VIVO Archive

VIVO Installation Instructions

Author: Jim Blake

Version: 29

Date: Aug 20, 2019 6:35 AM

Table of Contents

This document applies to VIVO release 1.6. For documents that apply to other releases, please consult the Documentation Archive page.

1 Introduction

This document tells you how to install VIVO on your computer.

- The introduction explains a few concepts about VIVO, how it is structured, and how it is built.
- The section entitled A simple installation describes a standard, simple installation for those who just want to get VIVO up and running.
- The section called Installation options describes several choices; some for production installations of VIVO, and some for developers who are working with VIVO on their desktops.

For information about this release, consult the VIVO 1.6 Release Announcement. Links to this and other documentation can be found on the support page at VIVOweb.org.

These instructions assume that you are performing a clean install. If you are upgrading an existing service, please consult the Upgrade Instructions for VIVO release 1.6. VIVO may not work as expected if you install over an an earlier version.

1.1 How is VIVO distributed?

1.1.1 VIVO Source code

The latest release of VIVO can be found at the download page of VIVOweb.org. VIVO is distributed as source code, which you will use to build the application. This is because almost all site want to add their own customizations to VIVO.

These instructions will lead you through the process of building and installing VIVO.

1.1.2 Virtual appliance

VIVO is also available as a "virtual appliance", which you not need to build. We rely on the VIVO community to create new versions of the virtual appliance, so you may not find one that contains the latest release of VIVO. The latest virtual appliance can be found at the download page of VIVOweb.org.

1.1.3 VIVO and Vitro

VIVO is a research networking application that is built around a general-purpose RDF editor and browser named Vitro. VIVO packages Vitro with a display theme, an ontology, and many customizations. You will see references to Vitro occasionally in the installation instructions. For example, setting a property named `vitro.home` where you might expect to see `vivo.home` instead.

Remember that VIVO is a customization of Vitro.

1.2 Where will VIVO be on your computer?

Before beginning the installation, you should be aware of the four locations on your computer that will hold VIVO.

1.2.1 The VIVO distribution directory

This is created when you unpack the VIVO distribution file (see Download the VIVO source code, below). This is where you will create your build.properties file (see Specify build properties), and where you will make any modifications to the VIVO theme or code. You can create this wherever you choose.

1.2.2 VIVO inside Tomcat

When you run the build script to compile and deploy VIVO (see Compile and deploy), the files will be deployed to a directory inside Tomcat. This is the actual executing code for VIVO, but you won't need to look at it or change it. If you need to change VIVO, make the changes in the distribution directory, and run the build script again. Tell the build script where to find Tomcat by setting `tomcat.home` in the build.properties file (see Specify build properties).

1.2.3 The VIVO home directory

This directory contains the runtime properties for VIVO. VIVO will also use this area to store some of its data. Uploaded image files are stored here, and the Solr search index is stored here also. This is also a place for the files of RDF data that will initialize the VIVO knowledge base. You can create this wherever you choose. Tell VIVO where to find the home directory by setting `vitro.home` in the build.properties file (see Specify build properties). You must create this directory before starting VIVO. You must create the `runtime.properties` file in this directory (see Specify runtime properties), and you must ensure that Tomcat has permission to read and write to this directory when it runs.

1.2.4 The VIVO knowledge base

Nearly all of the data that you enter into VIVO will be stored in a MySQL database. The actual location of this data depends on what system you have, and on how you install MySQL (see Install required software). You will access the data through VIVO, or occasionally through the MySQL client application.

> ⚠ Depending on your Installation options, these four locations may be in different places, or may be specified in different ways. They may even exist on different computers. Regardless of the options, these four locations are important for any installation of VIVO.

1.3 After the installation, what next?

When you have VIVO up and running, please read the Site Administrator's Guide.

2 A simple installation

How to get VIVO up and running on your computer, for testing or experimentation, or just to learn how to do it.

If you want to install VIVO on a production server, or if you want to develop VIVO code, you should also read the section on Installation options.

2.1 Preparing for VIVO

2.1.1 Install required software

Before installing VIVO, make sure that the following software is installed on your computer:

- Java (SE) 1.7.x http://java.sun.com
  - 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
- MySQL 5.1 or higher, http://www.mysql.com

Set up the environment variables for `JAVA_HOME` and `ANT_HOME` and add the executables to your path, as required. This requirement depends on the operating system you are using. Consult the installation directions from the software support websites.

The following browsers are supported for this release

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

Did it work?

You can test the software installation by typing these commands:

```
java -version
mysql --version
ant -version
```

Each of these command should print a response that tells you what version is installed. If any of these commands prints an error message, or reports an unexpected version number, you should review your installation.

2.1.2 Create an empty database and a database account

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

Log into your MySQL server and create a new database in MySQL that uses `UTF-8` encoding. At the MySQL command line you can create the database and user with these commands substituting your values for `dbname`, `username`, and `password`. Most of the time, the hostname will be `localhost`.

```
CREATE DATABASE dbname CHARACTER SET utf8;
GRANT ALL ON dbname.* TO 'username'@'hostname' IDENTIFIED BY 'pass
word';
```

2.2 Building VIVO

2.2.1 Download the VIVO source code

Download the VIVO application source as either `rel-1.6.zip` or `rel-1.6.gz` file and unpack it on your web server: http://vivoweb.org/download

2.2.2 Specify build properties

At the top level of the VIVO distribution directory, rename the file `example.build.properties` to `build.properties`. Edit the file to suit your installation, as described in the following section.

These properties are used in compiling VIVO and deploying it to Tomcat. They will be incorporated into VIVO when it is compiled. If you want to change these properties at a later date, you will need to stop Tomcat, repeat the Compile and deploy step, and restart Tomcat.

Windows: For those installing on a Windows operating system, include the windows drive, but use the forward slash "/" and not the back slash "\" in the directory locations, e.g. `c:/tomcat`.

Property name	**`vitro.core.dir`**
Description	The directory where Vitro code is located. In the simple installation, this is set to `./vitro-core`
Default value	NONE
Example value	`./vitro-core`

Property name	**`tomcat.home`**
Description	The directory where tomcat is installed.
Default value	NONE
Example value	`/usr/local/tomcat`

Property name	**`webapp.name`**
Description	The name of your VIVO application. This is not a name that will be displayed to the user. This name appears in the URL used to access VIVO, and in the path to the VIVO directory within Tomcat.
Default value	NONE

Property name	**webapp.name**
Example value	vivo

Property name	**vitro.home**
Description	The directory where VIVO will store the data that it creates. This includes uploaded files (usually images) and the Solr search index. Be sure this directory exists and is writable by the Tomcat service.
Default value	NONE
Example value	/usr/local/vivo/home

2.2.3 Compile and deploy

In the previous step, you defined the location of the VIVO home directory, by specifying `vitro.home` in the `build.properties` file. If that directory does not exist, create it now.

At the command line, from the top level of the VIVO distribution directory, type:

```
ant all
```

to build VIVO and deploy to Tomcat's webapps directory.

The build script may run for as much as five minutes, and creates more than 100 lines of output. The process includes several steps:

- collecting the source files from the distribution directory,
- compiling the Java source code,
- running unit tests,
- preparing the Solr search engine,
- deploying VIVO and Solr to Tomcat.

Did it work?

If the output ends with a success message, the build was successful. Proceed to the next step.

```
BUILD SUCCESSFUL
Total time: 1 minute 49 seconds
```

If the output ends with a failure message, the build has failed. Find the cause of the failure, fix the problem, and run the script again.

```
BUILD FAILED
Total time: 35 seconds
```

The output of the build may include warning messages. The Java compiler may warn of code that is outdated. Unit tests may produce warning messages, and some tests may be ignored if they do not produce consistent results. If the output ends with a success message, these warnings may be ignored.

What user account owns the VIVO directories?

In many operating systems, the issue of file permissions is important. Who owns the files? Who is authorized to read them, or to write new files?

When running the VIVO build script, it must have permission to read and write to:

- the VIVO distribution directory
- the Tomcat webapps directory
- the VIVO home directory

When VIVO is started under Tomcat, Tomcat must have permission to read and write to:

- the Tomcat webapps directory
- the VIVO home directory

There are several ways to make this work. People who are experimenting with VIVO often use their own account to create the VIVO distribution directory, to run the build script, and to run Tomcat.

In more formal environments, it may be necessary to run Tomcat as a service, under its own account. In that case, some people choose to run the build script with `root` privilege, and then assign the resulting files to Tomcat:

```
sudo ant all
sudo chown -R tomcat /usr/local/vivo/home
sudo chown -R tomcat /usr/local/tomcat/webapps/vivo*
```

When installing on Microsoft Windows, this is not usually a problem.

2.3 Running VIVO

2.3.1 Configure Tomcat

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 that 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:

`setenv.sh`

```
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. Lower values may be sufficient, especially for small test installations.

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

Set security limits

VIVO is a multithreaded web application that may require more threads than are permitted under your Linux installation's default configuration. Ensure that your installation can support the required number of threads by making the following edits to `/etc/security/limits.conf`:

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

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.

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.

See the section entitled Running VIVO behind an Apache server for an example of overriding the VIVO context fragment.

2.3.2 Specify runtime properties

The build process in the Compile and deploy step created a file called `example.runtime.properties` in your VIVO home directory (specified by `vitro.home` in the `build.properties` file). Rename this file to `runtime.properties` and edit the file to suit your installation, as described below.

These properties are loaded when VIVO starts up. If you want to change these properties at a later date, you will need to restart Tomcat for them to take effect. You will not need to repeat the Compile and deploy step.

> ⚠ *Windows:* For those installing on Windows operating system, include the windows drive and use the forward slash "/" and not the back slash "\" in the directory locations, e.g. `c:/tomcat`.

Basic properties

These properties define some fundamental aspects of your VIVO installation. Most sites will need to modify all of these values.

Property name	**`Vitro.defaultNamespace`**
Description	The default RDF namespace for this installation. VIVO installations make their RDF resources available for harvest using linked data. Requests for RDF resource URIs redirect to HTML or RDF representations as specified by the client. To make this possible, VIVO's default namespace must have a certain structure and begin with the public web address of the VIVO installation. For example, if the web address of a VIVO installation is http://vivo.example.edu/ the default namespace must be set to "http://vivo.example.edu/individual/" in order to support linked data. Similarly, if VIVO is installed at http://www.example.edu/vivo the default namespace must be set to "http://www.example.edu/vivo/individual/" *** The namespace must end with "individual/" (including the trailing slash).**
Default value	NONE
	`http://vivo.mydomain.edu/individual/`

Property name	**`Vitro.defaultNamespace`**
Example value	

Property name	**`rootUser.emailAddress`**
Description	Specify the email address of the root user account for the VIVO application. This user will have an initial temporary password of `rootPassword`. You will be prompted to create a new password on first login. NOTE: The root user account has access to all data and all operations in VIVO. Data views may be surprising when logged in as the root user. It is best to create a Site Admin account to use for every day administrative tasks.
Default value	NONE
Example value	`vivoAdmin@my.domain.edu`

Property name	**`VitroConnection.DataSource.url`**
Description	Specify the JDBC URL of your database. Change the end of the URL to reflect your database name (if it is not "vitrodb").
Default value	NONE
Example value	`jdbc:mysql://localhost/vivo`

Property name	**VitroConnection.DataSource.username**
Description	Change the username to match the authorized user you created in MySQL.
Default value	NONE
Example value	`username`

Property name	**VitroConnection.DataSource.password**
Description	Change the password to match the password you created in MySQL.
Default value	NONE
Example value	`password`

Property name	**email.smtpHost**
Description	Specify an SMTP host that the application will use for sending e-mail (Optional). If this is left blank, the contact form will be hidden and disabled, and users will not be notified of changes to their accounts.
Default value	NONE
Example value	`smtp.servername.edu`

Property name	**email.replyTo**
Description	

Property name	**email.replyTo**
	Specify an email address which will appear as the sender in e-mail notifications to users (Optional). If a user replies to the notification, this address will receive the reply. If a user's e-mail address is invalid, this address will receive the error notice. If this is left blank, users will not be notified of changes to their accounts.
Default value	NONE
Example value	`vivoAdmin@my.domain.edu`

Connecting to the Solr search index

VIVO and its search index are actually two distinct web applications, and the simple installation puts them both into the same instance of Tomcat. Even so, the VIVO webapp must be told how to reach the Solr webapp.

Property name	**vitro.local.solr.url**
Description	URL of Solr context used in local VIVO search. Should consist of: `scheme + servername + 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 (specified above) + "solr"
Default value	NONE
Example value	`http://localhost:8080/vivosolr`

Additional properties

The `runtime.properties` file can accept many additional properties, but they aren't necessary for this simple installation. If you choose any of the Installation options, you will probably need to set some of those properties.

2.3.3 Start Tomcat

Most Tomcat installations can be started by running `startup.sh` or `startup.bat` in Tomcat's `bin` directory. Start Tomcat and direct your browser to `http://localhost:8080/vivo` to test the application. Note that Tomcat may require several minutes to start VIVO.

On start up VIVO will run some diagnostic tests. If a problem is detected the normal VIVO pages will redirect to a startup status page describing the problem. You can stop Tomcat, attempt to fix the problem and proceed from the Compile and deploy step. If the problem is not serious, the startup status page may offer a `continue` link which will allow you to use VIVO in spite of the problems.

If the startup was successful, you will see the VIVO home page.

If Tomcat does not start up, or the VIVO application is not visible, check the files in Tomcat's logs directory. Error messages are commonly found in `[tomcat]/logs/catalina.out`, `[tomcat]/logs/vivo.all.log` or `[tomcat]/logs/localhost.log`

> ⚠ Remember that Tomcat must have permission to read and write its own files, and the files in the VIVO home directory. This may mean that you must use a particular script or a particular user account to start Tomcat.

2.4 Start using VIVO

2.4.1 Log in and add RDF data

Direct your browser to the VIVO home page. Click the "Log in" link near the upper right corner. Log in with the `rootUser.emailAddress` that you set in the `runtime.properties` file. The initial password for the root account is `rootPassword`. When you first log in, VIVO will require you to change the password. When login is complete, the search index is checked and, if it is empty, a full index build will be triggered in the background, in order to ensure complete functionality throughout the site.

After logging in, you will be presented with a menu of editing options. Here you can create OWL classes, object properties, data properties, and configure the display of data. Currently, any classes you wish to make visible on your website must be part of a class group and any individual must have an rdfs:label. There are a number of visibility and display options available for classes and properites. VIVO comes with a core VIVO ontology, but you may also upload other ontologies from an RDF file.

Under the "Advanced Data Tools" click "Add/Remove RDF Data." Note that Vitro currently works best with OWL-DL ontologies and has only limited support for pure RDF data. You can enter a URL pointing to the RDF data you wish to load or upload from a file on your local machine. Ensure that the "add RDF" radio button is selected. You will also likely want to check "create classgroups automatically."

Clicking the "Index" tab in the navigation bar at the top right of the page will show a simple index of the knowledge base.

See more documentation for configuring VIVO, ingesting data, and manually adding data at http://vivoweb.org/support.

2.4.2 Set the Contact Email Address

If you have configured your application to use the "Contact Us" feature (`email.smtpHost` is set in the `runtime.properties` file), you will also need to add an email address to the VIVO application. This is the email to which the contact form will submit. It can be a list server or an individual's email address.

Log in as a system administrator. Navigate to the "Site Admin" table of contents (link in the right side of the header). Go to "Site Information" (under "Site Configuration"). In the "Site Information Editing Form," enter a functional email address in the field "Contact Email Address" and submit the change.

If you set the `email.smtpHost` in the `runtime.properties` file, and do NOT provide an email address in this step, your users will see an error message instead of the expected contact form.

2.4.3 Review the VIVO terms of use

VIVO comes with a "Terms of Use" statement linked from the footer. The "Site Name" you assign in the "Site Information" form under the **Site Admin** area will be inserted into the "Terms of Use" statement. If you want to edit the text content more than just the "Site Name", the file can be found here:

```
[vivo_source_dir]/vitro-core/webapp/web/templates/freemarker/body/termsOfUse.f
```

Your "Terms of Use" statement is also referenced in the Linked Open Data (RDF) that your site produces, so you should be sure that it accurately reflects the way that your data may be used.

Be sure to make the changes in your source files and deploy them to your tomcat so you don't lose your changes next time you deploy for another reason.

2.5 Was the installation successful?

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.
- When you Log in and add RDF data, you verify that you can log in to the root VIVO account.

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**.

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://marbles.sourceforge.net/. In the pink box on that page enter the URI of the individual you created in the previous step and click "open."
- In the resulting page search for the URI of the "test individual." You should find it towards the bottom of the page next to a red dot followed by "redirect (303)." This indicates that you are successfully serving linked RDF data. If the URI of the "test individual" is followed by "failed (400)" 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 the search index, you can log in as a site administrator and go to **Site Admin -> Rebuild search index**.

3 Installation options

These options can make VIVO more scalable, add features, or run VIVO in a different environment.

3.1 Linking user accounts to VIVO profile pages

Configure VIVO so each user can edit their own profile page.

At many VIVO sites, users are encouraged to log in an edit their own profile pages. But how to tell which page belongs to which user?

VIVO will try to associate the user with a profile page, so the user may edit his own profile data. When a user logs in, VIVO searches the data model for a person with a property that matches the user's network ID. You need to tell VIVO what property should be used for matching.

Add this to `runtime.properties`:

Property name	**`selfEditing.idMatchingProperty`**
Description	The URI of a property that can be used to associate an Individual with a user account. When a user logs in with a name that matches the value of this property, the user will be authorized to edit that Individual (the value of the property must be either a String literal or an untyped literal).
Default value	NONE
Example value	`http://vivo.mydomain.edu/ns#networkId`

This process is discussed in greater detail in the VIVO Customization Guide, in the section called How are User Accounts associated with Profile pages?

3.2 Running VIVO behind an Apache server

Most sites choose to configure their VIVO system with an Apache HTTPD web server to accept requests and then proxy them to the VIVO Tomcat context. This will make Vitro available at `http://example.com` instead of `http://example.com:8080/vitro`. It will also allow the use of external authentication.

Setup HTTPD to send all of the requests that it receives to Tomcat's AJP connector. This can be done in HTTPD 2.x with a simple directive in httpd.conf:

```
ProxyPass / ajp://localhost:8009/
```

Modify the <Host> in Tomcat server.xml (located in `[tomcat root]/conf/`) so that the context path is empty to allow VIVO to be servred from the root path. Locate the `<Host name="localhost"...>` directive and update as follows:

```
<Host name="localhost" appBase="webapps"
        DeployOnStartup="false"
        unpackWARs="true" autoDeploy="false"
        xmlValidation="false" xmlNamespaceAware="false">

    <Alias>example.com</Alias>

        <Context path=""
                docBase="/usr/local/tomcat/webapps/vitro"
                reloadable="true"
                cookies="true" >

            <Manager pathname="" />
        </Context>

        ...
```

After setting up the above, it is recommended that you modify the Tomcat AJP connector parameters in server.xml. Look for the <connector> directive and add the following properties:

```
    connectionTimeout="20000" maxThreads="320" keepAliveTimeout="2
0000"
```

Note: the value for maxThreads (320) is equal or greater than the value for MaxClients in the apache's `httpd.conf` file.

3.3 Using an external authentication system

VIVO can be configured to work with an external authentication system like Shibboleth or CUWebAuth.

In order to effectively use an external authentication system, VIVO must be accessible only through an Apache HTTP server. The Apache server will be configured to invoke the external authentication system. When the user completes the authentication, the Apache server will pass a network ID to VIVO, to identify the user.

If VIVO has an account for that user, the user will be logged in with the privileges of that account. In the absence of an account, VIVO will try to find a page associated with the user. If such a page is found, the user can log in to edit his own profile information.

3.3.1 Configuring the Apache server

Your institution will provide you with instructions for setting up the external authentication system. The Apache server must be configured to secure a page in VIVO. When a user reaches this secured page, the Apache server will invoke the external authentication system.

For VIVO, this secured page is named: `/loginExternalAuthReturn`

When your instructions call for the location of the secured page, this is the value you should use.

3.3.2 Configuring VIVO

To enable external authentication, VIVO requires two values in the `runtime.properties` file.

Property name	**`externalAuth.netIdHeaderName`**
Description	The name of the HTTP header that will hold the external user's network ID. When a user completes the authentication process, the Apache server will put the user's network ID into one of the headers of the HTTP request. The instructions from your institution should tell you which header is used for this purpose.

Property name	`externalAuth.netIdHeaderName`
Default value	NONE
Example value	`remote_userID`

Property name	`selfEditing.idMatchingProperty`
Description	### Associating a User with a profile page. VIVO will try to associate the user with a profile page, so the user may edit his own profile data. VIVO will search the data model for a person with a property that matches the User's network ID (the value of the property must be either a String literal or an untyped literal). You need to tell VIVO what property should be used for matching. This property is also mentioned in the insructions for A simple installation, because it can also be useful for sites that do not use external authentication.
Default value	NONE
Example value	`http://vivo.mydomain.edu/ns#networkId`

Finally, you will need to provide text for the Login button.

To start the authentication process, the user will click on a button in the VIVO login form. You need to tell VIVO what text should appear in that button. In your theme, add a line to the `all.properties` file, like this one:

```
external_login_text = [the text for your login button]
```

For example:

```
external_login_text = Log in using BearCat Shibboleth
```

The VIVO login form will display a button labelled "Log in using BearCat Shibboleth".

If your site supports additional languages, add lines to the corresponding files. For example, `all_es.properties` might contain this line:

```
external_login_text = Entrar usando Shibboleth GatoOso
```

3.3.3 More information

You can find more technical details about the Interface to external authentication systems.

Also, it may help to know How are User Accounts associated with Profile pages?

3.4 Tuning the database connection pool

The number of database connections can affect VIVO's speed.

Set these values in `runtime.properties`, to control the number of active connections to the database, and the number of idle connections.

Property name	**`VitroConnection.DataSource.pool.maxActive`**
Description	Specify the maximum number of active connections in the database connection pool to support the anticipated number of concurrent page requests.
Default value	`40`
Example value	`40`

Property name	`VitroConnection.DataSource.pool.maxIdle`
Description	Specify 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.
Default value	`10`
Example value	`10`

3.5 Using a different data store

VIVO can work with databases like Oracle or SQL Server, or with external triple stores.

3.5.1 Different database type

To use a database other than MySQL, set these values in runtime.properties. Depending on your database, you might also need to change the basic properties relating to VitroConnection.

Property name	`VitroConnection.DataSource.dbtype`
Description	Change the dbtype setting to use a database other than MySQL. Otherwise, leave this value unchanged. Possible values are DB2, derby, HSQLDB, H2, MySQL, Oracle, PostgreSQL, and SQLServer. Refer to http://openjena.org/wiki/SDB/Databases_Supported for additional information. ⚠ VIVO releases 1.5 and 1.6 use Jena version 1.3.4, which does not work correctly with Microsoft SQL Server. Unfortunately, it is not trivial to upgrade to a later version of Jena. Later versions of Jena do not support Jena RDB, which VIVO uses for contextual data. This will be repaired in VIVO release 1.7.

Property name	**`VitroConnection.DataSource.dbtype`**
Default value	`MySQL`
Example value	`Oracle`

Property name	**`VitroConnection.DataSource.driver`**
Description	Specify a driver class name to use a database other than MySQL. Otherwise, leave this value unchanged. This JAR file for this driver must be added to the the webapp/lib directory within the vitro.core.dir specified above.
Default value	`com.mysql.jdbc.Driver`
Example value	`com.mysql.jdbc.Driver`

Property name	**`VitroConnection.DataSource.validationQuery`**
Description	Change the validation query used to test database connections only if necessary to use a database other than MySQL. Otherwise, leave this value unchanged.
Default value	SELECT 1
Example value	SELECT 1

3.5.2 Optional external triple store

VIVO can configured to use a different triple store for the bulk of its semantic data, so long as this triple store supports Web-based use of the SPARQL language to query and modify its data. If you elect to use a separate triple store, note that VIVO's MySQL database is still required for basic configuration and user account data. In order to connect VIVO to an external triple store, you will need to know two URIs: the store's endpoint URI for issuing SPARQL queries that read data, and its URI for issuing SPARQL UPDATE commands. These URIs are typically kept separate in order to make it easier to secure the triple store against unauthorized edits. With Sesame, for example, the update URI is usually the query endpoint URI with "/statements" appended.

You will need to know these two URIs later when you specify runtime properties.

Property name	**`VitroConnection.DataSource.endpointURI`**
Description	Set the endpointURI only if you wish to store semantic data in an external triple store instead of MySQL. Enter the URI of the triple store's SPARQL endpoint for querying data.
Default value	NONE
Example value	

Property name	**`VitroConnection.DataSource.updateEndpointURI`**
Description	Set the updateEndpointURI only if you wish to store semantic data in an external triple store instead of MySQL. Enter the URI at which the triple store responds to SPARQL UPDATE requests. This setting is only necessary if the triple store does not support updates via its main URI. If the endpointURI above is not set, this setting has no effect.
Default value	NONE
	.

Property name	**`VitroConnection.DataSource.updateEndpointURI`**
Example value	

3.6 Developers' installation: obtaining VIVO source code from Git

Some people prefer to work with the very latest source code.

Some people prefer to get the VIVO code directly from the GitHub repositories, instead of downloading the distributed release files. The `master` branch of the repositories always contains the latest release code, and is available for anyone to access.

If you prefer not to wait for the next stable release, you can get the VIVO code as it is being developed. The `develop` branch has not been fully tested and is not recommended for production use, but it can be useful for people who want a "close-up" look at how VIVO is progressing.

3.6.1 VIVO code and Vitro code

The principal difference between the release files and the repositories is the location of the Vitro code, in relation to the VIVO code.

Working from the release files

In the VIVO release files, the Vitro code is embedded within VIVO.

When you create the `build.properties` file, you set `vitro.core.dir` accordingly.

Working from the GitHub repositories

When you use the code from the repositories, it is customary to place the Vitro code beside the VIVO code.

When you create the `build.properties` file, `vitro.core.dir` must be set to reflect this difference.

Working from the release files	Working from the GitHub repositories
/Users/jeb228/vivo-rel-1.6 build.xml config doc example.build.properties languages lib productMods rdf README.md revisionInfo src test themes utilities vitro-core doc opensocial README.md revisionInfo solr utilities webapp	/Users/jeb228/git Vitro doc opensocial README.md solr utilities webapp VIVO build.xml config doc example.build. properties languages lib productMods rdf README.md src test themes utilities
build.properties	**build.properties**
`vitro.core.dir = ./vitro-core`	`vitro.core.dir = ../Vitro`

3.6.2 Revision Information

Working from the release files

In the VIVO release files, the `VIVO` directory contains a file called `revisionInfo`. This file states the release version of VIVO, and a seven-character hashcode that can be used to locate the release tag in the GitHub repository. The `VIVO/vitro-core` directory contains a similar file, with similar information about the Vitro release.

Working from the GitHub repositories

The `revisionInfo` files are not stored in the GitHub repositories. Instead, when you run the build script, queries are made to Git to find the most recent tag, and the hashcode of the most recent commit. In this way, VIVO can report the level of code that was used to build it.

3.6.3 A Frequently Asked Question

Q: I'm running VIVO in Tomcat on a MacIntosh. When I upload an image file, a strange icon appears in the dock. What's up with that?

A: The image processing code in VIVO uses javax.imageio.ImageIO, which in turn uses code in the java.awt package. By default, this package will open a work area on your screen, producing this icon even though no visible window is created.

You can prevent this by editing {tomcat}/conf/setenv.sh, and adding the "java.awt.headless" option. For example,

```
export CATALINA_OPTS="-Xms1024m -Xmx1024m -XX:MaxPermSize=64m -
Djava.awt.headless=true"
```

3.7 Building VIVO in 3 tiers

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 files that differ from a regular VIVO build (other than the ones you are adding and changing) are the build.xml and deploy.properties file. I've attached the files here

- build.xml
- deploy.properties

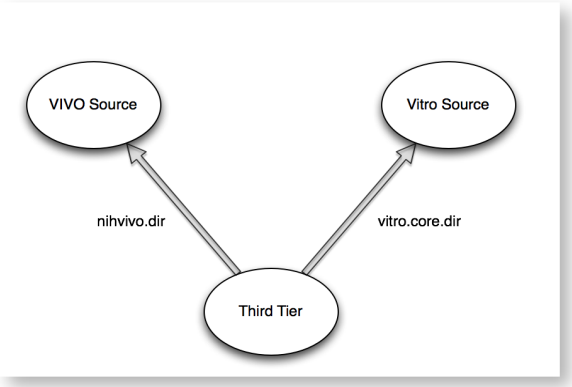
The main difference in these files is the presence of nihvivo.dir. This attribute points to where VIVO is for the build. There are two standard ways to setup VIVO, for development or deployment.

3.7.1 Development

With development you want to look at your VIVO agains the VIVO and Vitro sources. this means getting form the git repository the source for each project

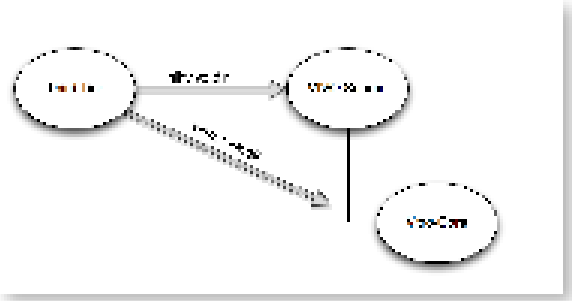
- https://github.com/vivo-project/Vitro
- https://github.com/vivo-project/VIVO

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.



3.7.2 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.



3.7.3 Project template

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

3.8 Building a VIVO distribution for other servlet containers

Some sites prefer to use a container like GlassFish or JBoss, instead of Tomcat

The simple installation process for VIVO tells how to build and deploy VIVO into a Tomcat servlet container. This document tells how to build VIVO so it can be used in any servlet container that supports the Java Servlet 2.4 Specification.

This process will produce:

- a WAR file for the VIVO application,
- a WAR file for the Solr application,
- a TAR file for the basic Vitro home directory, ready for configuring,
- a TAR file for the Solr home directory, configured to work with VIVO.

These artifacts can then be installed into a servlet container (or more than one), and configured to work together.

The configuration includes the customary `runtime.properties` file in the Vitro home directory. It also requires items that tell the VIVO application and the Solr application how to find their respective home directories. These items are specific to the servlet container. However, they are described so you can translate them to your container of choice.

Start by reading the customary instructions for installing VIVO, taking note of the Tomcat-specific sections. This document will consist mostly of comparisons to those instructions.

3.8.1 Before the build

Required software

Tomcat is not required. You can use any servlet container that supports the Java Servlet 2.4 Specification.

Logging properties for VIVO

The logging properties for VIVO are determined by the file `[vitro-core]/webapp/config/log4j.properties`. *(Note: if `debug.log4j.properties` exists, it will override `log4j.properties`).*

Notice how the location of the log file is determined:

```
log4j.appender.AllAppender.File=$${catalina.home}/logs/${webapp.name}.all.log
```

The filename of the log file is based on the `webapp.name` property found in the `build.properties` file. This substitution is made during the build process. The path to the log file is based on the system property `catalina.home` which is set by Tomcat at runtime.

You will likely want to change this line -- making it an absolute path, or basing it on some other system property. Notice that `ant` will substitute properties such as `${webapp.name}` during the build process. The name of a system property requires a second dollar sign, e.g. `$${catalina.home}` to protect it from `ant`. `Ant` will remove the second dollar sign, but will not try to substitute a value for the property. At runtime, when Log4J reads the properties file, it will substitute the matching system property.

Logging properties for Solr

The logging properties for Solr are determined by the file `[vitro-core]/webapp/config/solr/logging.properties`.

As with VIVO, the location of the log file is based on the system property `catalina.home` which is set by Tomcat at runtime.

```
org.apache.juli.FileHandler.directory = ${catalina.base}/logs
```

Note that only one dollar sign is used, since Ant does not attempt to substitute properties in this file.

The syntax of the Solr logging properties is different from the syntax for VIVO. This is because VIVO uses Log4J as a back end for the Commons Logging framework, while Solr uses JULI as a back end for SLF4J.

Note: Solr is able to do this because Tomcat provides the JULI framework by default. Other servlet containers may require JULI to be installed. If you encounter this as an issue, please share your experience with the VIVO Development mailing list (`vivo-dev-all@lists.sourceforge.net`), so we can improve our distribution.

The build.properties file

The standard installation instructions specify that these properties are required in `build.properties`

- `vitro.core.dir`
- `webapp.name`
- `tomcat.home`
- `vitro.home`

However, if you are building with `ant distribute`, then only these are required:

- `vitro.core.dir`

- `webapp.name`

`tomcat.home` is ignored by the `distribute` target. You may choose to specify `vitro.home` in `build.properties`, or later, when you deploy VIVO (see Deploying VIVO). If you specify `vitro.home` in `build.properties`, you can override it when you deploy, but you will receive a warning when VIVO starts, saying that `vitro.home` has been specified twice.

Running the build script

To build VIVO for other servlet containers, you will use one of these commands:

- `ant distribute` -- to incorporate changes since your previous build.
- `ant clean distribute` -- to do a full build from scratch

The build will produce a file named `distribution.tar.gz`, in the `.build` sub-directory of your VIVO distribution directory. This compressed archive contains these files:

- `vivo.war` -- a WAR file for the main VIVO application.
- `vivosolr.war` -- a WAR file for the Solr application.
- `vitrohome.tar` – a Vitro home directory that is ready for configuring,
- `solrhome.tar` -- a Solr home directory that is configured for use with VIVO.

The WAR files should be deployed to your servlet container. They may be renamed as desired when deployed. The TAR files will be unpacked to become your VIVO home directory and your Solr home directory.

3. Deploying Solr

The Solr application is packaged in `vivosolr.war` (see Running the build script). Deploy this file as required by your servlet container. The filename is not significant, and the file may be renamed as required by your container.

The Solr home directory is packaged in `solrhome.tar` (see Running the build script). Create a Solr home directory on your machine, and unpack this file into that directory. It is customary to use a `solr` sub-directory in your Vitro home directory, but this is not required. Note that the Solr home directory will contain VIVO's search index, so it may grow to be quite large.

You must tell Solr where to find the home directory. You can use one of two methods:

1. Set the sytem property `solr.solr.home` to the path of your Solr home directory.
2. Set a JNDI value at `java:comp/env/solr/home` to the path of your Solr home directory. For servlet containers, a JNDI prefix of `java:comp/env/` is assumed for all environment entries, so you will likely just specify a value for `solr/home`.

Which of these methods should you use? In general, it is easier to set a system property than an environment entry. However, a system property applies across the entire servlet container. If you want to deploy two instances of Solr in the same container, you will need to use environment entries to give each instance its own home directory.

The Solr application must be authorized to read and write to the Solr home directory.

4. Deploying VIVO

The VIVO application is packaged in `vivo.war` (see Running the build script). Deploy this file as required by your servlet container. The filename is not significant, and the file may be renamed as required by your container.

The Vitro home directory is packages in `vitrohome.tar`. You must create a `runtime.properties` file in the Vitro home directory. The contents of this file are exactly as specified in the standard installaion instructions. Pay attention to the value of `vitro.local.solr.url`. This must point to the base of the Solr application, as you have deployed it.

You must tell VIVO where to find the Vitro home directory. If you did not specify this in `build.properties`, you can use one of two methods:

1. Set the sytem property `vitro.vitro.home` to the path of your Vitro home directory.
2. Set a JNDI value at `java:comp/env/vitro/home` to the path of your Vitro home directory. For servlet containers, a JNDI prefix of `java:comp/env/` is assumed for all environment entries, so you will likely just specify a value for `vitro/home`.

Which of these methods should you use? In general, it is easier to set a system property than an environment entry. However, a system property applies across the entire servlet container. If you want to deploy two instances of VIVO in the same container, you will need to use environment entries to give each instance its own home directory.

The VIVO application must be authorized to read and write to the Vitro home directory.

Note: Session object in VIVO are not serializable, and therefore cannot be made persistent. The standard build process tells Tomcat not to attempt to persist Sessions. You may need to set a similar configuration option in your servlet container.

3.9 Adding OpenSocial gadgets to VIVO

Instructions for connecting VIVO and Open Research Networking Gadgets

This document contains instructions on how to configure your VIVO installation to use OpenSocial gadgets.

VIVO uses an extension of the OpenSocial protocols called Open Research Networking Gadgets, or ORNG. ORNG is a project of the Clinical & Translational Science Institute at the University of California, San Francisco. You can find out more about the ORNG project at their web site, http://www.opengadgets.org/index.html

ORNG supports gadgets using a modified version of Apache Shindig. These instructions tell you how to install the Shindig-ORNG web application, and how to configure it to work with VIVO.

Note: these instructions assume that you will be installing VIVO on Tomcat using the standard installation procedure. VIVO does not yet support ORNG on containers other than Tomcat.

3.9.1 Installing and Configuring

Configuring VIVO to support ORNG requires several steps, including additions to the VIVO properties, modifications to Tomcat, creating a security key for safe network operations, and running a build script.

Create database tables and procedures

Shindig-ORNG uses several database tables in MySQL to store its data: which gadgets appear on which pages, how large are the gadgets, what information applies to each individual, and more. Shindig-ORNG also creates stored procedures in MySQL. These are small pieces of code that simplify the use of the database tables.

In the VIVO distribution directory, a file called `vitro-core/opensocial /shindig_orng_tables.sql` contains SQL commands that create the tables and stored procedures for Shindig-ORNG to use.

Tell MySQL to process this file with a command like this:

```
mysql -u username -p database < sql_file
```

So, if your current directory is the VIVO distibution directory, and your VIVO database is `vivoDb` and your MySQL user account is `vivoUser`, then you might use the command this way:

```
mysql -u vivoUser -p vivoDb < vitro-core/opensocial
/shindig_orng_tables.sql
```

MySQL will prompt you for the password for your MySQL user account, and then process the file.

You may want to start your gadget collection with some example gadgets that have been developed by the ORNG group. The file called `vitro-core/opensocial /shindig_example_gadgets.sql` contains SQL commands that will add these gadgets to your system's configuration.

If you want to load these example gadgets, you can use a command similar to the previous one:

```
mysql -u vivoUser -p vivoDb < vitro-core/opensocial
/shindig_example_gadgets.sql
```

As before, MySQL will prompt you for the password for your MySQL user account, and then process the file.

Create configuration directory and key file

In your VIVO home directory, create a directory called *shindig*. Under that, create directories called *conf* and *openssl*. Your VIVO home directory will look something like this:

```
[VIVO home directory]
 |
 |--shindig
 |   |
 |   |--conf
 |   |
 |   |--openssl
 |
 |--solr
 |
 |--uploads
```

Shindig-ORNG uses an encryption key to insure that the communication between the gadget and the server is secure. You should create a file that contains the encryption key, and store that file in the *shindig/openssl* directory that you created.

On Unix-based systems (like Linux or Mac OS X), this command will create an encryption key from a random seed:

```
dd if=/dev/random bs=32 count=1 | openssl base64 > [key-file]
```

For example, if your VIVO home directory is */usr/local/vivo/data*, you might use the command this way:

```
dd if=/dev/random bs=32 count=1 | openssl base64 > /usr/local/vivo
/data/shindig/openssl/securitytokenkey.txt
```

If your VIVO installation is installed on a machine that runs Microsoft Windows, you will need to find another way to create an encryption key. The easiest way might be to find a Unix-based machine, issue the command above, and copy the resulting file to your Windows machine.

Modify Tomcat settings

The Shindig-ORNG application must know where to find the configuration file that you created in Step I. It must also know its own URL, so that URL can be inserted into the gadgets.

This information is provided through startup parameters in Tomcat. With most installations of Tomcat, the "setenv.sh" or "setenv.bat" file in Tomcat's bin directory is a convenient place to set these parameters. *If this file does not exist in Tomcat's bin directory, you can create it.*

Here is an example of the setenv.sh file, showing only the Shindig-ORNG requirements:

```
export CLASSPATH=/usr/local/vivo/data/shindig/conf
export CATALINA_OPTS="-Dshindig.host=localhost -Dshindig.
port=8080"
```

Here is the equivalent file for an installation in Windows.

```
set CLASSPATH=C:\vivo\data\shindig\conf
set CATALINA_OPTS=-Dshindig.host=localhost -Dshindig.port=8080
```

This assumes that your setenv file was empty before starting this process, and that you used the default location for the Shindig-ORNG configuration file in Step I. In fact, it's more common for the setenv file to contain other parameters besides those used for Shindig-ORNG. In that case, it might look more like this:

```
export CLASSPATH=/usr/local/vivo/data/shindig/conf
export CATALINA_OPTS="-Dshindig.host=localhost -Dshindig.
port=8080 -Djava.awt.headless=true -Xms1024m -Xmx1024m -XX:
MaxPermSize=128m"
```

Or, for Windows:

```
set CLASSPATH=C:\vivo\data\shindig\conf
set CATALINA_OPTS=-Dshindig.host=localhost -Dshindig.port=8080 -
Djava.awt.headless=true -Xms1024m -Xmx1024m -XX:MaxPermSize=128m
```

Configure VIVO

In the VIVO distribution directory, the file called *build.properties* contains configuration options for the VIVO application. You must set some additional parameters so VIVO will be able to communicate with Shindig-ORNG.

Property name	**`OpenSocial.shindigURL`**
Description	The base URL that VIVO will use when contacting the Shindig-ORNG application. Usually, this is the same host and port number as VIVO itself, with a context path of *shindigorng*
Default value	NONE
Example value	`http://localhost:8080/shindigorng`

Property name	**`OpenSocial.tokenService`**
Description	The host name and port number of the Token Service that Shindig-ORNG creates. Note that a value of *localhost* or *127.0.0.1* will not work. You must provide the actual host name of your machine, followed by *:8777*
Default value	NONE
Example value	`myhost.mydomain.edu:8777`

Property name	`OpenSocial.tokenKeyFile`
Description	The path to a key file that will be used to generate security tokens. This is the file that was created in Step I of this process.
Default value	NONE
Example value	`/usr/local/vivo/data/shindig/openssl/securitytokenkey.txt`

Run the deployment script

At the command line, from the top level of the VIVO distribution directory, type:

```
ant orng
```

to configure the Shindig-ORNG application and deploy it to Tomcat's webapps directory.

You must restart Tomcat so the main VIVO application will load the new settings in `runtime.properties`.

Does it work?

Startup tests

Start VIVO, and verify that you can see VIVO's home page in a browser.

On startup, VIVO runs a series of self-diagnostics, or "smoke tests". If these tests find any problems with the OpenSocial configuration, you will see a warning message instead of the VIVO home page.

Some of VIVO's "smoke tests" are run after the startup is finished, and may take up to a minute to complete. If one of these tests fails, you will see the warning message as you navigate from one VIVO page to the next.

If one of the OpenSocial tests fails, you may continue to use VIVO, but it is likely that no gadgets will be shown. You can review the warning message by selecting the "Startup Status" link from the "Site Admin" page.

Search page

If you loaded the example gadgets, you should be able to see the "Google Search" gadget on the Search Results page in VIVO.

Every VIVO installation comes with a geographic data model, so type "Chile" in the search box, and view the results. Near the bottom of the page, you should see the "OpenSocial" section heading, and beneath it, a gadget offering "Full Text Search Results". This gadget does a google search at UCSF, using the search term that you entered. Again, this gadget is just an example, to show what is possible with OpenSocial gadgets and VIVO.

The first time you bring up the search page, it may take several seconds for the gadget to appear. After the first time, the gadget response should be much faster.

Profile page

If your VIVO installation contains profiles of people, you can see several gadgets on their profile pages. You must be logged in to VIVO, with authority to edit the profile you are viewing.

Go to a personal profile page in VIVO. If you loaded the example gadgets, you will see the "OpenSocial" section heading above the property lists, with an assortment of example gadgets available for experimentation.

As with the search page, the first appearance of the gadgets may be slow.

Troubleshooting

If the gadgets do not appear as you expect, look for these symptoms, and check for the corresponding possible causes.

Symptoms	**P**
• The "OpenSocial" heading does not appear on Individual page or in search results. • Tomcat log files do not contain errors.	
• Gadgets do not appear on Individual page or in search results. • Tomcat "localhost" log file contains an error message: `Unable to load properties: shindigorng.properties`	
• Dialog box appears in the browser with the message: "Error 500 reading application data: internalError"	

Symptoms	P
• Tomcat "catalina" log file contains an error message: `java.sql.SQLException: Access denied for user`	

• "Smoke tests" fail at startup.
`Token key file for Shindig does not exist`
• Pages that display gadgets "hang" in the browser.
• Tomcat "localhost" log file contains error messages, including:
`com.google.inject.CreationException: Guice creation errors`	

• Gadgets do not appear on Individual page or in search results • vivo.all.log contains an error message: `MySQLSyntaxErrorException: Table 'vivo.orng_apps' doesn't exist`	
• Gadgets do not appear on Individual page or in search results • vivo.all.log contains an error message: `java.net.ConnectException: Connection refused`	

3.9.2 Changing the gadget configurations

VIVO will look at tables in MySQL to determine what gadgets should be made available, where they should appear, how big they will be, and much more. At this time, VIVO doesn't provide a user interface to edit the contents of these tables. Administrators will need to use a MySQL admin client, or SQL commands, to set these parameters.

The tables are named *orng_apps* and *orng_app_views*, and are described in the following sections.

The orng_apps database table

This table acts as a dictionary of the available gadgets. It includes the name and ID of the gadget and where the source code is stored on the web.

Field	Type	Usage
appid	int(11)	Identifies the gadget. In particular, this will be used to determine which rows in the *orng_app_views* table should apply to this gadget.
name	varchar (255)	A user-friendly name for the gadget. This will be displayed in the gadget's "Title Bar".
url	varchar (255)	The location where the gadget's contents and behavior are defined.
PersonFilterID	int(11)	deprecated - usually set to *NULL*
enabled	tinyint (1)	If set to 0, this gadget will never be displayed. If set to 1, it is displayed according to the rules in the *orng_app_views* table.
channels	varchar (255)	Keywords that identify the communication channels between the gadget and VIVO.

The orng_app_views database table

This table tells how, where, and when to display the gadgets that are described in *orng_apps*. Each row in this table is a "view", describing a single gadget and the rules that determine whether the gadget will be displayed on a particular page.

Note: If a gadget is described and enabled in the *orng_apps* table, but has no records in the *orng_app_views* table, the gadget will be displayed without restriction on all ORNG-enabled pages. This can be helpful when developing a new gadget. To avoid this, either

- remove the gadget from *orng_apps*, or
- set the *enabled* flag in *orng_apps* to *0*, or
- create a rule for the gadget in *orng_app_views.*

Field	Type	Usage
appid	int(11)	Determines which gadget in *orng_apps* is affected by this rule.

Field	Type	Usage
viewer_req	char (1)	What requirements must the viewer satisfy in order to see this view? • *NULL* -- There are no requirements on the viewer. • *'U'* -- The viewer must be logged in to VIVO. • *'R'* -- The viewer must be logged in, and must be registered as a user of this gadget.
owner_req	char (1)	What requirements must the owner of this page satisfy in order to see this view? • *NULL* -- There are no requirements on the owner of the page. • *'R'* -- The owner of the page must choose to display this gadget to the public. • *'S'* -- The viewer must be the owner of the page being viewed.
page	varchar (50)	What page does this rule apply to? A single gadget might have several views, but no more than one view per page. Recognized values are • *individual* -- The profile page of an individual, when it is not in "edit" mode. This applies when the viewer is not logged in, or does not have the right to edit the profile page. • *individual-EDIT-MODE* -- The profile page of an individual, when it is in "edit" mode. This applies when the viewer is logged in as an administrator or other privileged user, or as the owner of the profile page. • *search* -- The search results page. • *gadgetDetails* -- A page that contains only the selected gadget. This usually occurs when the user clicks on an icon or a link that expands the gadget to full-page mode.
view	varchar (50)	What is the view-mode of the gadget? These are defined as part of the OpenSocial standards.

Field	Type	Usage
		• *profile* -- The "standard" view, commonly used on the profile page of an individual • *small* -- The "condensed" view. • *home* -- The view which allows the user to change the gadget's settings. • *canvas* -- The "expanded", commonly used when the gadget is the only thing on a page.
closed_width	int(11)	How wide is the gadget when it is closed? (in pixels)
open_width	int(11)	How wide is the gadget when it is open? (in pixels)
start_closed	tinyint (1)	When the page is first loaded, is the gadget open or closed (1 = closed, 0 = open)
chromeId	varchar (50)	The gadget will be displayed on the page inside a with this id. Note: the page must contain this and its contents will be replaced with this gadget (or gadgets).
display_order	int(11)	If more than one gadget has the same chromeId, they will be displayed in order by this field.

3.9.3 Additional Considerations

Some things to be aware of when working with OpenSocial gadgets.

Re-running the deployment script

The OpenSocial framework relies on several of the settings in the *build.properties* and *runtime.properties* files, in addition to the ones that are explicitly linked to it.

Each time you change the settings in *build.properties* or *runtime.properties*, you should re-deploy the framework with

```
ant orng
```

Resetting the gadget cache

For efficiency, VIVO reads the gadget configuration only when it starts up. VIVO keeps a copy of the database tables in memory, for efficiency.

This means that if you change the gadget configuration in the database tables, you must either tell VIVO to read the tables again. Direct your browser to the orng/clearcache page within VIVO. For example,

```
http://localhost:8080/vivo/orng/clearcache
```

VIVO will re-read the gadget configuration, and display the VIVO home page.

You can achieve the same effect by restarting VIVO.

Issues with Linked Open Data

TBD

Disabling the OpenSocial gadgets

If you decide not to use OpenSocial gadgets in your VIVO installation, there are several ways to deactivate them, depending on how firm your decision is, and how thorough you wish to be.

Disable the gadgets

You can disable any or all of the installed gadgets by setting the *enabled* flag in *orng_apps* to zero (see section II.i. The *orng_apps* database table).

To make this change take effect, restart Tomcat, or clear the OpenSocial cache (see section III. ii. Resetting the gadget cache).

Disable the connection

Disabling the gadgets, as above, will remove essentially all of the OpenSocial processing within VIVO. To remove the remainder of it, you can disable the connection between VIVO and the OpenSocial service. Do this by removing or commenting the *OpenSocial* properties in *build. properties* (see section I.iv. Configure VIVO).

To make this change take effect, re-deploy VIVO and restart Tomcat.

Remove the OpenSocial webapp from Tomcat

Disabling the connection, as above, will remove all of the OpenSocial processing from your VIVO requests. However, you may still see that Tomcat takes longer to start up, and requires more memory.

To remove the OpenSocial webapp from Tomcat,

- stop Tomcat;
- in the [tomcat]/webapps directory, delete shindigorng.war and the shindigorng sub-directory;
- start Tomcat.

Clean up the remnants

To remove all traces of OpenSocial from your VIVO installation, you should take the steps outlined above, and also:

- Remove the *orng* tables from your MySQL database.
- Remove the changes to Tomcat by restoring your *setenv* file to its previous state.
- Remove the *shindig* directory and subdirectories from your VIVO home directory.

These steps will have no appreciable effect on the operation of VIVO, Tomcat, or MySQL. However, if these artifacts are not removed they could be a source of puzzlement for future VIVO maintainers.

3.10 VIVO in a language other than English

VIVO can display pages and data in different languages.

VIVO can be installed and configured to provide limited support for languages other than English. As of release 1.6, the support is limited to:

- Displaying user-oriented pages in other languages
- Filtering data from the data models, to display the most language-relevant data.

As of release 1.6, there is no support (or limited support) for:

- Displaying administrative pages in other languages
- Interactive editing of language-specific data.

Language support in VIVO will be extended in future releases.

3.10.1 Configuring VIVO for another language

Language support in VIVO requires three steps of configuration

1. The language files must be added to your VIVO installation
2. The build script must be directed to include the language files
3. The runtime properties must specify how to use the language

For more information about language support, consult the VIVO Customization Guide in the section called Language Support.

3.10.2 Adding language files to your installation

Is the language distributed with VIVO?

Demonstration language - Google Spanish

In ViVO release 1.6, a set of files for a "demonstration" language is included in the distribution. These files were created using the Google Translate service to produce Spanish words and phrases for VIVO. These files are likely to be grammatically incorrect, and should not be used in a production instance of VIVO. They are included only as a proof of concept.

Is the language available at GitHub?

GitHub holds two repositories of language files. Files for VIVO are stored in `vivo-project/VIVO-languages`, and files for Vitro are stored in `vivo-project/Vitro-languages`. You must use both sets of files for language support in VIVO.

Each repository is structured by release number, with sets of language files that are appropriate for that release. So for example, we see:

```
vivo-project/VIVO-languages
    vivo-1.6
        en-US
        es-GO
vivo-project/Vitro-languages
    vitro-1.6
        en-US
        es-GO
```

As you can see, the repositories contain two sets of language files for release 1.6. The files labeled `es-GO` are for the demonstration language (Google Spanish) mentioned above. The files labeled `en-US` are for American English. These are not useful to add to VIVO, since American English is already the default language. They were created as a basic template that translators may use when creating files to support other languages.

Creating language files

If the language you want to support is not available in the GitHub repositories, you should consider creating the files for your installation. The task involves translating a few hundred words and phrases and about a dozen short pages. For more information, consult the VIVO Customization Guide in the section called Language Support.

3.10.3 Including languages in the build

The language files will not be included when VIVO is built, unless they are specified in `build.properties`.

Property name	**`languages.addToBuild`**
Description	Languages (in addition to American English) that will be built into your VIVO site. The languages must be found in the `languages` directory of the VIVO distribution.
Default value	NONE
Example value	`es_GO`

3.10.4 Specify languages at run-time

You must set values in `runtime-properties` to tell VIVO how to support other languages.

Property name	**`RDFService.languageFilter`**
Description	If this is true and the VIVO finds values in more than one language for a particular property, VIVO will display the value that is best suited to the user's preferred language. If this is false, all values are displayed.
Default value	`false`
Example value	`true`

Property name	`languages.forceLocale`
Description	Force VIVO to prefer a specific language or Locale. If this is set, users will not be allowed to choose a language or Locale, and browser settings for Locale will be ignored. This is useful if you want your installation to support only one language, and that language is not American English.
Default value	NONE
Example value	`fr_FR`

Property name	`languages.selectableLocales`
Description	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 is useful if your installation supports more than one language.
Default value	NONE
Example value	`en, es, fr_FR`

3.11 Other installation options

An assortment of installation options that aren't necessary for your first installation, but might be helpful later.

These are properties that many sites will not need to modify.

Property name	`homePage.geoFocusMaps`
Description	On the VIVO home page, display a global map highlighting the geographical focus of foaf:person individuals.
Default value	`enabled`
Example value	`disabled`

Property name	`multiViews.profilePageTypes`
Description	**MultiViews for** foaf:person **profile pages.** VIVO supports the simultaneous use of a full foaf:Person profile page view and a "quick" page view that emphasizes the individual's own 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, usually for a small fee.**
Default value	`disabled`
Example value	`enabled`

Property name	`http.createCacheHeaders`
Description	Tell VIVO to generate HTTP headers on its responses to facilitate caching the profile pages that it creates. This can improve performance, but it can also result in serving stale data. The default is `false`. For more information, see the VIVO wiki page: Use HTTP caching to improve performance
	`false`

Property name	**http.createCacheHeaders**
Default value	
Example value	`true`

Property name	**harvester.location**
Description	If you intend to run the VIVO Harvester utility from the VIVO ingest menu, you must tell VIVO how to find the Harvester code. An absolute file path, pointing to the root directory of the Harvester installation. You must include the final slash.
Default value	NONE
Example value	`/usr/local/vivo/harvester/`

Property name	**visualization.topLevelOrg**
Description	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 guess at the top level organization in your instance. If you're unhappy with this selection, set the value of the property 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`
	VIVO will infer the top level organization in your instance.

Property name	**visualization.topLevelOrg**
Default value	
Example value	`http://vivo.psm.edu/individual/n2862`

Property name	**visualization.temporal**
Description	The temporal graph visualization can require extensive machine resources. This can have a particularly noticable impact on memory usage if • The organization tree is deep, • The number of grants and publications is large. VIVO V1.4 (and later) mitigates this problem by the way of a caching mechanism and hence we can safely set this to be enabled by default.
Default value	`enabled`
Example value	`enabled`

Property name	**proxy.eligibleTypeList**
Description	Types of individual for which we can create proxy editors. If this is omitted, defaults to `http://www.w3.org/2002/07/owl#Thing`
Default value	NONE
Example value	`http://xmlns.com/foaf/0.1/Person, http://xmlns.com/foaf/0.1/Organization`