Instead, the file begins with a table of contents for the full document.

11.2.3 Use all heading levels

The major headings on all pages should be Heading 1. The second headings on all pages should be Heading 2. Use the Heading styles only – do not format headings using bold, italics, etc. When the pages are combined into a PDF file, the heading levels will be displayed properly and numbered correctly in the table of contents and in the document.

Pagination in the PDF document is controlled by the PDF template, not by the author. Use page titles and page headings consistently. The resulting PDF document will then be consistent.

Headings within the child pages are demoted accordingly, to keep the organization intact. So a level 2 heading in the Table of Contents might represent:

- a level 2 heading in the parent page,
- a level 1 heading in a child page,
- the title of a grandchild page.

11.2.4 Code

Use a code block macro to represent code. A title for the code block is optional.

Use monospace in sentences to represent code.

11.2.5 Linking within the document

Links in the technical documentation wiki need to be checked to make sure they are referring to other parts of the tech doc, or external sources. Links to pages in the project wiki are “okay” but need to be done carefully -- such links are often red flags that the tech doc is drifting into content better suited for the project wiki. Links to the archive should be avoided. Content from the archive that is relevant for the current version should be copied into the documentation wiki.

11.2.6 End with a Children Display macro

The documentation wiki for VIVO includes a child display at the end of every page. There is no need to include a children display macro explicitly. It will be added for you.