Division of Sponsored Research

The Division of Sponsored Research (DSR) was established in 1962 by an act of the Florida Legislature to manage and stimulate an expanding and balanced research program. DSR facilitates institutional approval for all extramural proposal submissions, accepts and administers grant awards, and negotiates contracts and other research-related agreements on behalf of the University of Florida.

Data Source

The DSR as a data source provides a database view which is then harvested using the JDBCFetch tool.
The mapping of the provided data to VIVO RDF is specified in an XSLT.

XSLT

The example XSLT is provided here

Script

Provided with the code is an example script to harvest from DSR.

The script makes use of

- vivo.model.xml

Header

Contains the license information, authors etc.

```
#!/bash

#Copyright (c) 2010-2011 VIVO Harvester Team. For full list of contributors, please see the AUTHORS file
provided.
#All rights reserved.
#This program and the accompanying materials are made available under the terms of the new BSD license which
accompanies this distribution, and is available at
#http://www.opensource.org/licenses/bsd-license.html
#
# Contributors:
#     Christopher Haines, Dale Scheppler, Nicholas Skaggs, Stephen V. Williams, James Pence- initial API and
implementation
```

Setup

```
# set to the directory where the harvester was installed or unpacked
# HARVESTER_INSTALL_DIR is set to the location of the installed harvester
#        If the deb file was used to install the harvester then the
#        directory should be set to /usr/share/vivo/harvester which is the
#        current location associated with the deb installation.
#        Since it is also possible the harvester was installed by
#        uncompressing the tar.gz the setting is available to be changed
#        and should agree with the installation location
HARVESTER_INSTALL_DIR=/usr/share/vivo/harvester
export HARVEST_NAME=example-dsr
export DATE=`date +%Y-%m-%d'T'%T`

# Add harvester binaries to path for execution
# The tools within this script refer to binaries supplied within the harvester
#        Since they can be located in another directory their path should be
#        included within the classpath and the path environmental variables.
export PATH=$PATH:$HARVESTER_INSTALL_DIR/bin
export CLASSPATH=$CLASSPATH:$HARVESTER_INSTALL_DIR/bin/harvester.jar:$HARVESTER_INSTALL_DIR/bin/dependency/*
export CLASSPATH=$CLASSPATH:$HARVESTER_INSTALL_DIR/build/harvester.jar:$HARVESTER_INSTALL_DIR/build/dependency/*

# Exit on first error
# The -e flag prevents the script from continuing even though a tool fails.
#        Continuing after a tool failure is undesirable since the harvested
#        data could be rendered corrupted and incompatible.
set -e

# Supply the location of the detailed log file which is generated during the script.
#        If there is an issue with a harvest, this file proves invaluable in finding
#        a solution to the problem. It has become common practice in addressing a problem
#        to request this file. The passwords and usernames are filtered out of this file
#        To prevent these logs from containing sensitive information.
echo "Full Logging in $HARVEST_NAME.$DATE.log"
if [ ! -d logs ]; then
  mkdir logs
fi
cd logs
touch $HARVEST_NAME.$DATE.log
ln -sf $HARVEST_NAME.$DATE.log $HARVEST_NAME.latest.log
cd ..

#clear old data
# For a fresh harvest, the removal of the previous information maintains data integrity.
#        If you are continuing a partial run or wish to use the old and already retrieved
#        data, you will want to comment out this line since it could prevent you from having
#         the required harvest data.
#rm -rf data
#if [ -d data]; then
#mv -f data data.$DATE
#fi
```

Fetch

The information is pulled into the system. Since it is a standard database JDBCFetch is used.

```
# Execute Fetch
# This stage of the script is where the information is gathered together into one local
#        place to facilitate the further steps of the harvest. The data is stored locally
#        in a format based off of the source. The format is a form of RDF yet its ontology
#        too simple to be put into a model and be useful.
# The JDBCFetch tool in particular takes the data from the chosen source described in its
#        configuration XML file and places it into record set in the flat RDF directly
#        related to the rows, columns and tables described in the target database.
harvester-jdbcfetch -X jdbcfetch.config.xml
```

Translation

The XSLT is applied across the data from the fetch.

```
# Execute Translate
# This is the part of the script where the outside data, in its flat RDF form is used to
#        create the more linked and descriptive form related to the ontological constructs.
#        The traditional XSL language is used to achieve this part of the work-flow.
harvester-xsltranslator -X xsltranslator.config.xml
```

Transfer

The translated data is placed into an RDF database model.

```
# Execute Transfer to import from record handler into local temp model
# From this stage on the script places the data into a Jena model. A model is a
#        data storage structure similar to a database, but is in RDF.
# The harvester tool Transfer is used to move/add/remove/dump data in models.
# For this call on the transfer tool:
# -s refers to the source translated records file, which was just produced by the translator step
# -o refers to the destination model for harvested data
# -d means that this call will also produce a text dump file in the specified location
harvester-transfer -s translated-records.config.xml -o harvested-data.model.xml -d data/harvested-data/imported-
records.rdf.xml
```

Scoring and Matching

The various name spaces determined during the translation are scored against specific data in the vivo model.

The scoring process results in a model which contains information about the score results to be used in the matches.

The matching process changes the URI of the matched data to the URI's present in VIVO.

```
# Execute Score for Grants
# In the scoring phase the data in the harvest is compared to the data within Vivo and a new model
#         is created with the values / scores of the data comparisons.
harvester-score -X score-grants.config.xml

# Execute Score for Sponsor organizations.
# In the scoring phase the data in the harvest is compared to the data within Vivo and a new model
#         is created with the values / scores of the data comparisons.
harvester-score -X score-sponsor.config.xml

# Find matches using scores and rename nodes to matching uri
# Using the data model created by the score phase, the match process changes the harvested uris for
#         comparison values above the chosen threshold within the xml configuration file.
harvester-match -X match-grants.config.xml

# Execute Score for People
# In the scoring phase the data in the harvest is compared to the data within Vivo and a new model
#         is created with the values / scores of the data comparisons.
harvester-score -X score-people.config.xml

# Execute Score for Departments
# In the scoring phase the data in the harvest is compared to the data within Vivo and a new model
#         is created with the values / scores of the data comparisons.
harvester-score -X score-dept.config.xml

# Execute Score for Primary investigator roles
# In the scoring phase the data in the harvest is compared to the data within Vivo and a new model
#         is created with the values / scores of the data comparisons.
harvester-score -X score-pirole.config.xml

# Execute Score for Co-primary investigator roles
# In the scoring phase the data in the harvest is compared to the data within Vivo and a new model
#         is created with the values / scores of the data comparisons.
harvester-score -X score-copirole.config.xml

# Find matches using scores and rename nodes to matching uri
# Using the data model created by the score phase, the match process changes the harvested uris for
#         comparison values above the chosen threshold within the xml configuration file.
# This config differs from the previous match config, in that it removes types and literals from the
#       resources in the incoming model for those that are considered a match.
harvester-match -X match-roles.config.xml
```

Smush

The smush process is used to remove duplicates from the harvested data.

```
# Smush to remove duplicates
# Using a particular predicate as an identifying data element the smush tool will rename those
#        resources which have matching values of that predicate to be one resource.
harvester-smush -X smush-grant.config.xml

harvester-smush -X smush-org.config.xml

harvester-smush -X smush-person.config.xml

harvester-smush -X smush-sponsor.config.xml
```

Changing Namespaces

For those parts which didn't find a match they are given URIs within the vivo's namespace.

```
# Execute ChangeNamespace to get unmatched grants into current name-space
# This is where the new people from the harvest are given uris within the name-space of Vivo
#         If there is an issue with uris being in another name-space, this is the phase
#        which should give some light to the problem.
harvester-changenamespace -X changenamespace-grant.config.xml

# Execute ChangeNamespace to get unmatched organizations into current name-space
# This is where the new people from the harvest are given uris within the name-space of Vivo
#         If there is an issue with uris being in another name-space, this is the phase
#        which should give some light to the problem.
harvester-changenamespace -X changenamespace-org.config.xml

# Execute ChangeNamespace to get unmatched sponsoring organizations into current name-space
# This is where the new people from the harvest are given uris within the name-space of Vivo
#         If there is an issue with uris being in another name-space, this is the phase
#        which should give some light to the problem.
harvester-changenamespace -X changenamespace-sponsor.config.xml

# Execute ChangeNamespace to get unmatched People into current name-space
# This is where the new people from the harvest are given uris within the name-space of Vivo
#         If there is an issue with uris being in another name-space, this is the phase
#        which should give some light to the problem.
harvester-changenamespace -X changenamespace-people.config.xml

# Execute ChangeNamespace to get unmatched Primary investigator roles into current name-space
# This is where the new people from the harvest are given uris within the name-space of Vivo
#         If there is an issue with uris being in another name-space, this is the phase
#        which should give some light to the problem.
harvester-changenamespace -X changenamespace-pirole.config.xml

# Execute ChangeNamespace to get unmatched Co-primary investigator roles into current name-space
# This is where the new people from the harvest are given uris within the name-space of Vivo
#         If there is an issue with uris being in another name-space, this is the phase
#        which should give some light to the problem.
harvester-changenamespace -X changenamespace-copirole.config.xml

# Execute ChangeNamespace to get unmatched time intervals into current name-space
# This is where the new people from the harvest are given uris within the name-space of Vivo
#         If there is an issue with uris being in another name-space, this is the phase
#        which should give some light to the problem.
harvester-changenamespace -X changenamespace-timeinterval.config.xml
```

Updating

The Subtraction and Additions are found while comparing to the previous harvest model.

*Note: The previous model should be equivalent to the actual data in VIVO. *

If the previously harvested data is edited, then that edit should also be applied to the previous model

```
# Find Subtractions
# When making the previous harvest model agree with the current harvest, the entries that exist in
#        the previous harvest but not in the current harvest need to be identified for removal.
harvester-diff -X diff-subtractions.config.xml

# Find Additions
# When making the previous harvest model agree with the current harvest, the entries that exist in
#        the current harvest but not in the previous harvest need to be identified for addition.
harvester-diff -X diff-additions.config.xml
```

Applying updates

The updates are applied to the previous model and then to VIVO. This should cause the previous model to be equal to the actual data harvested. The VIVO should also now have the data which is reliant on the harvest changed to be equal to the new harvest's data.

```
# Apply Subtractions to Previous model
harvester-transfer -o previous-harvest.model.xml -r data/vivo-subtractions.rdf.xml -m
# Apply Additions to Previous model
harvester-transfer -o previous-harvest.model.xml -r data/vivo-additions.rdf.xml

# Now that the changes have been applied to the previous harvest and the harvested data in vivo
#        should agree with the previous harvest, the changes are now applied to the vivo model.
# Apply Subtractions to VIVO for pre-1.2 versions
harvester-transfer -o vivo.model.xml -r data/vivo-subtractions.rdf.xml -m
# Apply Additions to VIVO for pre-1.2 versions
harvester-transfer -o vivo.model.xml -r data/vivo-additions.rdf.xml
```