

# Harvester 1.0 Demo

Harvester webinar agenda

1. Get the harvester from source-forge and install / unpack it.
2. Demonstration of a JDBC fetch
3. Tool summaries
  - a. Fetches
    - i. JDBC
      1. Peoplesoft
      2. DSR (UF grant repository)
    - ii. PubMed
  - b. XSLTranslator
  - c. Transfer
  - d. Score
  - e. Match
  - f. ChangeNamespace
  - g. Diff
    - i. Graph Math updating
4. Demonstration of a Publication fetching
5. Q&A

Presenter1 - James Pence

Presenter2 - Chris Haines

Presenter3 - Michael Barbieri

## Harvester Script

## Intro and Setup

[Intro video webcast](#)

Presenter1: Welcome to the webinar concerning setup and use of the VIVO Harvester tools. Though the project has specific uses of these tools it is important to remember that they are designed to be able to be used separately as well as in scripts and as java libraries. For this presentation, We are assuming that you have a fresh working install of VIVO-1.2. If you already have a copy of the harvester then you can see this next part as a review. If you have the current virtual appliance, you already have a copy and it is under /usr/share/vivo/harvester..

## Goto Sourceforge

Presenter goes to Sourceforge page. < <http://vivo.sourceforge.net/> >

Presenter1: Here you see the base web page on sourceforge for the entire VIVO project, located on the web at <http://vivo.sourceforge.net/> . We are interested in going to the "Project Space", which is found by following the link on the upper right part of the screen.

Presenter goes to Sourceforge page

< <http://sourceforge.net/p/vivo/home/> >

by clicking on the link.

Presenter1: Now that we are in the project space, you can see a series of links to various parts of the VIVO project. Click on the Downloads link.

Presenter goes to Sourceforge downloads page

< <http://sourceforge.net/projects/vivo/files/> >

by clicking on the link.

Presenter1: Here you see a list of links. Click on the link named VIVO Harvester.

Presenter goes to Sourceforge VIVO Harvester page

< <http://sourceforge.net/projects/vivo/files/VIVO%20Harvester/> >

by following the link

Presenter1: The "tar.gz" is a compressed file, while the ".deb" file is the debian style package which can be installed on Debian and its derived systems like Ubuntu. The ".deb" file puts the harvester folders and files into the "/usr/share/vivo/harvester" directory, but the "tar.gz" file can be extracted into any location. The location of the harvester directory will be referred to as "the base Harvester directory" for the rest of the demonstration.

Presenter opens a command line terminal

to the "Harvester directory" and

does a listing of the files

## configure harvester

Presenter1: Within the "Harvester directory" you can see several folders. The ones of interest to us are the config and scripts. In order for the harvester to function it will have to be configured to work with the installed vivo on your system. In order to continue with the example database fetch you will have to configure some files. The first is the "vivo.xml" file found in the "[Harvester/config/models/vivo.xml](#)" it needs to comply with your deploy.properties which you used to set up your current version of VIVO.

Presenter opens deploy.properties of vivo.

*/usr/local/vivo/vivo-rel-1.2-final/* on virtual box

Finds the information.

In deploy.properties they are the properties listed with

"VidroConnection.DataSource"

Edits files to make them comply.

## Tools

Presenter1: In looking through a script you will see the array of tools that are the harvester. I will now describe them in summary one at a time. In the order they are used.

**RecordHandler:** A way to connect to a set of records which may be files in a folder, a database, or a triple store.

**Fetch:** The set of tools that take in the data and form it into simple text, which is stored in the recordhandler it is given.

**XSLTranslator:** Using Saxon's implementation of an xsl translator it translates between recordHandlers

**JenaConnect:** A java class that wraps and allows access to Jena Models. Can be used to do command line Sparql queries.

**Transfer:** Moves data to, from and between JenaModels.

**Score:** Finds and valuates potential matches. This is used to compare emails, names, ID numbers or any other properties.

**Match:** Uses the score data to alter the given model. Without match the score data would only be statistically interesting.

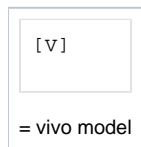
**ChangeNamespace:** Finds new URIs within VIVO to give to the unmatched data.

**Diff:** Returns the set of triples left if a model is removed from another model, without tampering with the original data.

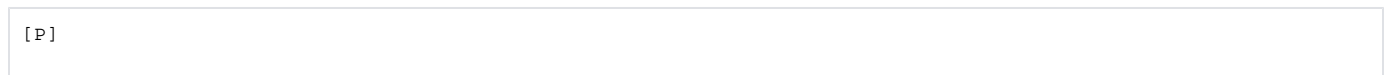
Presenter1: We do the updating based off graph math. The process is as follows:

## Graph Math Updating

(On the Diff page)



= new harvested model



= previous harvest model



= additions

```
[S]
```

= subtractions |

```
[A] = Diff [H] [P]
```

The additions are the triples that are in the new harvest when the old harvest triples are removed.

```
[S] = Diff [P] [H]
```

The subtractions are the triples that are in the old harvest when the new harvest is removed.

```
[P] = [V] - [S]
```

```
[P] =[V] + [A]
```

| The application of these to the old harvest will make the old harvest equal to the new harvest. |

```
[V] =[V] - [S]
```

```
[V] =[V] + [A]
```

| The application of these to the vivo model will make the information in the vivo model agree with the harvest without tampering with the data. |

Presenter1: Now we will walk you through a harvest of a simple sample database.

--Presenter2 takeover --docs.

## JDBCFetch

[JDBC video webcast](#)

Presenter2: One of our tools, [JDBCFetch](#), is designed to fetch data out of a standard relational database storing that data in a simple rdf/xml format that can later be translated to the VIVO ontology.

Presenter2: To demonstrate this tool, we have created an example database, containing HR data for a fictional university, and the example translation file and complete script.

## Setup demo Database

Presenter2: We do have to do a bit of setup before we can run this script however. First, we will need to setup a mysql database where we can store the sample HR data.

Presenter2: We can make this database using a variety of tools, but the only one I can guarantee you have is the commandline mysql tools, so I will demonstrate using that:

Presenter2: First, we need to connect to our mysql server. We will need a username that has permissions to create database and grant access to other users (the root account in my case)

```
>> mysql -u root -p
```

Presenter2: This will prompt you for the root mysql password. On the VIVO Appliance this is: vitro123

Presenter2: Now that we are logged in, we can create the sample database using this command:

```
>> CREATE DATABASE IF NOT EXISTS DemoDB
```

```
;
```

Presenter2: Then we create a user and grant it access to the database we just created:

```
>> GRANT ALL PRIVILEGES ON DemoDB.* TO DemoDB@localhost IDENTIFIED BY 'b8F5LdFPVEqHP3XX';
```

Presenter2: We can go ahead and exit the mysql prompt now:

```
>> exit
```

```
;
```

Presenter2: Whatever database, username, and password you set (I chose a stream of random numbers and letters 'b8F5LdFPVEqHP3XX'), we will need these later so keep that information handy.

Presenter2: Now that we have a database setup, we need to populate it with our sample HR data.

## Populate database

Presenter2: There is a file DemoDB.mysqlDump.sql at "<https://sourceforge.net/projects/vivo/files/VIVO%20Harvester/Example%20Files/>" which we will load into our database to show a simple JDBC harvest run.

Presenter2: Let's download that file into our current directory:

```
>> wget https://sourceforge.net/projects/vivo/files/VIVO%20Harvester/Example%20Files/DemoDB.mysqlDump.sql/download
```

Presenter2: Once that has finished downloading, we can push that dump into the database we just created. At the same time we will test the username and password we created:

```
>> mysql -u DemoDB -p DemoDB
```

```
<
```

DemoDB.mysqlDump.sql

Presenter2: This will prompt you for that password you created for the user (told you we would need it, but this is not the last time, so keep it around). This command may take a few moments depending on your system.

Presenter2: While that command is running, I am going to show you a bit of the structure of the sample HR data. I will use a tool called phpmyadmin to show the contents of the database.

– Presenter2 opens phpmyadmin to the DemoDB –

Presenter2: The sample HR database contains 5 tables: department, job, person, type, and user.

– Presenter2 navigates to the person table –

Presenter2: The person table stores information about people affiliated with the university, such as their names, phone, fax, preferred email, title, and name.

Presenter2: It also contains a flag that is set for if it is ok for this information to be published.

– Presenter2 navigates to the department table –

Presenter2: The department table stores very basic information about the organizational structure of this fictional university. We can see that each organization can have an id, a name, a type, and a super organization.

Presenter2: For the university itself (amusingly named the University of Sample Data), it has no value for the super organization because it is the top level org

Presenter2: The type value is a lookup from the type table, so let's look at the types.

– Presenter2 navigates to the type table –

Presenter2: Each type has an id, a value (the label for that type), and can have a super type (note that in this table, top level types use 0 as their super type)

– Presenter2 navigates to the user table –

Presenter2: The user table simply contains the person\_id, the person's login name, and whether this account is expired or not

– Presenter2 navigates to the job table –

Presenter2: The job table stores information about the current jobs that people hold at the university. Each record is a job, linking a department and a person (by their ids) and stores the type code (from type table) and start date.

– Presenter2 switches back to the console –

## Demo Harvest

Presenter2: By now we should have finished pushing this data into our database, so we can begin the actual fun of fetching it. All of this has been setup, remember, but don't worry, the rest is mostly done for you.

Presenter2: The example script we will be running is in the scripts/ folder under the base Harvester/ folder. The file is called 'run-jdbc.sh'. If we look at the script we can see that it does a bit of setup at the top.

Presenter2: Our scripts all follow this basic format, but you will see this HARVESTER\_TASK variable at the top, that is unique for each of our scripts. After it loads all the data from the ENV file, we setup some variables.

Presenter2: These variables are here so that (theoretically) the script is more readable and prevents inconsistencies such as a value getting changed in one location, but not another.

Presenter2: The first step of our process is to clone that database instance into a local h2 database that the harvester can query. While not necessary in this case, we took the opportunity to demonstrate the [\\$DatabaseClone](#) tool

Presenter2: In production situations, we have found it better to setup a clone of the data we want to query so that we can run queries against it without putting any load on production systems.

## Clone data

Presenter2: We see the tool [\\$DatabaseClone](#) is called, loading its parameters from the configuration file named example.databaseclone.xml, but we are explicitly setting the parameter named --outputConnection to be that of the [\\$CLONEDBURL](#)

Presenter2: You will note that the path to the configuration file is relative to the base Harvester/ folder. This works because in that block of setup stuff, we did make sure that the script is looking in the base Harvester/ folder.

Presenter2: I always suggest executing the scripts from the base Harvester/ folder anyways, because sometimes (on certain systems) that block of setup doesn't work correctly, but it should in most cases.

Presenter2: So, let's look at the config/tasks/example.databaseclone.xml file and make it comply with the database we just made.

Presenter2: You may need super-user rights to make these changes.

Presenter2: We need to set the inputConnection to be correct for the database you just setup (if your mysql server is on the same box and you named your database DemoDB, there should be no need to change this)

Presenter2: The inputUsername should be changed to the username we created earlier (again, if you are using the name DemoDB, there should be no need to change this)

Presenter2: The inputPassword should also be changed to the password for the user we just created (I doubt you used the same crazy password I did, so you will probably need to change this)

Presenter2: Let's save this file.

Presenter2: Also, as a note, you can see that we actually commented out the parameter called outputConnection, since whatever value was put in this file would just be overridden by the explicitly entered connection in the script.

Presenter2: The rest of the output parameters are set and used, and you can see that it pushes the data into an h2 database. It's location is set from the [\\$CLONEDBDIR](#) variable

– Presenter2 closes the file and switches back to the script –

Presenter2: Ok, so we are ready to kick this script off! Let's get back to the commandline. ( I will be explaining the script after we see what it does! )

– Presenter2 closes the script file –

## Run Harvest

Presenter2: To run the script, while in the base Harvester/ folder, we simply type:

```
>> bash scripts/run-jdbc.sh
```

– Presenter2 starts script –

Presenter2: We can watch the full log messages by watching the log file (shown at the very first line of output) with the tail command in another window

```
>> tail -f -n999999 logs
```

```
/<
```

```
logfile>.log
```

– Presenter2 waits for script to finish and restarts tomcat –

Presenter2: Restarting tomcat is not strictly necessary, but occasionally the reasoner is slow to add some statements, so I am doing this to force it to build the reasoned statements right now for the sake of time

– Presenter2 navigates to the vivo installation –

Presenter2: Here we can see that we now have fresh people ingested! Departments, linked to the University... Jobs... Seems everything went smoothly to get this data in here.

Presenter2: Now, I am going to go through the script and explain each step of the script to see what it did.

Presenter2: Any questions so far?

Presenter should attempt to answer questions, referring to the wiki pages as specifically as possible.

~~Presenter 3 takeover~~

## Publication Harvest

### [PubMed video webcast](#)

Presenter3: I'm going to show you a couple examples of the Harvester's ability to gather publication data from various sources. First I will demonstrate a harvest from an online source, in this case Pubmed. Next I will show you an example of a harvest from an existing file containing exported RefWorks data.

Presenter3: For Pubmed, what I want to demonstrate is the scoring and matching in action. What I'm going to do is create an account for a known genetics researcher at the University of Florida, Barry Byrne, and then I am going to fetch publications from the Pubmed database for which Mr. Byrne is an author. If all goes according to plan, what we will see in VIVO at the end of the harvest is that these publications have been automatically linked to Mr. Byrne's profile.

## Create example entry

Browse to VIVO, Admin. Create Barry Byrne, give bbyrne@ufl.edu email address.

Presenter3: Included with Harvester are several sample scripts to help get you started. The one we will base this harvest off of is run-pubmed.sh.

## Edit scripts

nano scripts/run-pubmed.sh.

Presenter3: Now we will need to make some changes to this script, so it's a good idea to save a copy.

Ctrl+O, pubmed-demo.sh, go to fetch line

Presenter3: The first thing we will need to mess with is the fetch configuration. In this case, it's pointing to a configuration file example.pubmedfetch.xml. Let's take a look at that.

## configure task

nano config/tasks/example.pubmedfetch.xml

Presenter3: There are four configuration options here. The first is an email address, which is used by Pubmed to notify you of any issues related to your query. Assuming the search goes off without a hitch, this is not really necessary, so we will leave this default address as is. The second is the search query. This is exactly what you would type into Pubmed's web search if you were searching for these publications on the website. To make sure we are getting publications for Barry Byrne at the University of Florida, and not much else, I'll put in an appropriate query.

change termSearch to ufl AND edu AND byrne

Presenter3: We'll ignore the third and fourth term for now. However, they are important if you are expecting large volumes of data, as they limit how many records get returned from the fetch. Let's return to the script.

Ctrl+O, enter, Ctrl+X.

nano scripts/pubmed-demo.sh

scroll to pubmed fetch

Presenter3: So this line is going to run to the web and give that query to Pubmed, which will return its records. They will go into the H2 data store specified by the last parameter. In the next set of lines this gets backed up.

scroll to XSLTranslator line

Presenter3: Pubmed returns data in its own format, which is different from VIVO's RDF-XML. This next line takes the raw data from Pubmed, and converts it into VIVO-compatible RDF-XML data, according to the XSLT translation file specified in the last parameter.

Presenter3: Now we have another backup block, and then we are transferring the RDF-XML data into a Jena model which allows our remaining tools to work with it.

scroll to Score

Presenter3: We have reached the part where the publications get matched to the profile, and this begins with Score. Here we will need to make another change, since the sample script uses workEmail but we want to compare with just the email field.

scroll up and change CWEMAIL and SWEMAIL to refer to email instead of workEmail

Presenter3: Score and Match work together to match the publications to the profile. Later the script is also looking for existing publications and journals, as we can see, and then it backs up all these results.

scroll to ChangeNamespace stuff

Presenter3: There's one final change we're going to make and then we are ready to run our fetch. Presently, in VIVO 1.2, for all the publications pulled in by the fetch, any co-authors of Barry Byrne would be listed in that publication as a missing author. If we would like instead to pull in these authors and make stubs for them with just their name, we can uncomment this ChangeNamespace line and comment the Qualify line.

## switch to create stubs

uncomment ChangeNamespace and comment Qualify

Presente3r: The rest of the script takes our data and puts it into VIVO

Ctrl+X

bash scripts/pubmed-demo.sh

when done, refresh Barry Byrne's profile, show publications, click on one

Presenter3: Now there are publications attached to Barry Byrne's profile. If I click on one, notice how it has its own profile and all the authors are linked to it. So there you have an example of a Pubmed harvest.

MODS (todo: bibutils download):

## Mods

[mods video webcast](#)

Presenter3: Now we're going to look at a harvest from an existing document. Most of it is similar, so I won't repeat too much, but there are some important things to mention. We are starting with a file from RefWorks in the BibTeX format. Now, it is certainly possible to write an XSLT translation from BibTeX to VIVO, but bundled with Harvester is a third-party tool called Bibutils that converts several different formats, including BibTeX, into an intermediate format called MODS. And also with Harvester is a sample XSLT translation file to convert a MODS file to VIVO. So let's take a look at run-mods.sh.

nano scripts/run-mods.sh

scroll to BIBUTILSBASE

Presenter3: The only thing we're going to look at is the beginning of this script, where instead of a fetch, we are calling our bundled tools to translate the file to MODS format. Take a quick look at BIBUTILSBASE. This is the base directory of Bibutils. By default, the tools for 32-bit systems are used, which should work on any system. If you wish to use the 64-bit tools, comment this line and uncomment the one above it. The next important variable is BIBUTILSINPUTFORMAT. This determines what format we are converting to MODS, and the valid values are defined in the documentation. For BibTeX, the default of "bib" is what we want.

Presenter3: The RunBibutils line performs the conversion on every file in the directory defined by BIBINDIR. So let's take a look at BIBINDIR, because that's where we will want to place our input file. It's rh-bibutils-in. It won't exist yet so we'll have to create it.

Ctrl+X

(get bibtexdemo.bib)

mkdir harvested-data/mods

mkdir harvested-data/mods/rh-bibutils-in

cp bibtexdemo.bib harvested-data/mods/rh-bibutils-in

Presenter3: So now our input file is in and ready to go. Let's take a quick look at it.

nano harvested-data/mods/rh-bibutils-in/bibtexdemo.bib

Presenter3: This contains a publication called "Accelerating Dynamic Iteration Methods with Application to Semiconductor Device Simulation". So to verify that our harvest worked, after we run the file we'll see if VIVO contains a publication by that name. So let's run the file.

Ctrl+X

bash scripts/run-mods.sh

Presenter3: And now we'll look for that publication in VIVO.

Go to Index, find publication

Presenter3: That concludes my presentation on Pubmed and MODS fetch.