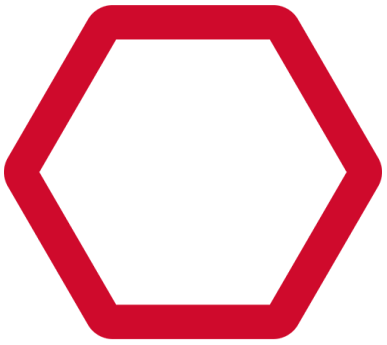


Symplectic Elements Harvester for VIVO:

Installation Guide

Symplectic, 4 Crinan Street, London,
N1 9XW, United Kingdom.

Contents

Introduction

This document describes the Elements VIVO Harvester and how to install it from a binary tar archive. It covers a variety of Installation and general configuration tasks and issues, but is not intended to be exhaustive documentation of the connector and its configuration options, nor is it intended to cover how to develop or alter the XSLT mapping scripts used by the connector.

There is also an appendix which covers our recommendations with regard to how you should configure your VIVO server to ensure that the integration with the Harvester works smoothly.

Connector Description

The connector fundamentally consists of two components:

1. The *Harvester*
2. The *Fragment loader*

Harvester

The Harvester performs several functions:

- Harvesting data from the Elements API.
- Translating the harvested data to a VIVO compatible RDF representation (triples).
- Loading those triples into a temporary triple store (Jena TDB).
- Creating change-sets (diffs) by comparing the current temporary triple store to the temporary triple store from the previous run (if present).
- Turning those change-sets into small fragment files (by default <2MB).

The Harvester is designed to minimize the load placed on the Elements API by making use of *delta updates* when pulling data from the Elements API (i.e. it will try to only pull data that has changed since the last time the Harvester was run). There are, however, various areas where this is not possible (e.g. Elements group/group membership information).

Note: *If you are connecting to an Elements API running an API Endpoint Specification earlier than v5.5 there can be some issues using delta updates. A small fraction of changed items can be missed and these will never appear in Vivo until a full refetch of all the data in Elements is run (these issues typically occur when data is being modified in Elements whilst a delta harvest is being run).*

The Harvester has multiple modes it can be run in by passing arguments on the command line:

- Default (No argument) : This mode will run a full harvest on the first run and a delta on subsequent runs
- --full : Forces the harvester to perform a full re-harvest, refetching all the data from

Elements
- --skipgroups : Instructs a delta run **not** to re-process the Elements group/group membership structures. Instead the harvester relies on a cache of group membership information from the previous run.
- --reprocess : Reprocesses the existing cached data against the current XSLT mappings without touching the Elements API (useful when developing custom mappings).

It is expected that these different modes will be combined to create a harvest schedule using cron or another scheduling utility. e.g:

- Run a --skipgroups delta every 3 hours.
- Run a normal delta every day at 4 am.
- Run a --full on the last Sunday of each month.

Fragment Loader

The Fragment Loader meanwhile has just one function - to load any fragments generated by the Harvester into VIVO via VIVO's Sparql update API. The fragment files generated by the Harvester are time-stamped and indexed, so they effectively form a queue which the Fragment Loader works through one by one. Note that the Fragment Loader is designed to be run as a daemon process and as such there are example files for integrating it with SystemD.java run

Installation Prerequisites

- A Java runtime environment on your server.
- The "flock" utility
- Access to and credentials for the source Elements API.
- Access to and credentials for the target VIVO instance (these credentials must correspond to a user who can use the Sparql update API, which by default is only available to the VIVO root user.

IMPORTANT: It has been found that Oracle's JDK 1.7.0 (runtime build 1.7.0-b147) has a problem with the concurrency methods used. Please ensure that a more recent JDK is used (1.7.0_03 and upwards have been tested with no apparent issues).

File System Permissions

Neither the Harvester nor the Fragment Loader generally require any special permissions to run. They need to be able to read and write files within the connector's installation folder, but that is all.

Note: *There is typically just one exception to this rule if you choose to have the connector write processed image files directly into the deployed webapp within tomcat. This is discussed in the section "Configuring Tomcat to Display Images".*

If you choose to run the connector components as a user other than root then for convenience it is useful to pick that user account ahead of time, change the ownership of the install archive [chown] to that user and perform all the installation steps below using that user account.

Note:

- You will need to specify the selected user when configuring the FragmentLoader to run via SystemD.
- You should ensure that you are editing the selected user's crontab when configuring an automated harvest schedule. There may be additional steps to enable cron for non root users (e.g. cron allow/deny) – this is outside the scope of this document

If you choose to install and run the connector as root, you should ensure that the Fragment Loader's SystemD integration is configured so that the process is run as root. When setting up an automated harvest schedule meanwhile, you should just add a file to /etc/cron.d to specify the system's cron schedule (which is run as root). You can still edit the root user's crontab file if you wish, but using the system cron directories is simpler and cleaner.

Performance Considerations

Disk IO

The Harvester can be disk intensive. The disk is constantly being read from and written to during various harvester operations:

- Harvesting raw data.
- Writing translated data.
- Populating temporary triple store.
- Writing Change sets and fragments.

This is particularly true during loading of the temporary triple store where, for a typically sized institution, several gigabytes of data will be read from the internal cache of translated triples and written into an on disk TDB triple store. As this process has to be performed every time the Harvester is run (even on a delta update) disk IO is critical to the overall performance of the harvester. We therefore very strongly recommend that you use SSD based storage and ensure that you have adequate storage capacity on your server.

Memory

The Harvester can also be very memory intensive, this is particularly true during the diff operation where the current temporary triple store is compared to the equivalent temporary store from the previous run. This process can result in both copies of the triple store being loaded into ram. To accommodate this the default elementsfetch.sh script ensures that the Java Virtual Machine (JVM) being used to run the process has access to up to 10 gigabytes of ram.

If your datasets ends up being significantly larger than 5Gb you may end up with poor performance (paging) or crashes during the diff operation (e.g. taking longer than 5-10

minutes). If you experience issues of this nature then you may need to increase the amount of RAM assigned to the JVM for the FetchAndTranslate operation. Similarly if your dataset is much smaller than 5Gb you may be able to reduce the amount of RAM being assigned to the JVM in the elementsfetch.sh file.

Installation

Copy the connector install media ***elements-vivo-harvester-(**version-info**).tar.gz*** to your VIVO server.

The following assumes you have placed it in the logged in user's home folder "~", it also assumes you are installing as a user with appropriate file system permissions (e.g. root).

Note: if you install via "sudo" commands, be aware that when invoking the Harvester applications from the command line you will need to use sudo or they will not have the necessary permissions to read and write to the install directory.

By convention we recommend installing in a "harvester" directory located alongside VIVO's home directory (the home directory, not the webapp), e.g:

```
mkdir /usr/local/vivo/harvester
```

Change directory to your new Harvester directory

```
cd /usr/local/vivo/harvester
```

Copy the archive containing the Harvester code into your newly created directory

```
cp ~/elements-vivo-harvester.tar.gz .
```

Extract the Harvester code from the tarball

```
tar -xvzf elements-vivo-harvester.tar.gz
```

at this point you should have a folder with a structure similar to this:

```
# drwxrwxrwx. 4 root root  85   Feb 15 13:12 examples
# drwxrwxrwx. 2 root root  4096 Feb 15 13:12 init
# drwxrwxrwx. 2 root root  4096 Feb 15 13:12 lib
```

Clean up

Delete or otherwise backup the tar archive, by convention we usually create a "downloads" folder alongside the examples, init and lib folders and keep any harvester tar-balls there

```
rm elements-vivo-harvester.tar.gz
```

Initialisation

If this is the first deployment of the Harvester on this box, navigate into the init folder and run initialise.sh, then return to the main install folder.

```
cd init
./initialise.sh
cd ..
```

This step copies the configuration properties files (from the *examples/example-config*) folder and the shell scripts (from the *examples/example-bin* folder) for both the harvester and the fragment loader into the main Harvester install folder. It also copies the example xslt translation scripts from the *examples/example-scripts* folder into a newly created *scripts* folder.
At this point your harvester installation folder should have the following contents (new files highlighted in *red*):

- *elementsfetch.properties*
- *elementsfetch.sh*
- examples
- *fragmentloader.properties*
- *fragmentloader.sh*
- init
- lib
- *scripts*

Configuration

There are three primary areas that need to be configured before running the connector:

1. The Harvester.
2. The Mapping Scripts.
3. The Fragment Loader.

You may also need to make some minor configuration changes to VIVO and its hosting platform tomcat.

Configuring the Harvester

Edit the **elementsfetch.properties** file using the text editor of your choice, e.g.

```
sudo nano elementsfetch.properties
```

At a minimum you will need to configure how to connect to the Elements API:

- **apiEndpoint** (Elements API URL).
- **apiVersion** (set to the version of Elements API that the URL above represents).
- **apiUsername/apiPassword**
  (the credentials needed to access the above API - relevant if endpoint is https).

If you have custom crosswalks you may need to alter the xslt pipeline being used for the translation to VIVO's data structure

- **xslTemplate** (path to the entry point of the xsl mapping script being used).

You may also wish to configure which data is to be fetched from Elements and how it is transferred to VIVO, please see Appendix B or the inline comments in the **elementsfetch.properties** config file for details.

Note: *Appendix B contains a listing of the configuration options for the Harvester.*

Configuring the Mapping Scripts

Note: If you have custom crosswalks then this information may not directly apply, but you will generally still need to customise your mapping scripts to some degree to ensure that the output is suitable for your VIVO system.

To configure the default crosswalks you will want to edit the file scripts/example-elements/elements-to-vivo-config.xml using the text editor of your choice.

```
nano scripts/example-elements/elements-to-vivo-config.xml
```

Note: it is the .XML file not the .XSL file you want to edit.

At a minimum you will need to configure:

- **defaultBaseURI**:
  The base URI that should be used when generating identifiers to represent items in VIVO (this should usually match the URI configured in the target VIVO's runtime.properties file).
  Note*: Alternatively you can configure the baseURI externally to the crosswalks by adding the parameter "xsl-param-baseURI" to* ***elementsFetch.properties****.*
  The default crosswalks are explicitly set up to enable this.

WARNING: **Make certain that you have configured the base URI correctly before running the initial harvest as altering it will mean that the URI's of all the data created by the Harvester will change, resulting in a very large change set to be transferred to VIVO.**

You often also want to configure:

- **internalClass**
  The class that internal users and groups should be tagged with to allow filtering in browse pages within VIVO.
  Note: *The internalClass can also be configured externally using the parameter xsl-param-internalClass in* ***elementsFetch.properties***

- **Label-schemes**
  How should label schemes be treated by the mappings.

You can also alter:

- **record-precedences/data-exclusions**
  Which Elements record is used as the selected source of data for VIVO if several are present.
- **Journal-precedences**
  Which type of journal information should be used to create publication venue information.
- **Publication-types**
  How Elements publication types are mapped to VIVO classes.
- **Organization-types**
  How specific organisations are represented as VIVO classes.

Please refer to the inline comments within the XML config file for more details.
More extensive changes to the mapping scripts may of course be necessary to meet your particular needs.

Configuring the Fragment Loader

Edit the **fragmentloader.properties** file using a text editor of your choice, e.g.

```
nano fragmentloader.properties
```

At a minimum you will need to configure how to connect to VIVO's sparql API:

- **sparqlApiEndpoint**
  A URL where the target VIVO application can be reached (usually something like localhost:8080/vivo or similar). Note that the process will append the correct relative path to the sparql API if it is omitted.
- **sparqlApiUsername/sparqlApiPassword**
  (the credentials of the VIVO user with permission to access the update API - in a default VIVO installation, this will just be the VIVO root user (see the rootUser.emailAddress property in VIVO's runtime.properties file).

WARNING: **When the Fragment Loader uploads data, it passes the configured username and password over the network. If the sparql API endpoint being targeted is not secure (e.g. if it is plain http) then there is a risk that those credentials could be compromised. To avoid this ensure that either the endpoint is protected by SSL (e.g. using an Apache reverse proxy server) or that you only access the endpoint via localhost or an SSH tunnel.**

Additional Configuration

Configuring Tomcat to Display Images

If the Harvester is processing user photos from Elements (which it will be unless you have set elementsImageType to "NONE" in **elementsfetch.properties**) it will generate user images suitable for use with VIVO on disk. These images need to be made available to the Java servlet container (e.g. Tomcat) hosting VIVO in a specific manner, so that they will display correctly.

By default the harvester will store the processed images in the folder

```
data/harvestedImages.
```

These images need to be made available within the Tomcat application hosting VIVO at the appropriate relative path ("/harvestedImages" by default) e.g:

```
http://localhost:8080/vivo/harvestedImages
```

There are two main ways to achieve this:

1. Create a symlink named harvestedImages within the deployed VIVO webapp pointing to the folder where the Harvester places the processedImages, e.g (if VIVO is deployed as the "ROOT" webapp):

```
cd /usr/share/tomcat/webapps/ROOT
ln -s harvestedImages
/usr/local/vivo/harvester/data/harvestedImages
```

2. Copy/Sync the files so that they are physically stored within a folder named harvestedImages within the VIVO webapp.

If you choose to take the symlink option then you will also need to update the deployed VIVO webapp's context to specify that symlinking is allowed. To do this either update the context.xml file within the webapps META-INF directory or the ROOT.xml file within the /conf/Catalina/localhost directory depending on how you have deployed the webapps.

Once you have located the relevant file:

On Tomcat 7: Edit the main "Context" element to ensure that an allowLinking attribute is present and set to true e.g :

```
<Context allowLinking="true" ↩
docBase="/usr/share/tomcat/manually-deployed-webapps/vivo">
```

On Tomcat 8+: Add a "Resources" elements within the main "Context" element:

```
<Context>
   <Resources allowLinking="true" />
    ...
</Context>
```

Note: *The … represents the remainder of the "Context" element, which should not be altered.*

Note: You will also need to ensure that the user account running the Tomcat application has read access to the image files within the linked directory.

If instead you choose to use option 2 then it is easiest to edit **elementsFetch.properties** and specify the *vivoImageDir* parameter to point directly to the *harvestedImages* folder within tomcat.

Note: In this case you will need to ensure that the account being used to run the Harvester has write access to the target folder.

Configuring VIVO to Understand Elements Label Schemes

If you are using the ability of the default crosswalk to generate VIVO Subject/Research Areas from Elements label schemes, you generally want to configure VIVO so that it knows how to describe the "scheme".

To achieve this you need to add "triples" to VIVO's vocabularySource.ns file (found in the home/rdf/abox/filegraph directory). Each label scheme being mapped to VIVO should be assigned a unique URI in the "defined-by" attribute of the crosswalk config xml file.

```
nano /usr/local/vivo/home/rdf/abox/filegraph/vocabularySource.ns
```

The value of these URI's is up to you, provided it is a valid URI format, but something descriptive added to the end of the baseURI is standard practice for any mapped custom label schemes.

To allow VIVO to know how to represent the scheme this "defined-by" URI should be assigned an rdf:type of Thing and an rdfs:label representing the name of the scheme as it should be presented in VIVO, by adding the two relevant lines to vocabularySource.ns.

The file addToVocabularySource.ns in the crosswalk scripting directory is an example of what needs to be done. It contains examples for the mesh, fields of research (for) and science-metrix label-schemes.

Running the Processes

Running the Harvester

Once you have completed the configuration of **elementsfetch.properties** file you can run the Harvester by running the script `./elementsfetch.sh` within the main installation folder. You can pass the Harvester options (--full, --skipgroups or --reprocess) to this script on the command line.

The Harvester can run for a long time, so we recommend running it in a terminal multiplexing system such as "screen" or "tmux" so that you can leave your shell without terminating the program.

You can monitor the process by inspecting the relevant time-stamped log file in the "logs" directory.

Initial run advice

Your VIVO instance should be empty prior to the first run of `./elementsfetch.sh`. Subsequent executions of elementsfetch.sh will then perform differential updates, unless you request otherwise, but ONLY if you retain the state.txt file and the 'data' directory created by the process. If either of these gets removed, you should ideally start again with a clean, empty VIVO instance.

If you wish to clear down your VIVO instance and start again from scratch, you should remove the state.txt file and the 'data' directory and start again with an empty VIVO instance.

Fragment Loader Daemon

To transfer the fragments generated by running the Harvester to VIVO you should run the `./fragmentloader.sh` script once you have configured the **fragmentloader.properties** file.

Generally, however, you will want to run the fragmentloader as a constantly running daemon process, if your Linux distribution uses systemd, you will find advice and example integration files to assist you in doing this in *examples/example-integrations/systemd.* To use these, first edit the fragmentloader.service file in the editor of your choice:

```
nano examples/example-integrations/systemd/fragmentloader.service
```

Replace `%HARVESTER_INSTALL_DIR%` with the path where you have installed the Harvester (e.g. /usr/local/vivo/harvester)

Copy the systemd unit file into systemd's library, then enable the new unit:

- `cp examples/example-integrations/systemd/fragmentloader.service /lib/systemd/system`
- `systemctl enable fragmentloader`

You should now be able to control the loader with normal systemctl commands:

- sudo systemctl start fragmentloader
- sudo systemctl stop fragmentloader
- sudo systemctl status fragmentloader

You can monitor the process through both the systemctl status command or by inspecting the fragment-loader.log file in the logs directory that should appear beneath your

"harvester" install directory

User accounts

To alter the user running the Fragment Loader you will need to make appropriate changes to the ".service" file to add the desired User and or Group to the [Service] section of the systemD unit file prior to deploying and enabling it.

Loading Large Volumes of Data (e.g. initial load)

Note that when performing the initial load the re-inferencing and re-indexing processes within VIVO can be a significant bottleneck that increase the load time of the initial dataset by several days. To work around this we recommend the following order with regard to loading fragments for an initial load:

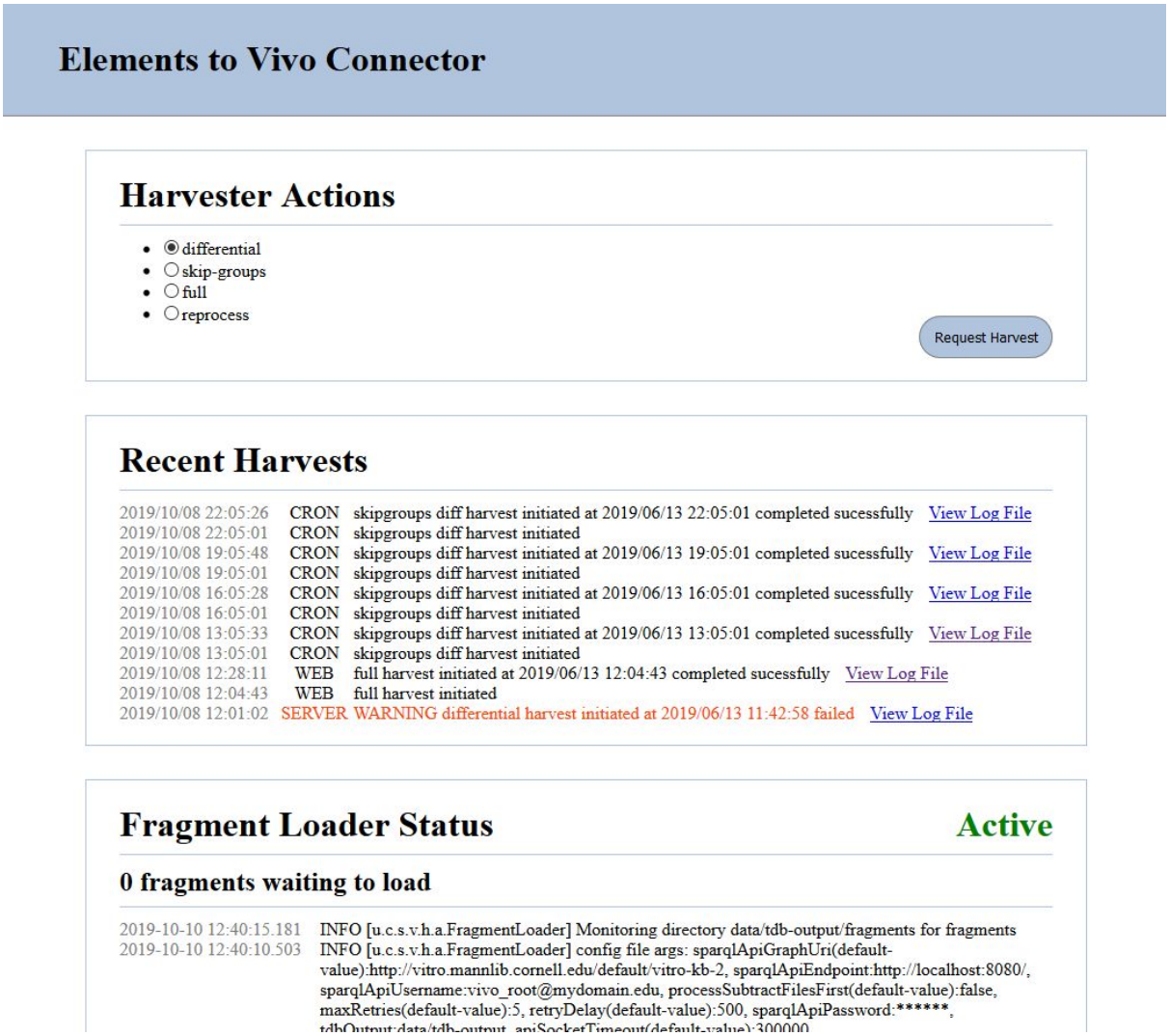
1. Turn off the Fragment Loader daemon. [`systemctl stop fragmentloader`]
2. Disable inferencing in your VIVO server:
   Edit WEB-INF/resources/startup_listeners.txt within your deployed VIVO web app within tomcat's webapps folder and comment out the line "edu.cornell.mannlib.vitro.webapp.servlet.setup.SimpleReasonerSetup".
3. Restart tomcat. [`systemctl restart tomcat`]
4. Log into VIVO as an Admin, Enable developer mode and disable indexing of changed triples:
   Developer mode is enabled by clicking the "Activate developer panel" link on the Site Admin page. To disable indexing of changed triples click on the yellow bar below the header to expand the panel, switch to the search tab, tick "Suppress the automatic indexing of changed triples" and click save settings.
5. Start the Fragment loader. [`systemctl start fragmentloader`]
   Ensure that all fragments have been processed before continuing.
   You can monitor the progress of the loader via systemctl status commands or via the log file.
6. Re-enable inferencing (i.e. reverse step 2).
7. Restart tomcat. [`systemctl restart tomcat`]
   ***Note**: This will disable developer mode and therefore re-enable indexing.*
8. Log into VIVO and re-compute the inferences (this will also re-index all the data).

You should adopt a similar procedure if you deploy mapping changes that generate a particularly large change set, resulting in a large number of fragments to be loaded (i.e. 100 – 1000s)

Web Interface

The Harvester ships with a set of perl-CGI scripts that can generate a simple "*Web interface*" from which you can: monitor the current status of the *Harvester* and *Fragment loader*; view log files of previous harvests; browse the data held within the harvester's internal caches; etc.

Note: *With suitable additional configuration, it is also possible to initiate harvests through this interface.*



The interface scripts, and details of how to deploy them (e.g. to Apache), secure access, etc can be found within the folder *examples/example-integrations/web-interface*.

Configuring for regular updates

Typically you will want to schedule runs of elementsfetch.sh using cron. A fairly normal update schedule would be to run a differential update once a day. If you are connecting to an Elements API using an Elements API Endpoint Specification prior to v5.5 we recommend running a full re-harvest (--full) fairly regularly (e.g. once a month). If your API is using an endpoint specification > v5.5 we still recommend running full re-harvests, but you should be able to schedule them less frequently (e.g. once every few months), as the v5.5 API

results are more reliable for "differential" updates.

Note: *The elementsfetch.sh script, used to launch the harvester, uses the "flock" utility to acquire an exclusive lock on a file, before proceeding. This ensures that we don't spawn multiple concurrent harvester processes (e.g. if a previous harvest is taking a long time to complete, and the next scheduled harvest attempts to start whilst it is still running).*

To edit the crontab for the current user use the command : crontab -e

Alternatively (if running the harvester as the "root" user) you can simply create a Vivo specific text file within /etc/cron.d/ and enter the relevant cron scheduler there.

Example crontabs:

You can find advice and an example crontab file in the folder *examples/example-integrations/cron/.* The information below is repeated and expanded on there.
Note: ***You will need to remove the # at the start of each line to enable that schedule:***

Run a delta at 4 am every Mon-Sat, Run a full at 3 am on a Sunday

```
# 00 04 * * 1-6     /usr/local/vivo/harvester/elementsfetch.sh
# 00 03 * * 0       /usr/local/vivo/harvester/elementsfetch.sh --full
```

Run a skipgroup delta every 3 hours at 5 past the hour between 7am and 11pm, Run a non group skipping every day at 4am and run a full once every month on the 1st Sunday of each month.

```
# 05 7-23/3 * * * /usr/local/vivo/harvester/elementsfetch.sh --skipgroups
# 00 04 * * *       /usr/local/vivo/harvester/elementsfetch.sh
# 00 03 * *  0      test $(date +\%d) -lt 8 && /usr/local/vivo/harvester/elementsfetch.sh --full
```

Note: *Pay attention to how the "test" and "date" utilities to check the date lies in the first week of the month in the final schedule. Also be aware that the '%' character must be escaped with a preceding "\" as it has special meaning in crontab files.*

Setting the "HARVEST_ORIGIN"

It is worth setting the environment variable "HARVEST_ORIGIN" to the value "CRON" whenever an automated harvest is run. This ensures that the "harvests_run" log will report the origin of the harvest correctly. This can usually be achieved by simply adding a line like:

```
HARVEST_ORIGIN=CRON
```

To the top of your crontab.
If this approach is not supported on your system you will need to prepend the command on each active schedule line with "export HARVEST_ORIGIN=CRON &&"

Appendix A : VIVO server specification and configuration.

This appendix covers our recommendations for how to set-up a linux server to host VIVO when you are planning to populate it via the Elements harvester. It is not intended to cover how to install VIVO (for which we recommend consulting the Duraspace documentation) and instead focuses on how best to configure the environment VIVO is running in to ensure smooth operation.

Performance Concerns

Recommended Specifications

Both VIVO and the Harvester can be both cpu and memory hungry and the Harvester in particular makes heavy use of disk both in terms of high IOPS and overall volume.

Therefore For a typical institution we would recommend a server similar to :

- i7\Xeon family 4 core hyperthreaded (8 thread) desktop/server class CPU
- 64 Gb of RAM
- 500Gb of SSD disk

Server Configuration

You should ensure that you have set up Tomcat and any reverse proxy server (e.g. apache) to be able to use a reasonable number of threads. On most distributions this is done by editing the file `/etc/security/limits.conf` and adding the following lines:

```
apache hard nproc 400
tomcat hard nproc 1500
```

You may also want to tweak the connection timeout and maxthreads properties of both Tomcat and apache.

Security Concerns

You should ensure that if your distro uses SELinux that it is configured so that it does not prevent any of the processes (across both VIVO and the harvester) from running correctly (how to do this is outside the scope of this document).

Database Configuration

Most institutions will host VIVO using an SDB triplestore backed by either a MySQL or a MariaDB database engine. Under these circumstances, we recommend that the database engine is configured with these options:

```
[mysqld]
innodb_buffer_pool_size=16G
innodb_flush_log_at_trx_commit=2
tmp_table_size=512M
max_heap_table_size=512M
query_cache_size=64M
thread_cache_size=16
performance_schema=0
innodb_log_file_size=2G
```

The exact configuration file where these belong can vary between Linux distributions, but typically they belong in `/etc/my.cnf.d/server.cnf`.

Note : Altering `innodb_log_file_size` is much easier to do before you ever start the database engine on your server. Changing it after you have used the database engine can be quite involved, see:

- https://dev.mysql.com/doc/refman/5.6/en/innodb-data-log-reconfiguration.html
- https://dev.mysql.com/doc/refman/5.6/en/innodb-data-log-reconfiguration.html

Security Concerns

We recommend that you run `mysql_secure_installation` to ensure that your server's mysql configuration is appropriately secure.

Tomcat Configuration

In order to ensure that VIVO has enough memory we recommend configuring it so that the JVM running Tomcat is started with these Java options:

```
-Xms2G -Xmx24G -XX:MaxPermSize=128m
```

The exact configuration file where these should be entered can vary depending on both your distribution and how you have installed tomcat. Typically, for a system where Tomcat is integrated with SystemD it will be `/etc/sysconfig/tomcat`, where you should add the above to the `JAVA_OPTS` being used.

Webapp deployment

One area where you should make a decision whilst installing VIVO is how you want the server to appear to the outside world in terms of its URL address space. If your server's DNS is http://institution.com are you comfortable with the pages of your VIVO instance being.

http://institution.com/**vivo**/display/publication186656

Or would you prefer to keep it as clean as possible and just have :

http://institution.com/display/publication186656

If you are happy with the former then you can install your VIVO application, in the typical manner as an application named "vivo" (or indeed whatever other name you prefer) in tomcat. In this model, If you would prefer people to still reach your VIVO instance when landing directly on http://institution.com your simplest option is to employ a reverse proxy server and add a redirect from "/" to "/vivo".

Alternatively if you prefer the latter option you need to deploy your VIVO application at the root ("/") path within tomcat. There are three ways to achieve this:

1. Edit tomcat's server.xml file (e.g. `/usr/share/tomcat/conf/server.xml`). *This method has been deprecated by Tomcat's developers*.
2. Manually deploy the webapp (keep it outside of the webapps folder) and add an external context fragment file to deploy it named "ROOT.xml".
3. Name the webapp "ROOT" instead of "vivo".

Option 3 is the simplest and the only real consequences are that your Tomcat log file names will be unusually named (ROOT.all.log and ROOTsolr.log), and you will need to update the path to VIVO's Solr server in VIVO's runtime.properties files (as you always do if you use any name other than "vivo" during installation).

VIVO Configuration

Once you have decided how to deploy your webapp you should ensure that your VIVO's runtime.properties file has its **`Vitro.defaultNamespace`** configured appropriately.

If the initial part of this value matches the URL where your VIVO server is deployed then you will be able to download the underlying RDF data directly from the server. Whether this is desirable or not is entirely the choice of the institution, but if you wish to make this possible you must use the correct value:

If you **do** want to allow direct access to the underlying RDF data and assuming your server's DNS is http://institution.com then the **`Vitro.defaultNamespace`** value should be:

http://institution.com/vivo/individual

If deployed as “vivo”, or

http://institution.com/individual

If deployed at the root path.

If you **do not** want to allow direct access to the underlying RDF data you may as well leave **`Vitro.defaultNamespace`** at its default value, unless the default happens to match the URL where you have deployed your VIVO server.

Note: Once you have configured **`Vitro.defaultNamespace`** you should ensure that the **defaultBaseURI** used by the harvester’s mapping scripts matches exactly.

List View Config Configuration

When creating "context" objects (e.g. authorships/editorships/roles), the default crosswalks make use of "VCard" objects, to represent people that do not have a profile in Vivo (e.g. any co-authors of an academic paper from another institution).

(*vcard<--authorship-->publication*) **Vs** (*user<--authorship-->publication*)

In fact, the default crosswalks deliberately create a VCard connected to the context object regardless of whether there is also a link to an actual Vivo user:

(*vcard<--authorship-->publication*) **Vs** (*user + vcard <--authorship-->publication*)

This is done to ensure that all authors/editors/etc are listed even if a particular user's relationship with a publication is marked as "hidden". It also allows for situations where the published name, as listed on the paper, does not match the user's name as it appears on their profile.

Unfortunately Vivo's out of the box support for "VCard" objects in context objects is not as complete as it might be, for example, it does not list "VCard's" in "editorship" objects at all. Additionally, not all aspects of Vivo cope well with context objects containing links to both a user object and an equivalent Vcard representation of the linked user.
These issues mostly relate to how various "listViewConfigs" process data:

- **listViewConfig-informationResourceInEditorship.xml**
  this does not handle "VCard" data at all.
- **listViewConfig-informationResourceInAuthorship.xml**
  this always list the user's "label" rather than the VCard name, potentially hiding the published name.
- **listViewConfig-relatedRole.xml**
  the behaviour here is buggy and inconsistent in terms of which names gets listed and which profiles get linked.

The "example-integrations/vivo-list-view-configs" directory contains some example

updated list view configs for Vivo v1.9.3 that can be used to address these issues.

Proxy Server Configuration

If Apache or a similar web server (e.g. nginx) is being used as a reverse proxy in front of Tomcat you should ensure that the appropriate paths are being forwarded onto tomcat.

If your app is deployed as "vivo" or similar in Tomcat then you can simply proxy that path, otherwise you should proxy the root path.

From here on we will assume you are using apache. The location of apache's configuration files varies between Linux distributions. You will usually either be editing the main Apache config file (e.g. */etc/httpd/conf/httpd.conf*) or adding a new break out file specific to your use case in the appropriate directory e.g. (*/etc/httpd/conf.d/vivo.conf*).

To setup proxying you should start with a simple VirtualHost definition something like this:

```
<VirtualHost *:*>
      ProxyPreserveHost On
      SSLProxyEngine On
      SSLProxyCheckPeerCN off
      SSLProxyCheckPeerName off
      ServerName localhost
</VirtualHost>
```

To this you will need to add configuration . (within the <VirtualHost> section)to perform re-directs and proxy requests to tomcat as appropriate:

You can use a tomcat's AJP proxy connector by adding:

```
ProxyPass "/vivo" "ajp://localhost:8080/vivo"
```

Or if deployed at root:

```
ProxyPass "/" "ajp://localhost:8080/"
```

Alternatively you can use a more traditional proxy config by adding something like:

```
ProxyPass /vivo/ http://0.0.0.0:8080/vivo/
ProxyPassReverse /vivo http://0.0.0.0:8080/vivo/
```

Or if deployed at root:

```
ProxyPass / http://0.0.0.0:8080/
ProxyPassReverse / http://0.0.0.0:8080/
```

If your webapp is deployed at "/vivo" within Tomcat but you want to forward any requests to your plain server DNS to the VIVO home page you should add a redirect rule, e.g. :

```
RewriteEngine on
```

```
RewriteCond %{REQUEST_URI} ^/$
RewriteRule / /vivo [R]
```

Add this ahead of the ProxyPass directives (i.e. earlier in the file).

Note : *Apache configuration is beyond the scope of this document. The above is just general advice.*

Timeouts

If you configure your Fragment Loader process so that it accesses your VIVO server's Sparql update API via the proxy server, you will need to ensure that the proxy timeout is set to a high value (e.g. 600s), at least for the path to VIVO's sparql update API (relative path /api/sparqlUpdate) to minimise the chance that data imports will time out, e.g.:

```
ProxyPass / http://0.0.0.0:8080/ timeout=600
```

If you are using the AJP connector to proxy to Tomcat you may wish to alter tomcat's server.xml to increase that connectors timeout value:
https://wiki.duraspace.org/display/VTDA/Running+VIVO+behind+an+Apache+server

Security Concerns

There is a risk that VIVO user credentials may be passed in the clear during both VIVO login and sparql update operations. You can remain secure in two ways:

1. Ensure that these operations only ever occur via localhost or SSH tunnel
   e.g. by configuring the Fragment Loader to use a localhost URL to access the Sparql API.
2. By deploying SSL certificates and protecting the appropriate paths in apache.
   This is outside the scope of this document.

Note: *There are example apache configuration files (for both plain http and https deployments) in the folder examples/example-integrations/apache-proxy.*

Appendix B : Harvester configuration options

The various parameters that can be set within the harvester's "elementsfetch.properties" config file are listed below:

Those highlighted in grey are rarely used, those highlighted in orange should only normally be used in a test environment.

Parameter Name Data Type	Description	Default
ELements API Configuration		
apiEndpoint String	**Required** The URL where the Elements API can be reached.	N/A
apiUsername String	***Required for secure apiEndpoints*** The username of the Elements ApiAccount being used to access the apiEndpoint.	N/A
apiPassword String	***Required for secure apiEndpoints*** The password of the Elements ApiAccount being used to access the apiEndpoint.	N/A
apiVersion String (e.g. "v5.5)	**Optional** The version of the Elements API Endpoint Specification used by the configured apiEndpoint. Will be verified if provided.	null
rewriteMismatchedPaginationUrls Boolean	**Optional** If the apiEndpoint is returning data containing self referential links (*e.g. nextPage*) that do not match the configured apiEndpoint, should the program error out (false) or rewrite the urls to use the baseURL configured in apiEndpoint (true).	true
ignoreSSLErrors Boolean	**Optional** Should the system error out if apiEndpoint is protected by an insecure SSL chain (e.g. self signed, expired or otherwise invalid SSL certificate).	false
apiRequestDelay Integer	**Optional** Number of ms to wait between requests to	250ms

	the API.	
apiSocketTimeout Integer	**Optional** Number of ms to wait for a response from the API before timing out	300000ms (5 mins)
refDetailPerPage Integer	**Optional** Number of items to request per page when a request is made to retrieve items from the API in "ref" detail level.	100
fullDetailPerPage Integer	**Optional** Number of items to request per page when a request is made to retrieve items from the API in "full" detail level. *Note: effective maximum is 25 as the elements API will never return more than 25 items in "full" detail.*	25
Data to be Retrieved From the API		
queryObjects String{,} comma separated list of strings	**Required** List of Elements objects categories to be processed by the harvester, e.g: *"users, publications, grants"* ***Note**: The supplied category names must be valid.*	N/A
elementsImageType Enum (profile|thumbnail|original|none)	**Optional** Which version of the Elements user profile photo should be used to generate the Vivo user photo?	"profile"
visibleLinksOnly Boolean	**Optional** Should invisible Elements relationships ever be sent to Vivo. *This represents a hard block implemented by the Harvester itself as opposed to relying on the crosswalks to produce appropriate output.*	false
repullRelsToCorrectVis Boolean	**Optional** Should relationships be re-pulled from the Elements API if any contained non-user objects are modified, this ensures "visibility" is tracked correctly	true
relTypesToReprocess String{,} comma separated list of strings	**Optional** List of Elements relationship-type names. Specifying the types of relationships that need to be "reprocessed" if either of the contained objects are modified. ***Note**: These are the types of relationships for which "extraObjects"* data is passed into the crosswalks when	"activity-user-association, user-teaching-association, publication-user-authorship"

	they are translated.	
Translation Process Configuration		
xslTemplate String (Path)	**Required** Path to the entry point of the XSLT crosswalks.	N/A
zipFile Boolean	**Optional** Should intermediate files held in the harvester's internal cache be "gzipped" to preserve space?	false
changeProtectionEnabled Boolean	**Optional** Whether the harvester should error out if the number of user or non-user objects being sent to Vivo changes by a fraction greater than specified (see below). ***Note**: This should typically be set to false if you are running an "opt in" process, as user numbers can vary dramatically between harvests, particularly during initial roll out.*	true
allowedUserChangeFraction Number decimal fraction between 0 and 1, e.g. 0.5	**Optional** The percentage change (e.g 0.5 = 50%) by which the number of users being sent to Vivo is allowed to change.	0.2 (20%)
allowedNonUserChangeFraction Number	The percentage change (e.g 0.5 = 50%) by which the number of non-user objects being sent to Vivo is allowed to change.	0.3 (30%)
vivoImageDir String (Path)	**Optional** The directory where the harvester will store processed photos suitable for use with Vivo.	data/harvestedImages/
vivoImageBasePath	**Optional** The URL path fragment (beneath the Vivo base path) where you will arrange for the generated photos to be made available in your web hosting environment.	/harvestedImages/
Who data to include in Vivo		
paramUserGroups Integer {.}	**Optional** List of Elements group ids. For any group	null *Corresponds to "all users"*

comma separated list of integers	listed here (or selected by one of the sibling regex parameters below), all "implicit" users of that group are candidates for being sent to Vivo, unless otherwise excluded *(e.g. by excludeUserGroups, publicStaffOnly, etc)*	
paramUserGroupRegexes CSV Fragment comma separated list of strings, with special characters (e.g. ",) encoded as per csv spec	Optional List of regex patterns. Any Elements group whose "*name*" matches one of the patterns is considered selected.	null
paramUserGroupDescriptionRegexes CSV Fragment	Optional List of regex patterns. Any Elements group whose "*description*" matches one of the patterns, is considered selected.	null
excludeUserGroups Integer {,} comma separated list of integers	Optional List of Elements group ids. For any group listed here (or selected by one of the sibling regex parameters below), all "implicit" users of that group are excluded from being sent to Vivo.	null
excludeUserGroupRegexes CSV Fragment	Optional List of regex patterns. Any Elements group whose "*name*" matches one of the patterns is considered selected.	null
excludeUserGroupDescriptionRegexes CSV Fragment	Optional List of regex patterns. Any Elements group whose "*description*" matches one of the patterns, is considered selected.	null
currentStaffOnly Boolean	Optional Should the harvester only transfer "current" staff to Vivo?	true
academicsOnly Boolean	Optional Should the harvester only transfer "academic" staff to Vivo?	true
publicStaffOnly Boolean	Optional Should the harvester only transfer "public" staff to Vivo?	true
Elements Group Translation		
	Optional	null *Corresponds to*

paramGroups Integer {.} comma separated list of integers	List of Elements group ids. For any group listed here (or selected by one of the sibling regex parameters below), that group and all its child groups (recursively down the tree) will be represented as internal organisations in Vivo unless they are explicitly excluded.	*"all groups"*
paramGroupRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*name*" matches one of the patterns is considered selected.	null
paramGroupDescriptionRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*description*" matches one of the patterns, is considered selected.	null
includeChildGroupsOf Integer {.} comma separated list of integers	**Optional** List of Elements group ids. For any group listed here (or selected by one of the sibling regex parameters below), all its child groups (recursively down the tree) will be represented as internal organisations in Vivo unless they are explicitly excluded. The selected group itself, however, will NOT be included in Vivo.	null
includeChildGroupRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*name*" matches one of the patterns is considered selected.	null
includeChildGroupDescriptionRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*description*" matches one of the patterns, is considered selected.	null
excludeGroups Integer {.}	**Optional** List of Elements group ids. For any group listed here (or selected by one of the sibling regex parameters below), that group and all its child groups (recursively down the tree) will be excluded from Vivo unless they are explicitly included. Group memberships of "Excluded groups" are rewired (where possible) to the nearest ancestral group that is being included in Vivo	null
excludeGroupRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group	null

		whose "*name*" matches one of the patterns is considered selected.	
	excludeGroupDescriptionRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*description*" matches one of the patterns, is considered selected.	null
excludeChildGroupsOf Integer {.}		**Optional** List of Elements group ids. For any group listed here (or selected by one of the sibling regex parameters below), all its child groups (recursively down the tree) will be excluded unless they are explicitly included. The selected group itself, however, will NOT be excluded from Vivo. Group memberships of "Excluded groups" are rewired (where possible) to the nearest ancestral group that is being included in Vivo.	null
	excludeChildGroupRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*name*" matches one of the patterns is considered selected.	null
	excludeChildGroupDescriptionRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*description*" matches one of the patterns, is considered selected.	null
exciseGroups Integer {.}		**Optional** List of Elements group ids. For any group listed here (or selected by one of the sibling regex parameters below), all its child groups (recursively down the tree) will be excluded unless they are explicitly included. The selected group itself, however, will NOT be excluded from Vivo. Group memberships of "Excised groups" are excluded from Vivo entirely.	null
	exciseGroupRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*name*" matches one of the patterns is considered selected.	null
	exciseGroupDescriptionRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*description*" matches one of the	null

		patterns, is considered selected.	
exciseChildGroupsOf Integer {.}		**Optional** List of Elements group ids. For any group listed here (or selected by one of the sibling regex parameters below), all its child groups (recursively down the tree) will be excluded unless they are explicitly included. The selected group itself, however, will NOT be excluded from Vivo. Group memberships of "Excised groups" are excluded from Vivo entirely.	
	exciseChildGroupRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*name*" matches one of the patterns is considered selected.	null
	exciseChildGroupDescriptionRegexes CSV Fragment	**Optional** List of regex patterns. Any Elements group whose "*description*" matches one of the patterns, is considered selected.	null
includeEmptyGroups Boolean		**Optional** Should Elements groups containing no users be represented in Vivo?	true

Undocumented Parameters

The list below is a set of additional parameters that should not typically be altered. They are included here for completeness but should only ever be used by people who are very familiar with the harvester.

Parameter Name	Default Value
rawOutput String (Path)	"data/raw-records/"
rdfOutput String (Path)	"data/translated-records/"
tdbOutput String (Path)	"data/tdb-output/"
otherOutput String (Path)	"data/other-data/"
useFullUTF8	false

Boolean	
maxXslThreads Integer	0
maxResourceThreads Integer	0
maxFragmentFileSize Integer	1228800
eligibilityFilterType Enum (label-scheme|generic-field)	null
eligibilityFilterName String	null
eligibilityFilterInclusionValue String	null
eligibilityFilterExclusionValue String	null