Upgrading VIVO

VIVO 1.10.x Documentation

Exported on 05/24/2024

Table of Contents

1 Overview

Upgrading to VIVO 1.10 requires upgrading your triple stores (content and configuration). A procedure is provided below. You should consider:

1. System downtime is required to upgrade the triple stores
2. SPARQL queries will need to be checked for use of string datatypes. See Data types for string and language[1]
3. Applications directly accessing the SDB triple store will need to be upgraded to use Jena 3 libraries.
4. With the upgrade to Jena 3.x, Java 8 is required. The Maven projects have been upgraded to state a dependency on version 8, and Maven will not run without it.
5. If you have customizations, please see
    a. Preserving Customizations During Build (see page 14) for processes to include your customizations in a VIVO build
    b. Additional Considerations (see page 8) for notes which may impact your customizations

1 https://wiki.lyrasis.org/display/VIVODOC110x/Data+types+for+string+and+language

2 Changes to runtime.properties

2.1 Location

For VIVO 1.10, the preferred location of `runtime.properties` has changed from the `<vivo_home>` directory to `<vivo_home>/config`. VIVO will raise a warning on startup if runtime.properties is found in `<vivo_home>` or both `<vivo_home>` and `<vivo_home>/config`. Continue startup by refreshing or clicking continue. Move `runtime.properties` into the config directory to avoid the warning. Note, jenatools expects to find runtime.properties in <vivo_home>.

2.2 Password hashing settings

VIVO 1.10 includes security enhancements to the way passwords are stored in VIVO. Three new settings are now required in runtime.properties. Add the following to an older version of runtime.settings (default settings shown):

runtime.settings (snippet)

```
argon2.parallelism = 1
argon2.memory = 1024
argon2.time = 1000
```

For a full explanation of the new settings, see example.runtime.properties[2].

2 https://github.com/vivo-project/VIVO/blob/rel-1.10.0-RC-1/home/src/main/resources/config/example.runtime.properties

3 Upgrading The Triple Stores

Upgrading the triple stores (there are two - content and configuration) involves dumping the contents of your stores, and then reloading them, using tools provided with VIVO.

In order to upgrade your triple stores, 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, 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 area to update your web application and home directory
   This will install the required tools into your `<vivo_home>/bin` directory.

3. To export your triple stores, 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 -e -d <vivo_home>
   ```

   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 VIVO configuration within the vivo home directory, and get the necessary information to connect to your configuration triple store, and your content triple store. If your triple store(s) are not SDB or TDB backed, they will be skipped.

   jena2tools will then extract the contents of the triple stores, and write them to `<vivo_home>/dumps`

   > In rare cases `jena2tools` will fail with a `java.lang.NullPointerException`.
   > This can occur if VIVO was not properly shut down before upgrading.
   > The preferred fix for this is to restore VIVO 1.9.x and see that it is properly started and shut down before upgrading. If this is not practical, another workaround is to delete the file named `tdbModels/journal.jrnl` in the Vitro/VIVO home directory. This may result in the loss of the most recent login information.

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
   - 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 content triple store using the triples produced by jena2tools

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

   - Delete all files in `<vivo_home>/tdbModels`. Jena3tools will rebuild your configuration tdbModels based on the content created by jena2tools

     ```
     rm -rf <vivo_home>/tdbModels
     ```

6. Reload your triple stores

   Having exported your triple stores, you can reload them using jena3tools, also available with VIVO 1.10, specifying the import command.

   ```
   java -jar jena3tools.jar -i -d <vivo_home>
   ```

   Arguments:

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

   -i - run in import mode

   On execution, the program will find your 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 VIVO instance.

   jena3tools will be present in <vivo home dir>/bin when you install the 1.10.

   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

   VIVO checks its filegraphs when starting. Restarting for the first time after an upgrade will take some time.

4 Additional Considerations

4.1 Overview

If your site does not have customizations, and does not use Jena-based applications, these considerations do not apply.

4.2 Upgrade 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.

4.3 Notes 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[3] and Json.org[4] 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.

while 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.

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

3 http://Sourceforge.net
4 http://Json.org

4.4 UI Changes

4.4.1 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.

4.4.2 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

4.4.3 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[5] in the future, for more dynamic visualization building.

4.5 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,

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

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

can be rewritten as

```
SELECT ?menuItem
       ?linkText
 WHERE {
    ?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.

4.6 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.

4.7 Java Dependencies

4.7.1 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.

4.7.2 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[6] for how this is done.

4.7.3 JSON Parsers

Four JSON parsers - GSON (com.google.gson), Glassfish (javax.json.Json, Sourceforge.net[7] (net.sf.json) and JSON.org[8] (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.

4.7.4 Replaced Dependencies

The CSV parser has been replaced with commons-csv.

4.7.5 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-

6 https://github.com/vivo-project/Vitro/blob/develop/dependencies/pom.xml
7 http://Sourceforge.net
8 http://JSON.org

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

5 Preserving Customizations During Build

Draft

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

5.1 Overview

VIVO can be customized in many ways. Most sites customize the interface using themes. See Creating a custom theme[9]. Many sites create ontology extensions to introduce local terminology. See Adding Additional Ontologies to VIVO[10]. Sites can add custom configurations for the Data Distribution API (see Data Distribution API[11]), providing custom APIs returning data you specify in formats you specify at web addresses you specify.

In each case, files distributed with VIVO are replaced or added. The process described here uses a Maven custom installer to replace and add your files to those distributed with VIVO.

5.2 Maven Custom Installer

5.3 Example – Custom Theme

5.4 Example – Local Ontology

5.5 Example – Data Distribution API

9 https://wiki.lyrasis.org/display/VIVODOC110x/Creating+a+custom+theme
10 https://wiki.lyrasis.org/display/VIVODOC110x/Adding+Additional+Ontologies+to+VIVO
11 https://wiki.lyrasis.org/display/VIVODOC110x/Data+Distribution+API