HPCTARDIS Documentation

Release 1.0

Ian Thomas, Venki Balasubramanian

March 18, 2014

MyTARDIS is a multi-institutional collaborative venture that facilitates the archiving and sharing of data and metadata collected at major facilities such as the Australian Synchrotron and ANSTO and within Institutions.

HPCTardis is a extension of the MyTARDIS system for the task of transferring data and metadata from remote high-performance computing facilities, automatic extraction of package-specific metadata, curation of experiments for researcher access, and publishing of metadata to ANDS.

CHAPTER 1

System Administration's Manual

This document describes how to install and perform maintenance of the HPCTardis system.

1.1 Installation of HPCTardis

1.1.1 Prerequisites

Redhat:

```
sudo yum install cyrus-sasl-ldap cyrus-sasl-devel openldap-devel libxslt libxslt-devel libxslt-pytho
```

Debian/Ubuntu:

```
sudo apt-get install libsasl2-dev libldap libldap2-dev libxslt1.1 libxslt1-dev python-libxslt1
```

1.1.2 Download

To get the current trunk:

```
git clone https://code.google.com/p/hpctardis/
cd hpctardis
```

1.1.3 Building

HPCTARDIS is using the buildout buildsystem to handle dependencies and create the python class path:

```
python bootstrap.py
./bin/buildout
```

1.1.4 Testing

Run testcases to verify success:

```
bin/django test
```

1.1.5 Running Development Server

Copy prototypical settings file for local version:

```
cp tardis/settings_hpctardis.py tardis/settings.py
```

If required, modify standard Django `tardis/settings.py` file to change database etc. Documentation in `settings_hpctardis.py`

To configure HPCTardis for interactive use, modify the file `bin/django` and replace:

```
djangorecipe.manage.main('tardis.test_settings')
```

with:

```
djangorecipe.manage.main('tardis.settings')
```

This means that the `bin/django` command will run the interactive configuration rather than the test configuration.

Setup database and initial data:

```
bin/django syncdb --migrate --noinput
```

Create admin user:

```
bin/django createsuperuser
```

Start the development server:

```
bin/django runserver
```

System should now be running at http://127.0.0.1:8000

1.2 Common Admin and Maintenance Tasks

1.2.1 Admin Tool

Admin user can access the django administration tool to do routine database maintenance:

http://127.0.0.1/admin

1.2.2 Management Commands

Create a new HPCTardis user:

```
bin/django createuser
```

1.3 Installation of Facility Scripts

ssh-keygen -t rsa -f ~/.ssh/id_hpc

Do not enter any passphrase

```
[vxb595@vayu3 ~/dcscript]$ ssh-keygen -t rsa -f ~/.ssh/id_hpc
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/595/vxb595/.ssh/id_hpc.
Your public key has been saved in /home/595/vxb595/.ssh/id_hpc.pub.
The key fingerprint is:
79:45:17:09:36:41:73:53:7b:54:4c:49:88:f3:bd:9f vxb595@vayu3
```

Check for the creation of the keys – private and public keys

```
[vxb595@vayu3 ~/dcscript]$ ls -l ~/.ssh/
total 28
-rw------- 1 vxb595 h72 1675 Feb  9 15:40 id_hpc
-rw-r----- 1 vxb595 h72  394 Feb  9 15:40 id_hpc.pub
-rw------- 1 vxb595 h72 1675 Jan 26 16:09 id_rsa
-rw-r----- 1 vxb595 h72  394 Jan 26 16:09 id_rsa.pub
-rw-r----- 1 vxb595 h72  394 Nov 11 08:21 id_rsa.pub_venki
-rw------- 1 vxb595 h72 1675 Nov 11 08:21 id_rsa_venki
-rw-r--r-- 1 vxb595 h72  840 Dec  1 11:32 known_hosts
[vxb595@vayu3 ~/dcscript]$
```

Copy the public file to the tardis using the following command. The command requires the hpctardis password for transferring the file.

The following command copies the `id_hpc.pub` file from `~/.ssh/` directory to the home directory (~) in `gaia1.isis.rmit.edu.au`:

```
scp -P 8889 ~/.ssh/id_hpc.pub venki@gaia1.isis.rmit.edu.au: ~
```

Login inside the tardis using the following command; please note, use small p instead of capital P as in scp:

```
ssh -p 8889 venki@gaia1.isis.rmit.edu.au
```

The above command should ask for the hpctardis password. After successful login, list the files to check whether the transfer is successful.

```
venki@hpc:~$ ls -l
total 32
drwxrws--- 31 venki dcweb  4096 2012-02-09 14:21 dcweb
-rwxr-xr-x  1 venki venki 14259 2011-12-07 17:16 execsum.sh
-rwxr-xr-x  1 venki venki  3181 2011-12-07 18:48 fta.sh
-rw-r-----  1 venki venki   394 2012-02-09 15:42 id_hpc.pub
-rw-r-----  1 venki venki   401 2011-12-07 16:16 id_rsa.pub
venki@hpc:~$
```

check to see whether you have `~/.ssh/` in your tardis machine. If not create one, using the following command:

```
mkdir ~/.ssh
```

To create authorised keys for seamless login execute the following command:

```
cat id_hpc.pub >> ~/.ssh/authorized_keys2
```

```
total 32
drwxrws--- 31 venki dcweb  4096 2012-02-09 14:21 dcweb
-rwxr-xr-x  1 venki venki 14259 2011-12-07 17:16 execsum.sh
-rwxr-xr-x  1 venki venki  3181 2011-12-07 18:48 fta.sh
-rw-r-----  1 venki venki   394 2012-02-09 15:42 id_hpc.pub
-rw-r-----  1 venki venki   401 2011-12-07 16:16 id_rsa.pub
venki@hpc:~$ cat id_hpc.pub >> ~/.ssh/authorized_keys2
venki@hpc:~$
```

Type `exit` to logout of the hpctardis system, so that you can check whether the keys are properly placed in the appropriate directories using the following command from the hpc facilities:

```
ssh -p 8889 -i ~/.ssh/id_hpc venki@gaia1.isis.rmit.edu.au
```

If the above command does not prompt for password and lets you login inside hpctardis then the above configuration is successful.

In order to run the harvesting scripts from any directory, either place the scripts:

```
execsum.sh
fta.sh
```

in the `/usr/bin` directory or

Place following 2 commands inside the `~/.bashrc` file – these command assumes that both the harvesting scripts are inside the sub-directory `dcscript` in the user's home directory:

```
PATH=$HOME/dcscript:$PATH
export PATH
```

```
PATH=/home/595/vxb595/dcscript:$PATH
export PATH

".bashrc" 55L, 1511C
```

The home directory in the above figure is `($HOME=)/home/595/vxb595`

See also:

https://www.djangoproject.com/ The Django Project

https://docs.djangoproject.com/en/1.3/intro/install/ Django Quick Install Guide

CHAPTER 2

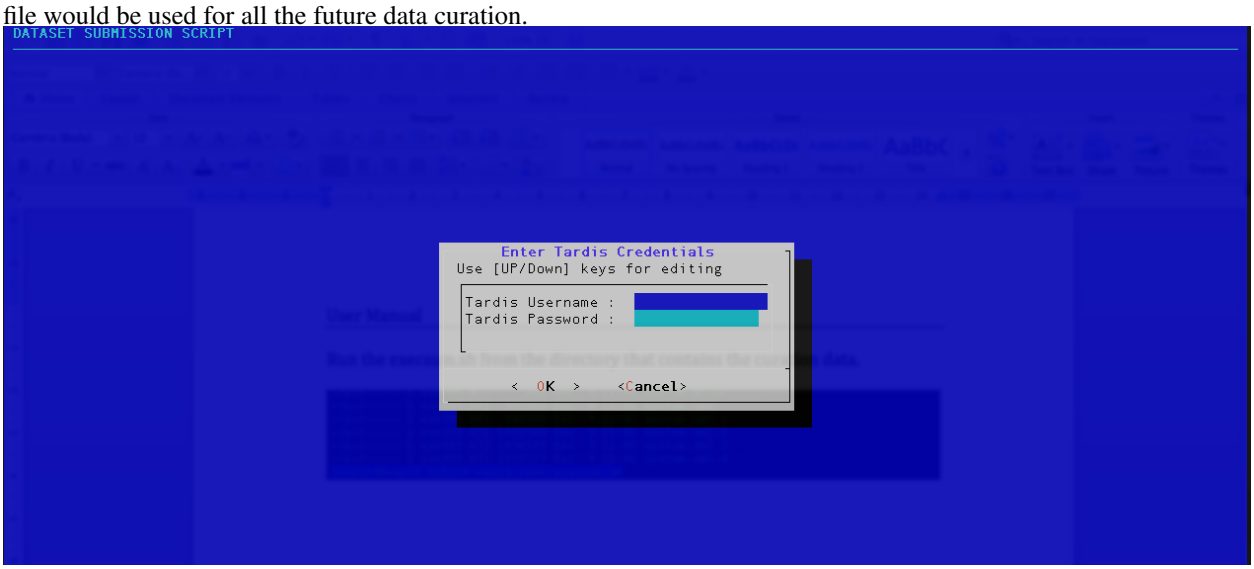
User Manual

This document shows an end to end use of the HPCtardis system from the script through to publishing at ANDS

Run the execsum.sh from the directory that contains the curation data.

```
-rw-r----- 1 vxb595 h72     268 Dec  9 11:01 system.alloc
-rw-r----- 1 vxb595 h72  957363 Dec  9 11:01 system.xml
-rw-r----- 1 vxb595 h72  948980 Dec  9 11:01 system.xml.1
-rw-r----- 1 vxb595 h72 1058544 Dec  9 11:01 system.xml.2
-rw-r----- 1 vxb595 h72 1058543 Dec  9 11:01 system.xml.3
-rw-r----- 1 vxb595 h72 1058527 Dec  9 11:01 system.xml.4
[vxb595@vayu3 SIESTA-new]$ bash execsum.sh
```

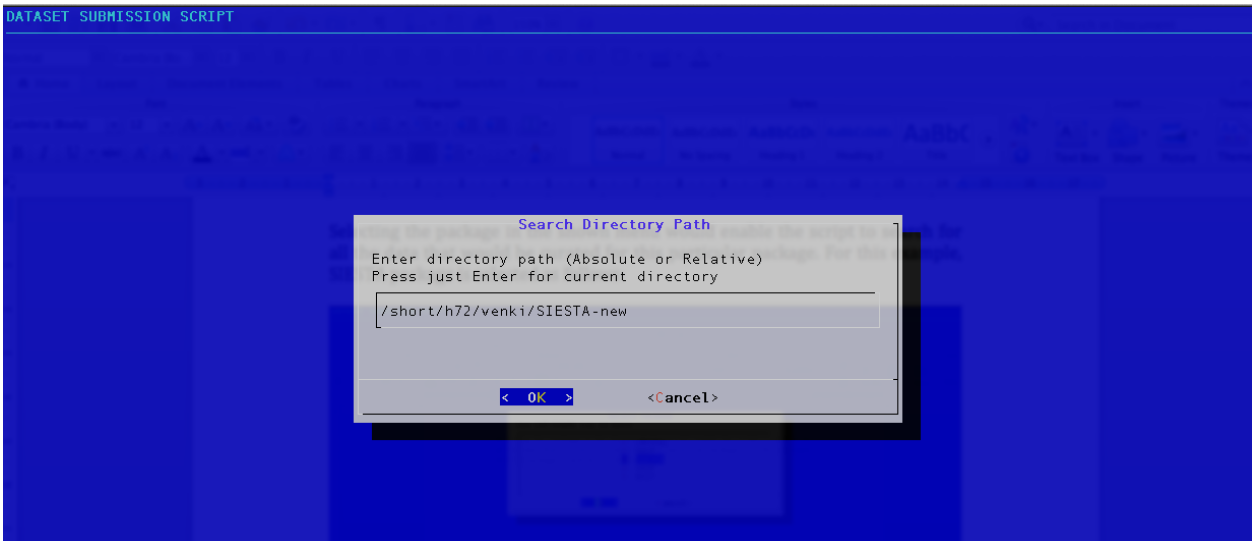
The script asks for hpctardis username/password for authentication to create the config file. Please note that this config file would be used for all the future data curation.



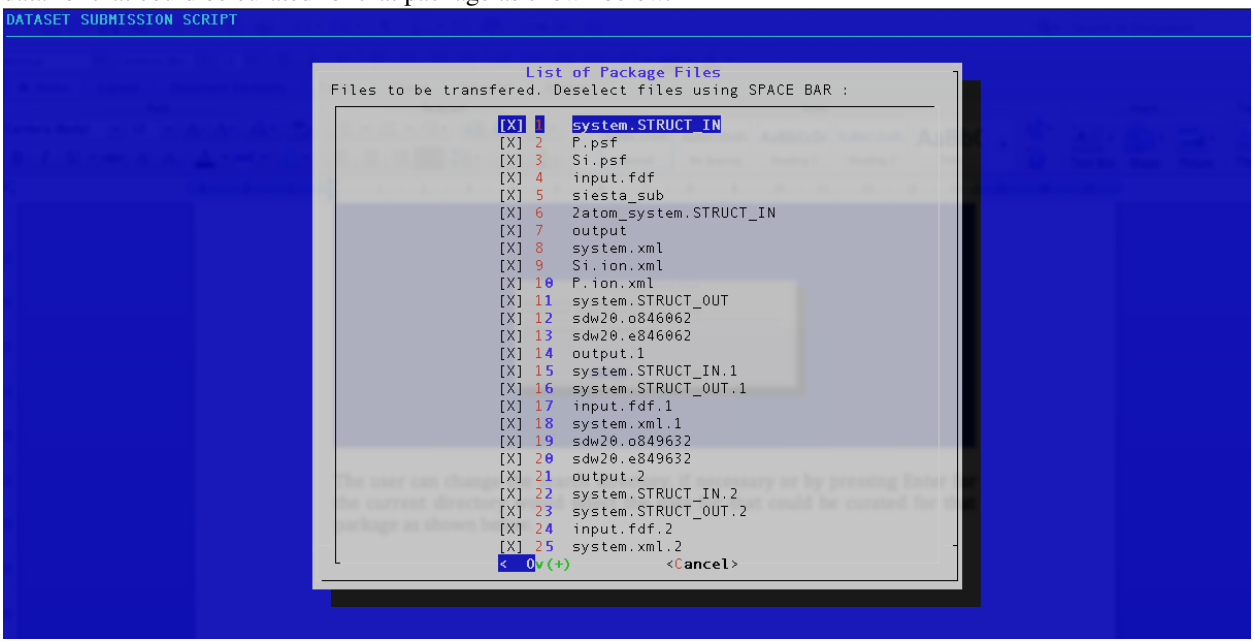
The hpctardis are validated as shown in the below message

Selecting the package in the shown menu would enable the script to search for all the data that would be curated for this particular package. For this example, SIESTA package is selected as follows:

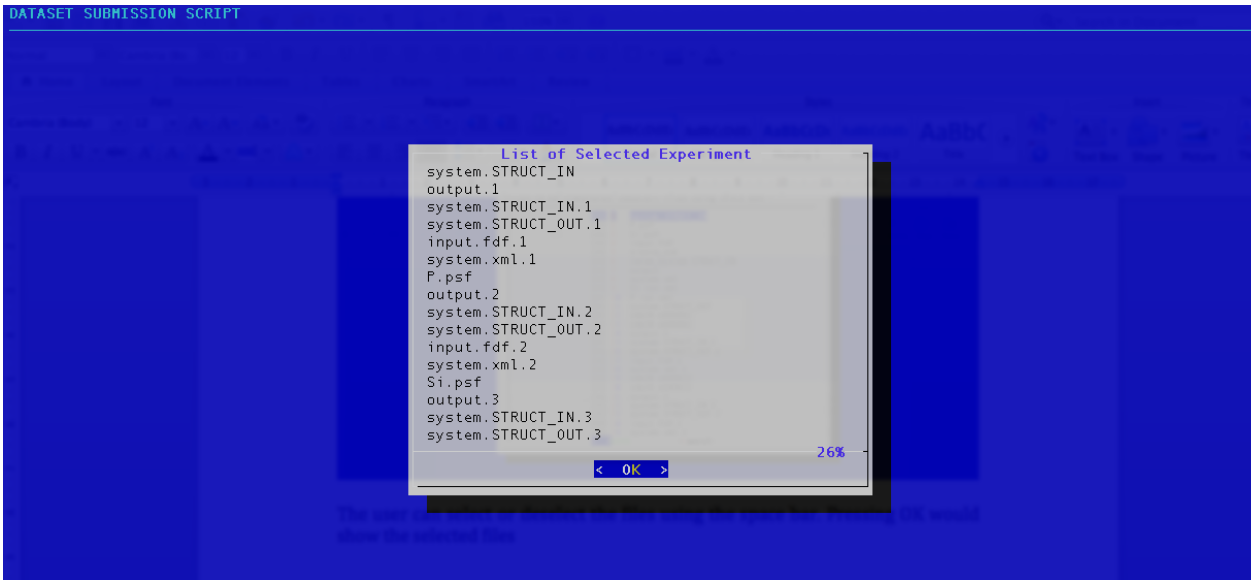
For user confirmation, the script shows the search directory for this package data as shown below:



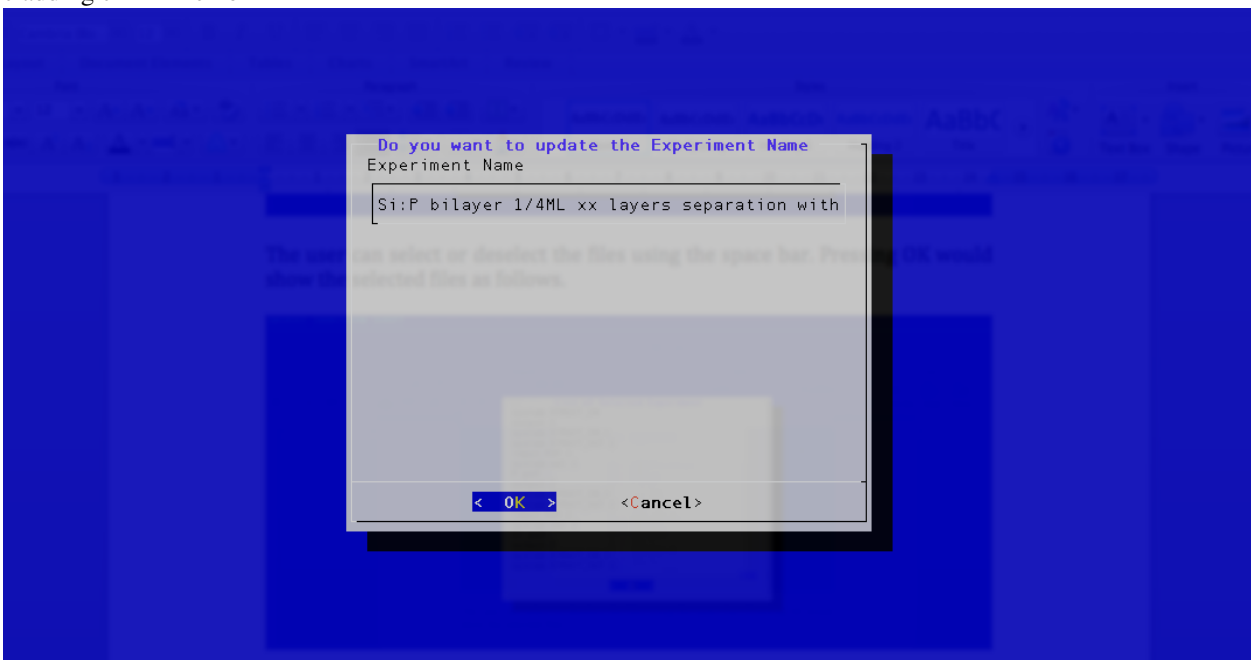
The user can change the search directory, if necessary or by pressing Enter for the current directory would select the data for that could be curated for that package as shown below.



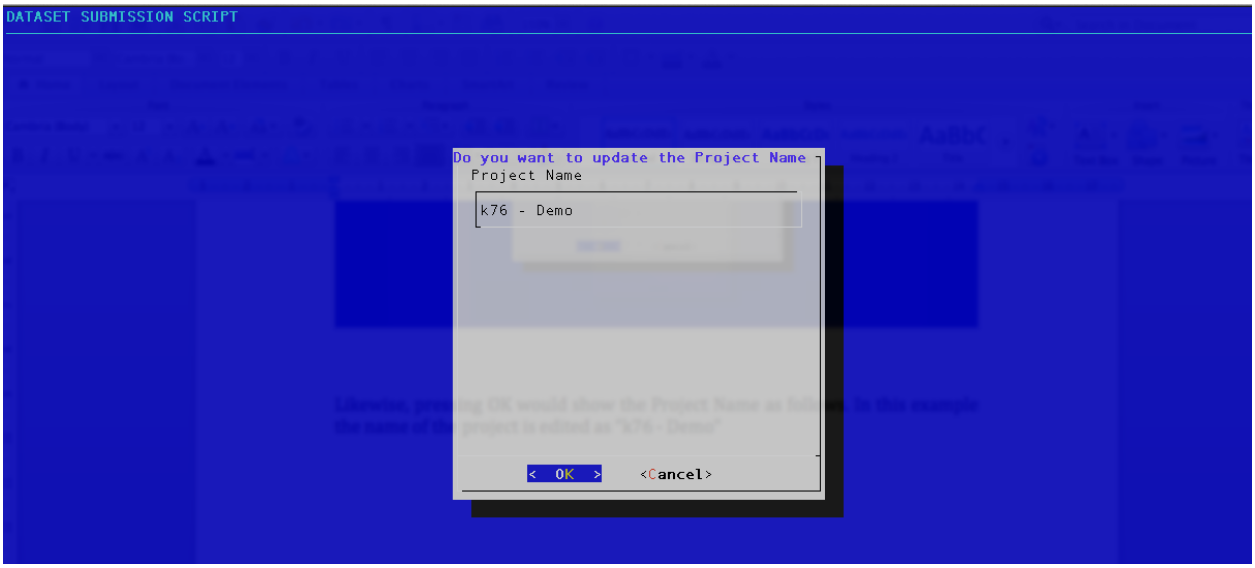
The user can select or deselect the files using the space bar. Pressing OK would show the selected files as follows.



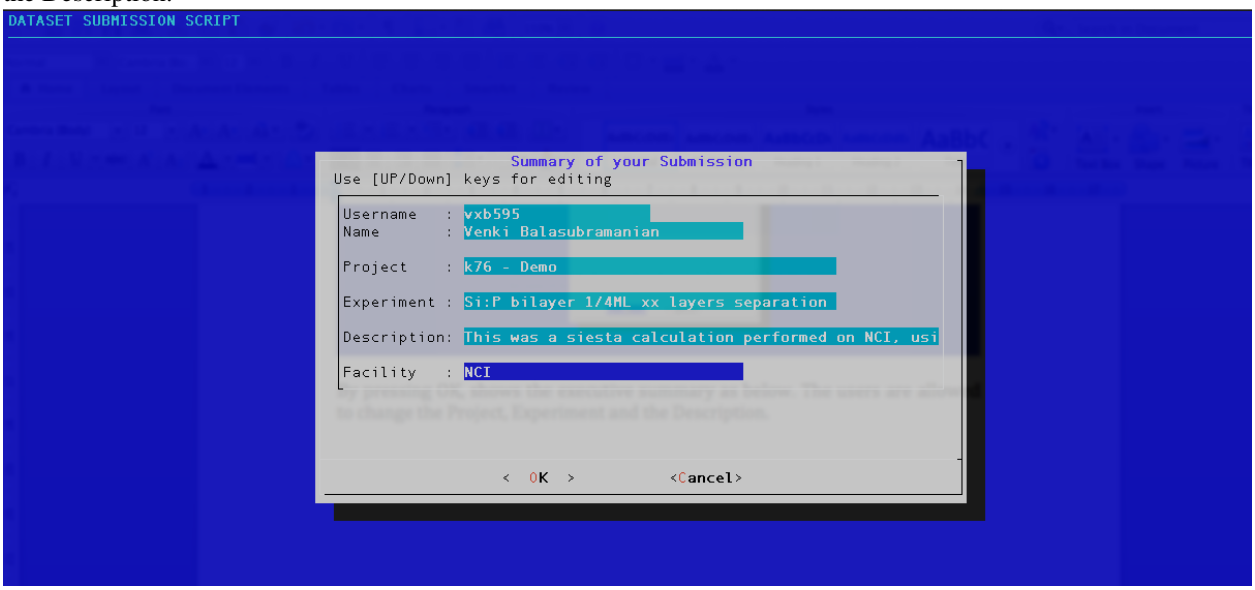
The scripts extract the experiment name from one of the selected file. If incorrect, the user is allowed to change the name. In this example the name of the experiment is edited as "Si:P bilayer 1/4ML xx layers separation with 80 layers cladding c2x2 - Demo"



Likewise, pressing OK would show the Project Name as follows. In this example the name of the project is edited as "k76 - Demo"

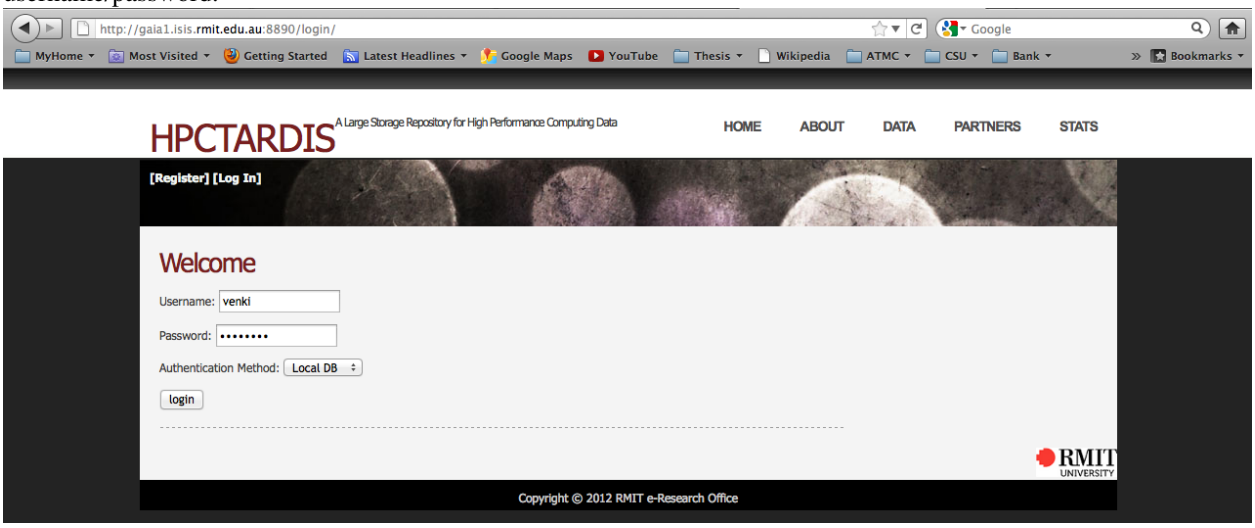


By pressing OK, shows the executive summary as below. The users are allowed to change the Project, Experiment and the Description.

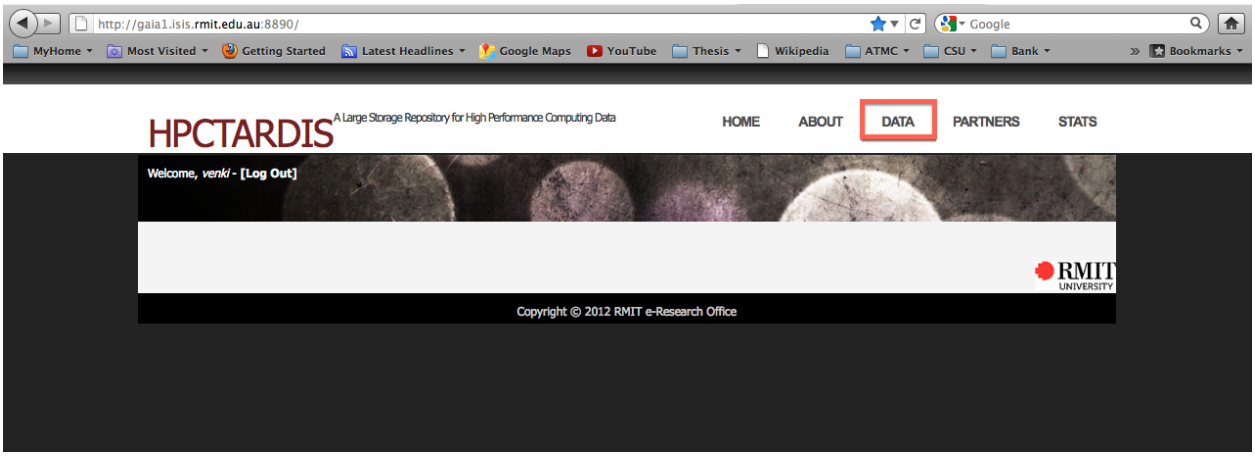


Once all the editing is done in this summary page. By pressing OK the script transfers all the files to the staging area.

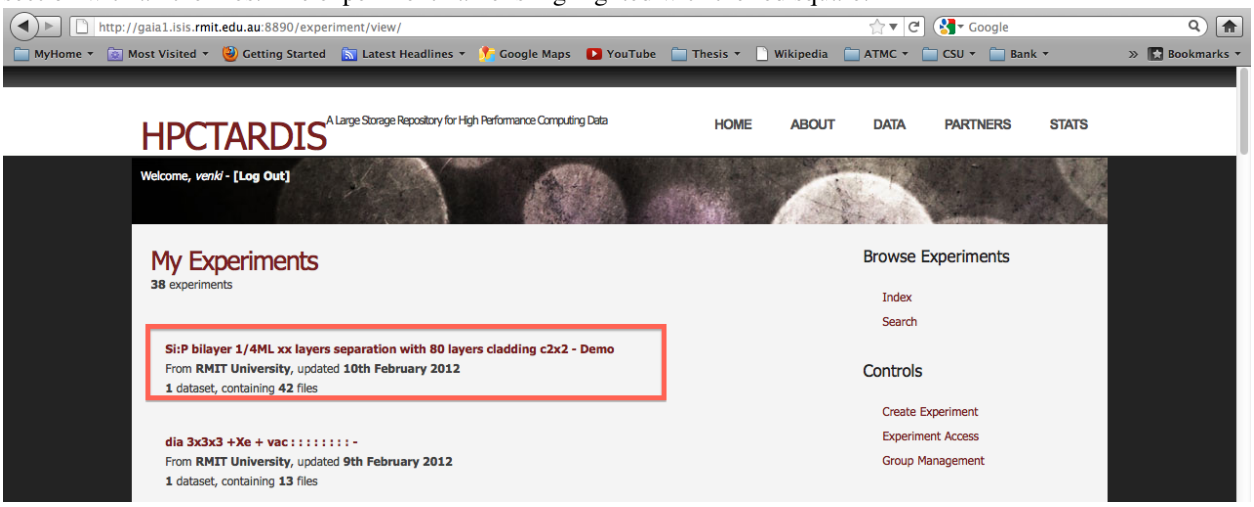
The fta.sh script automatically transfers the data/files from the staging area to the curation web portal hpctardis. The user can check by visiting the hpctardis webpage (http://gaia1.isis.rmit.edu.au:8890/login/) by entering the hpctardis username/password.



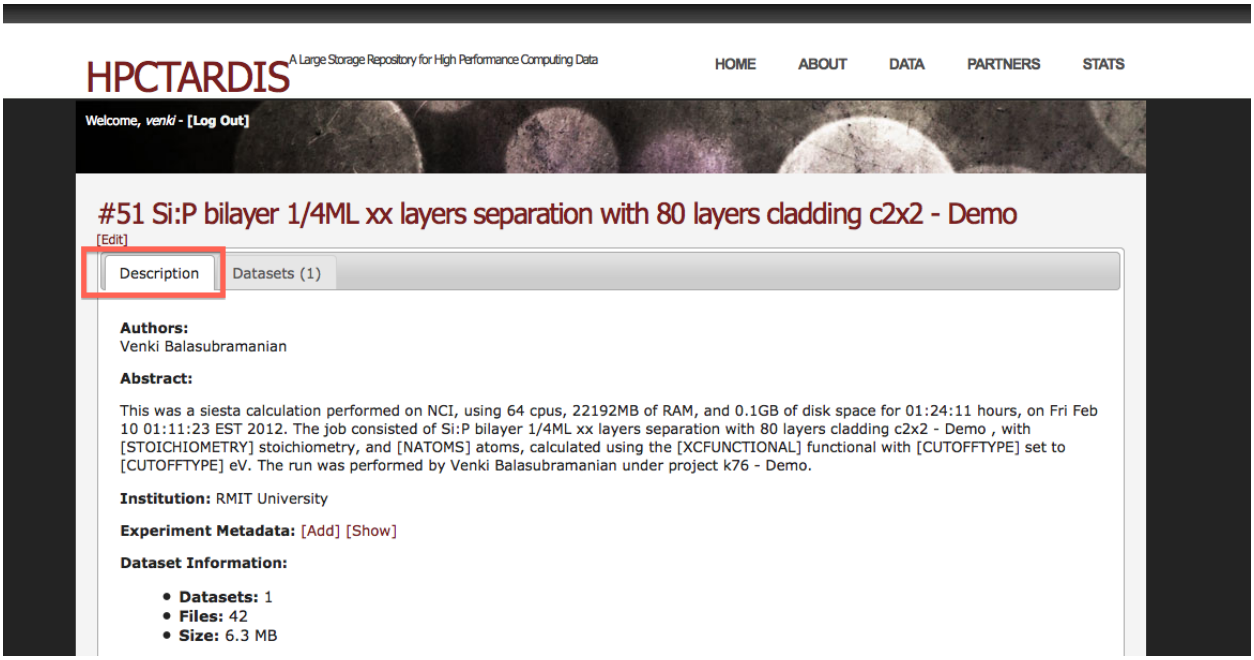
After successful login, click DATA link – marked with red square, to see the curate datasets.



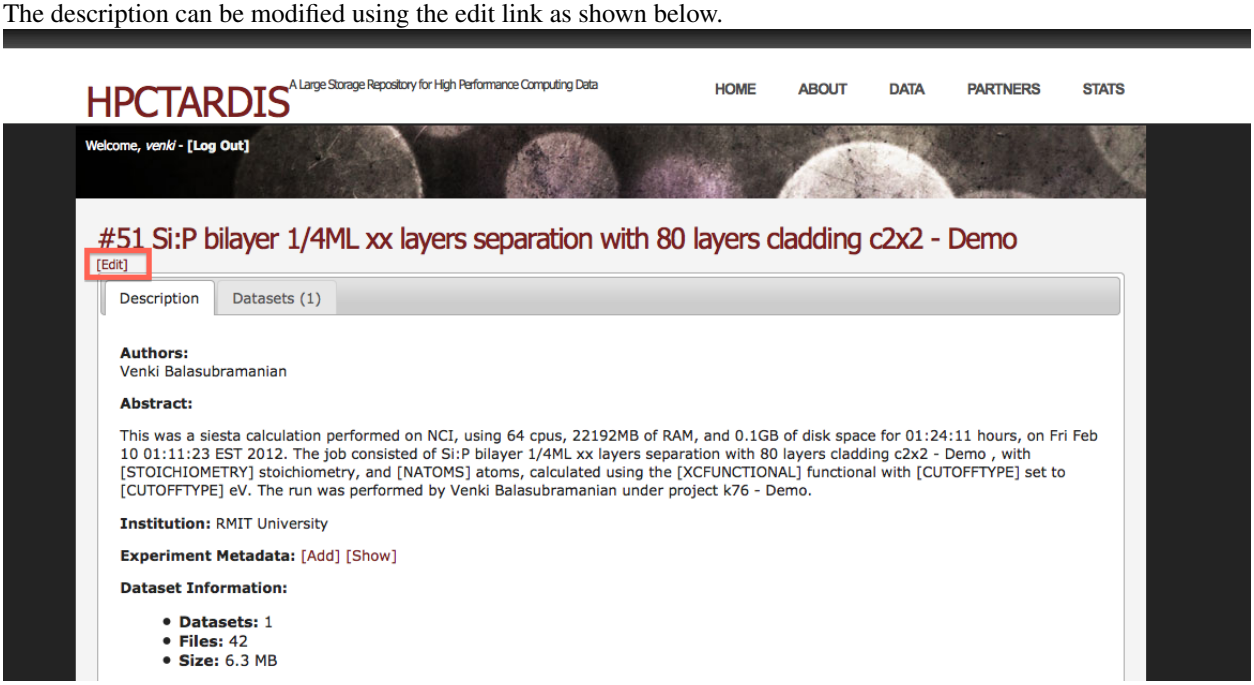
The list shows the curated data from the facility has created an experiment in the portal under the "My Experiment" section with all the files. The experiment name is highlighted with the red square.



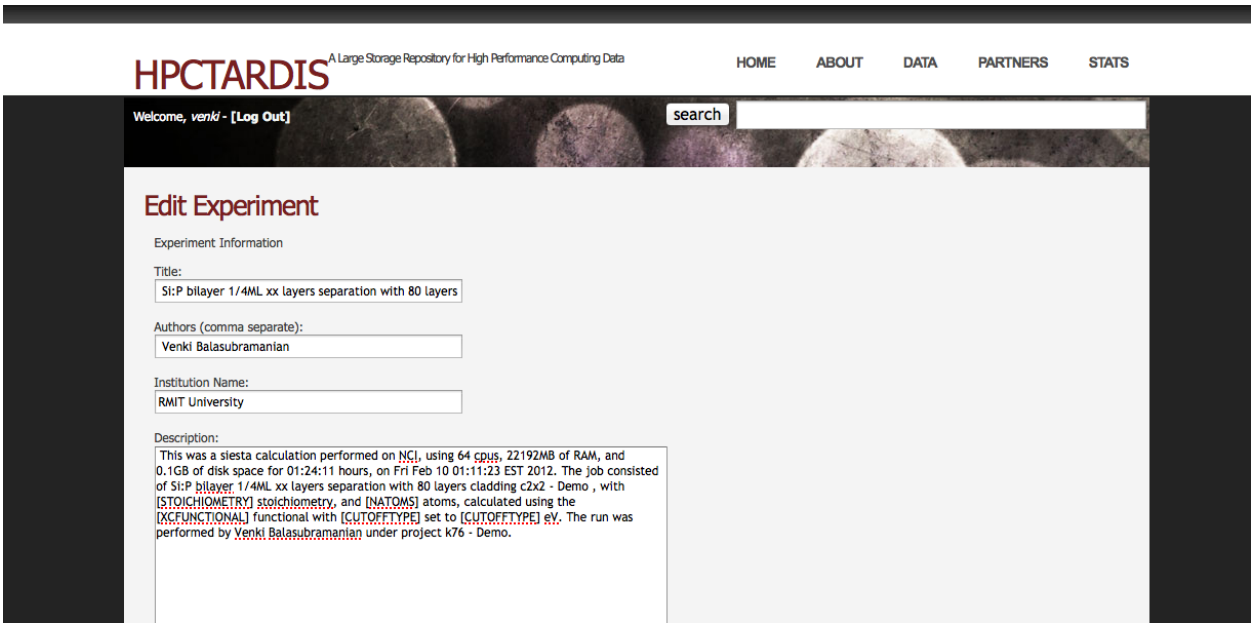
Clicking the experiment name shows the description of the experiment by default.



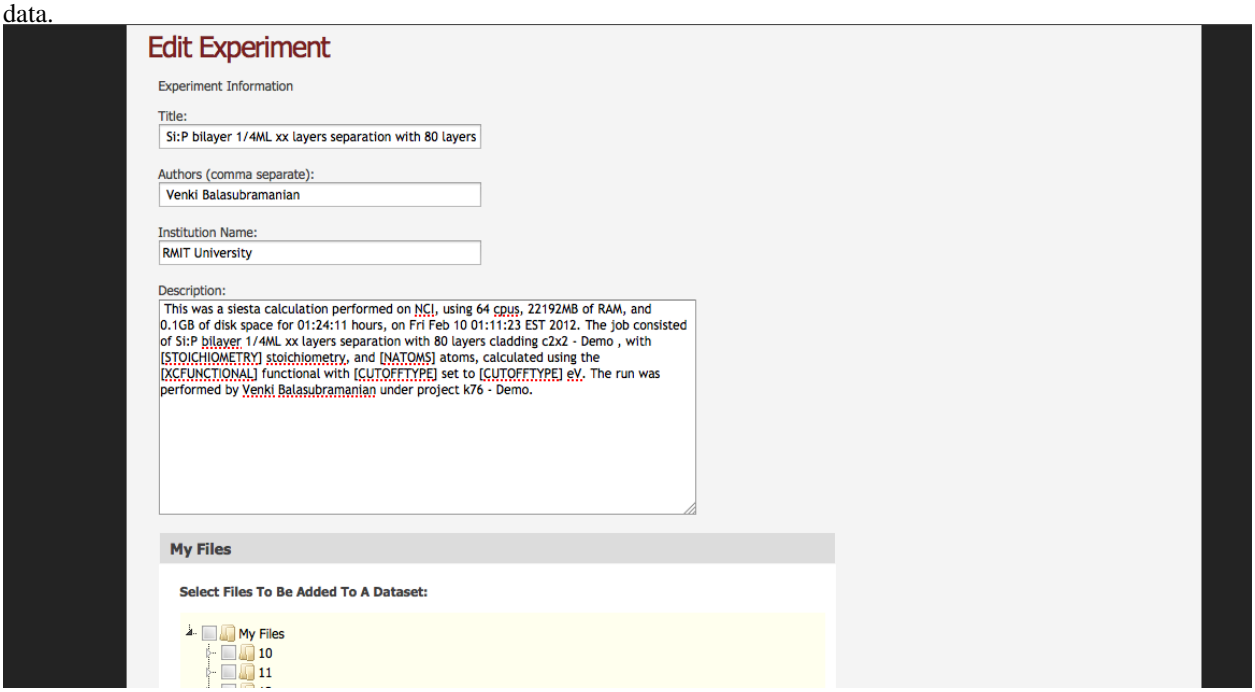
The description can be modified using the edit link as shown below.



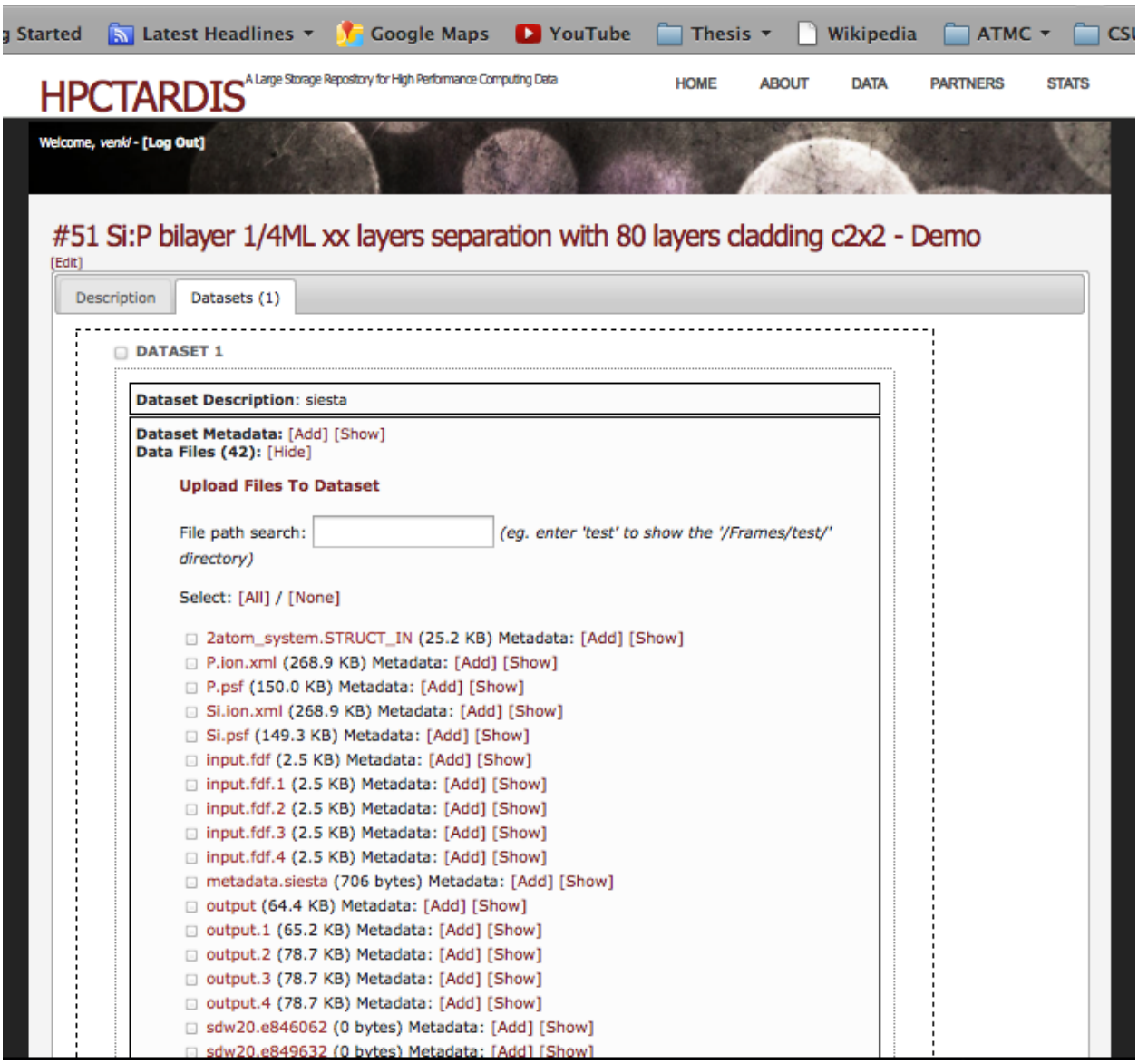
Upon clicking the edit link, lets the user to edit all the information about the experiment.



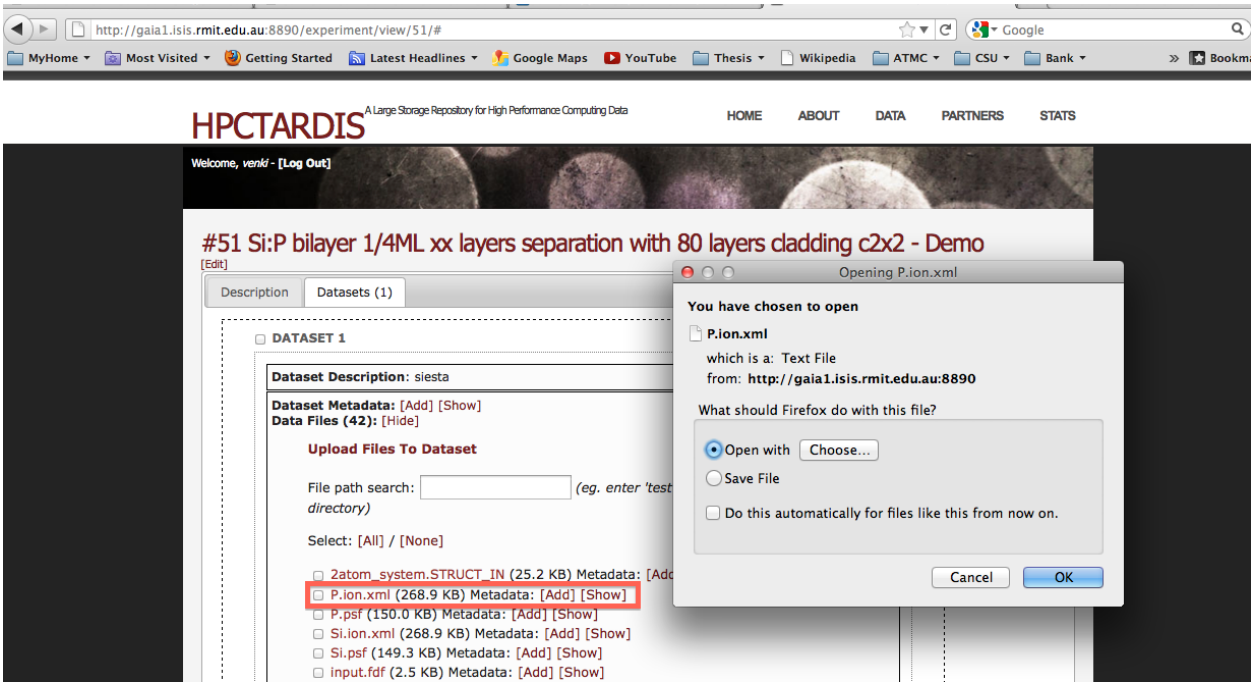
In the edit page, the user can edit the description, the name of the experiment, author name and even add or delete any data.



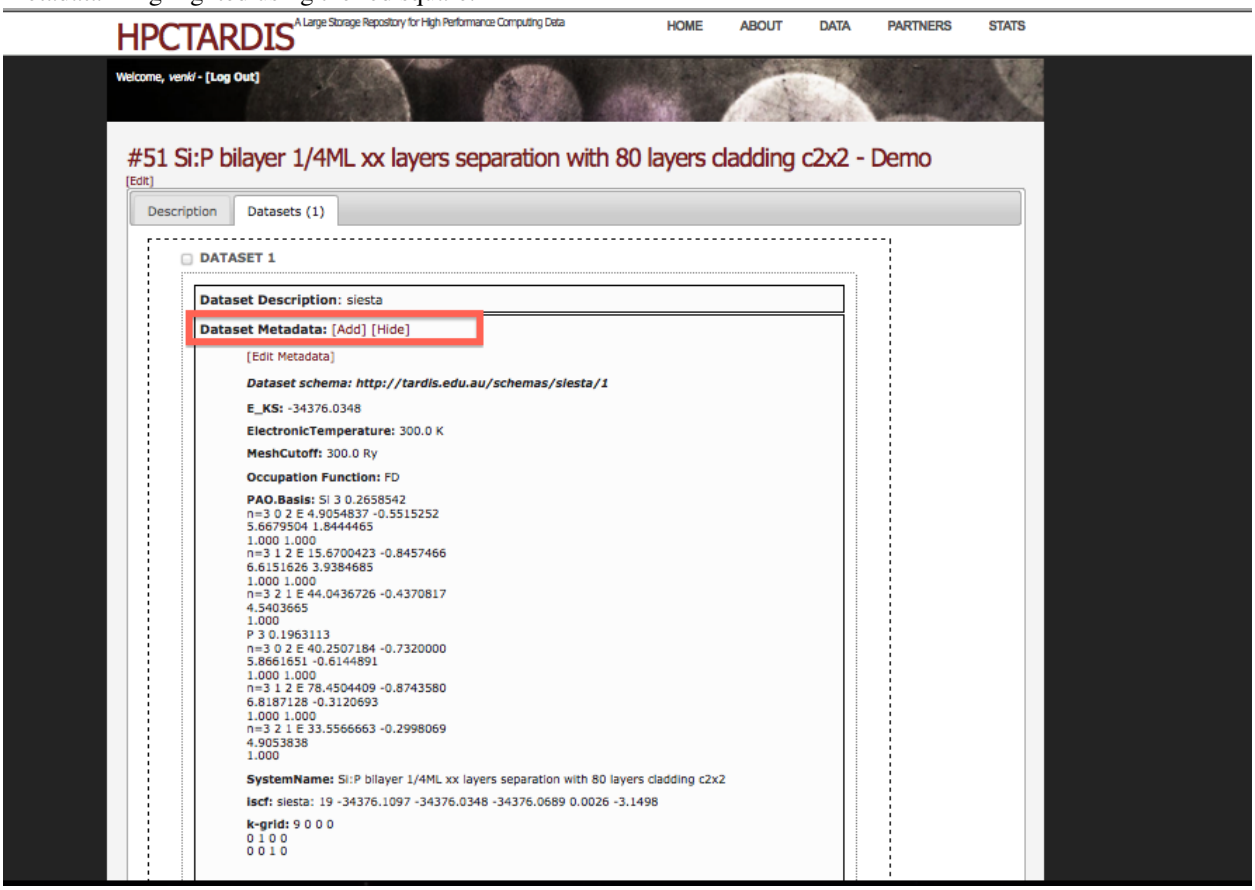
The dataset tab shows the list of data/files curated from the hpc facilities using the scripts.



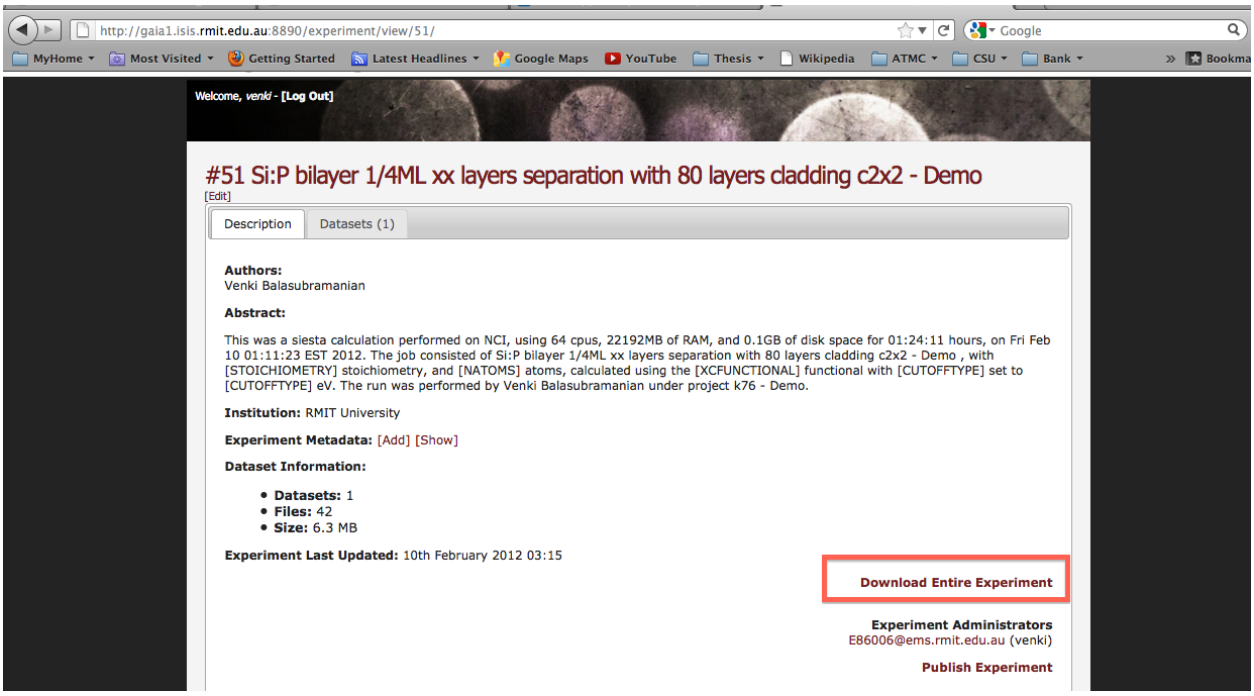
Any files can be downloaded and viewable by click at the each file. The below figure shows the P.ion.xml download.



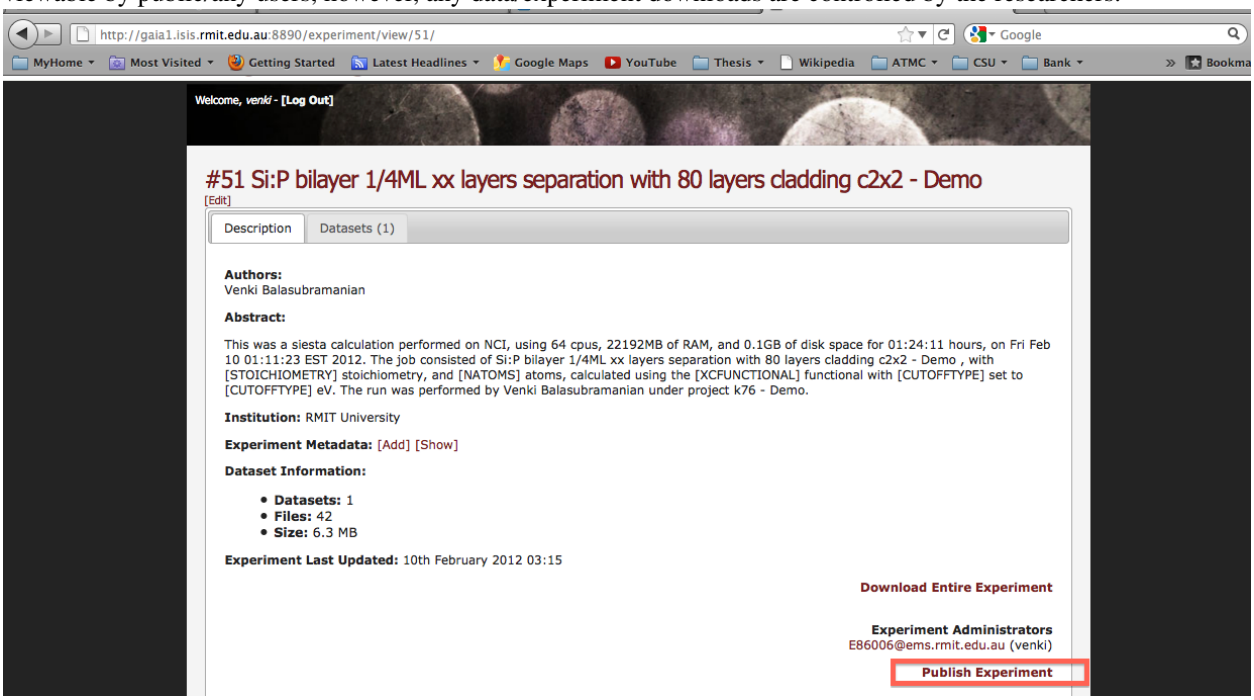
The Dataset tab also show the metadata extracted from the harvested files by clicking the show link in the Dataset Metadata – highlighted using the red square.



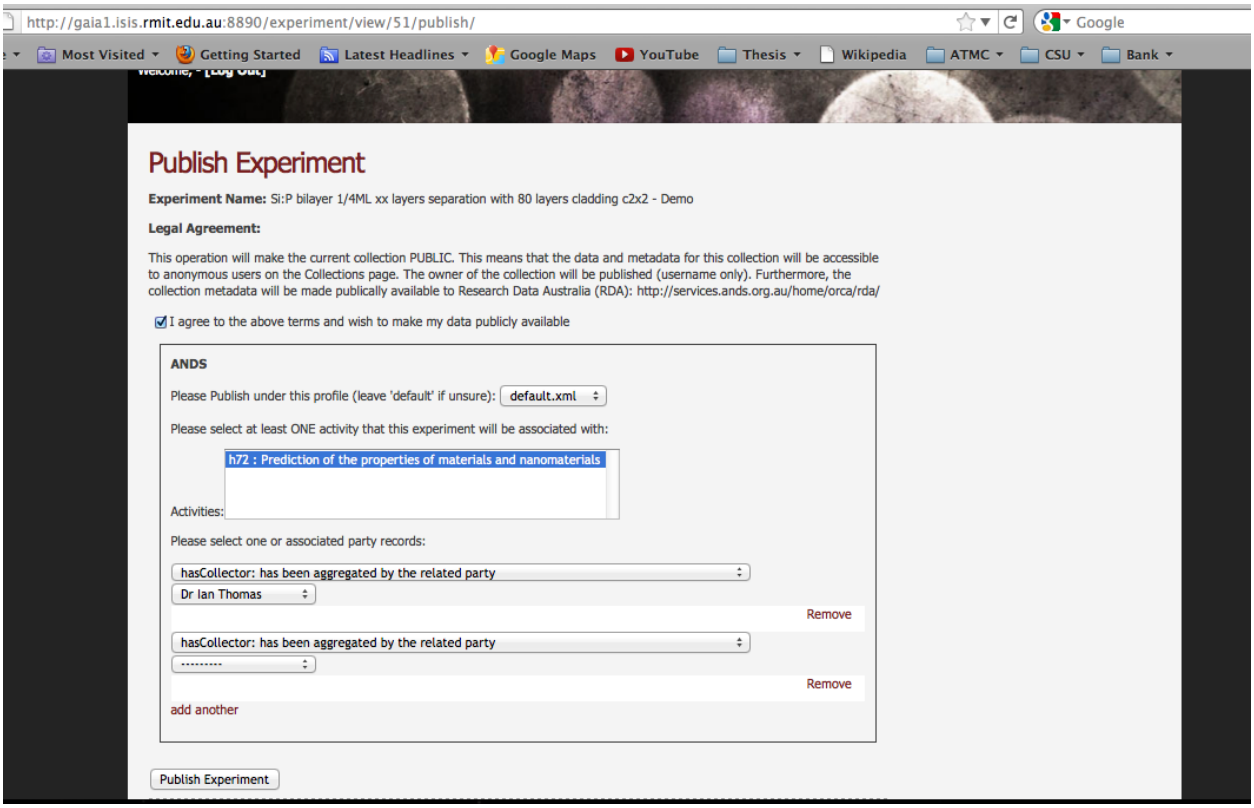
Clicking the "Download Entire Experiment" – as marked by the red square, can download the entire experiment.



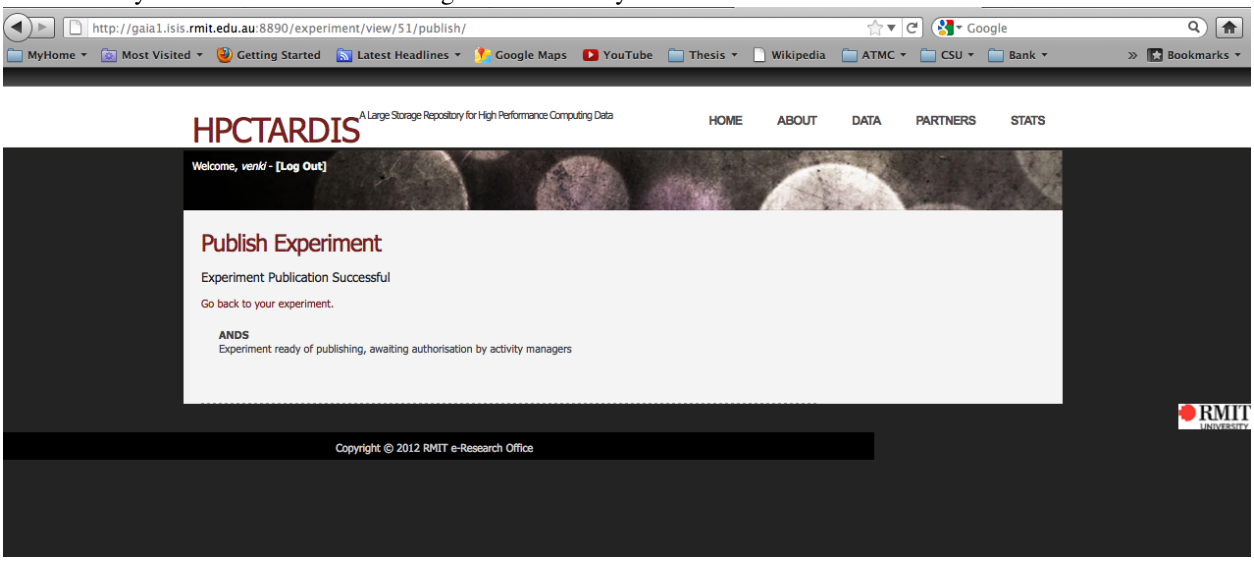
By default, the created experiment resides as user's private experiment under My Experiment section. It is viewable only by the user, however, in order to generate RIF-CS for metadata harvesting by ANDS, the experiment has to be made public by clicking the link Publish Experiment –highlighted with red square. Please note, public experiment are viewable by public/any users, however, any data/experiment downloads are controlled by the researchers.



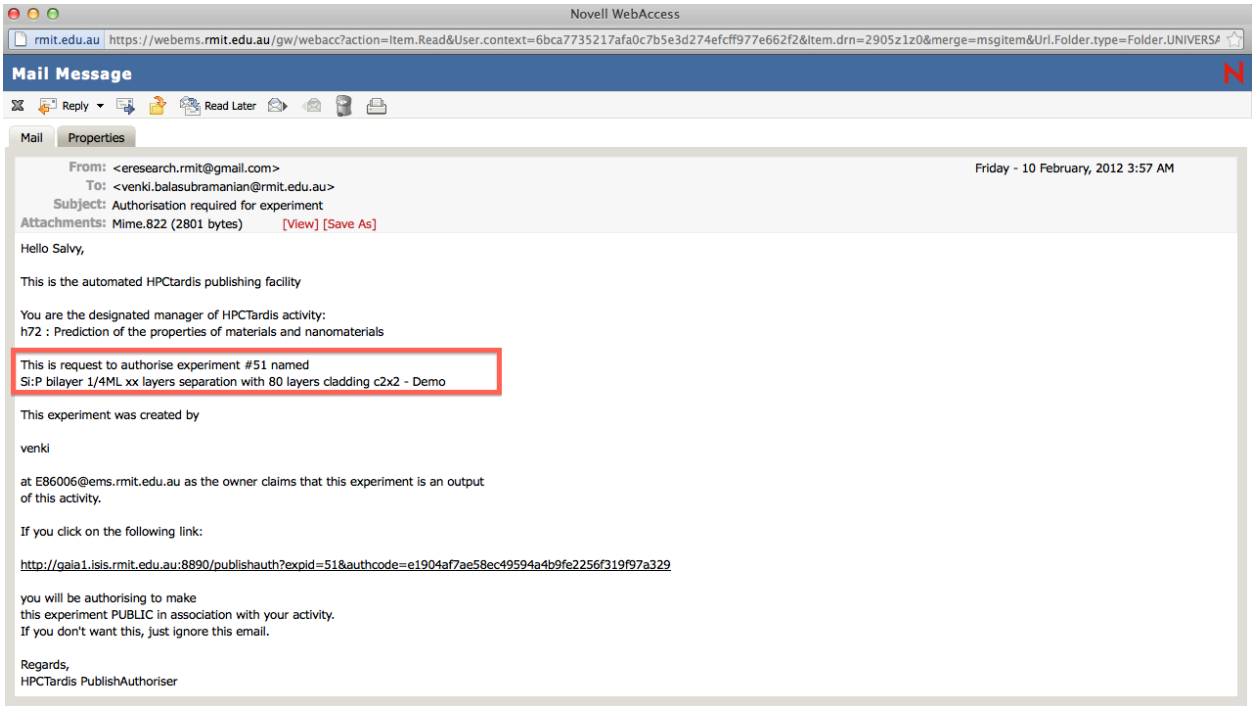
Clicking the publish experiment shows the screen for the user to select the activity i.e. the project associated with this experiment. Any other researcher (party) associated with the data collection can be selected, more than one party can be added. In addition, the user has to accept the legal agreement to publish the experiment.



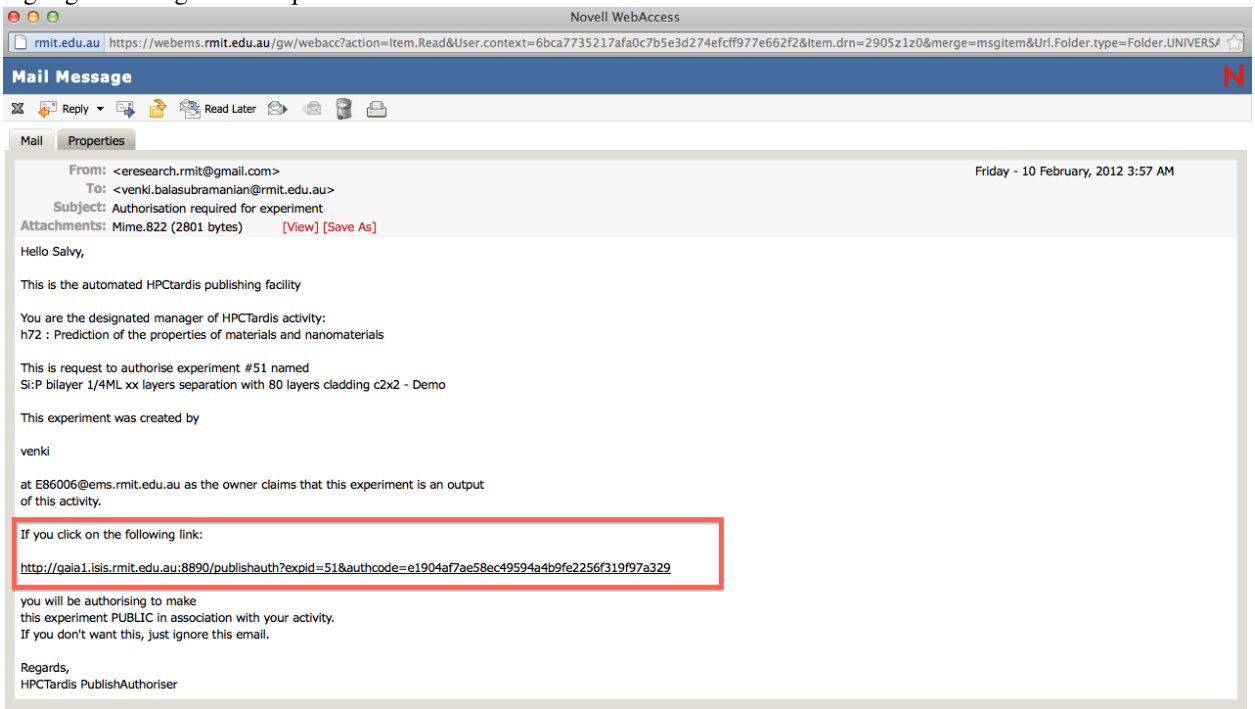
Upon clicking the publish experiment button would generate the web page that shows the confirmation and also automatically sends an email to the manager of the activity for authorisation.



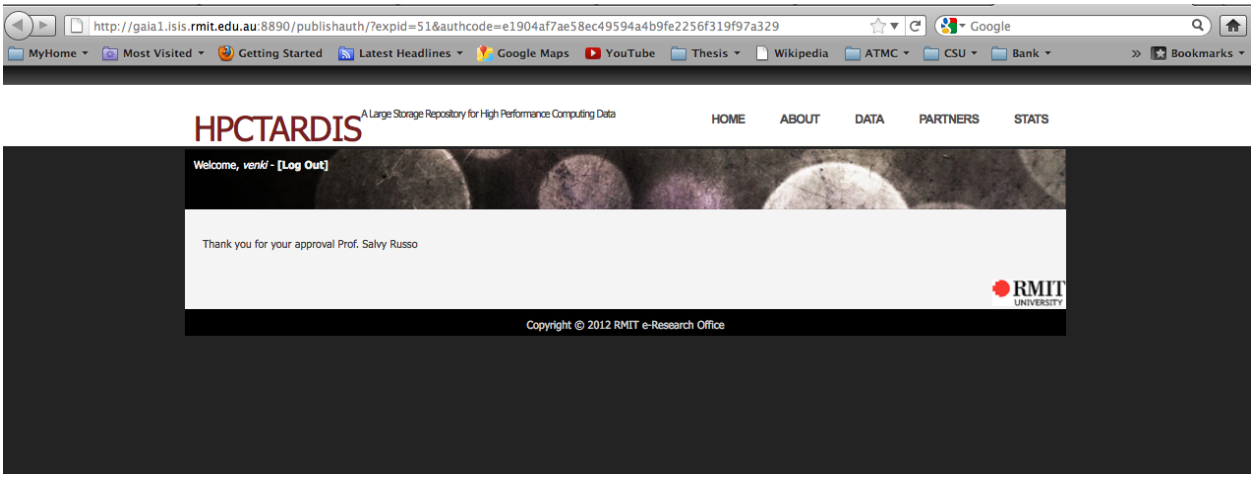
The manager receives an email for authorising the experiment to go public - the red square highlights the experiment name.



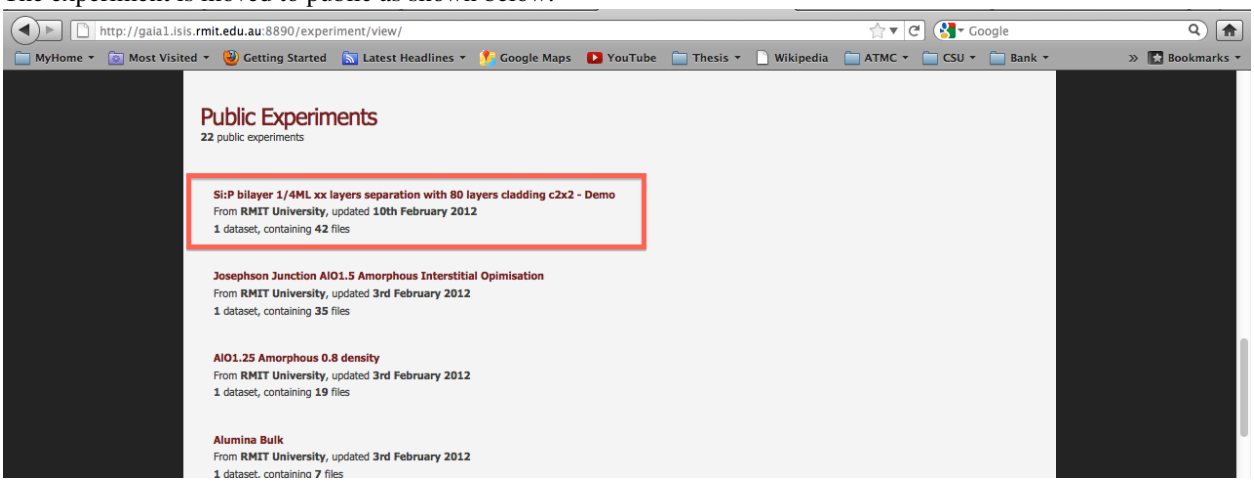
Upon clicking the authorisation link would publish the experiment in the public domain. The authorisation link is highlighted using the red square.



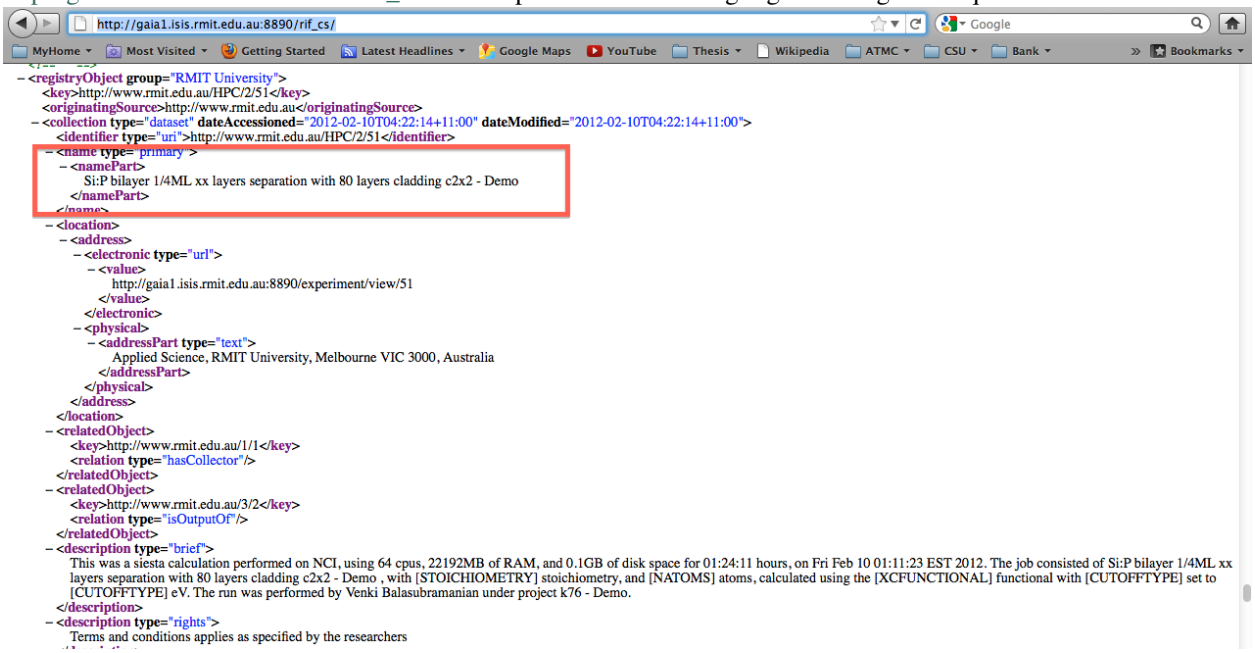
The confirmation web page when the experiment goes public, upon clicking the above authorisation link.



The experiment is moved to public as shown below:



The rif-cs is generated automatically for all the public experiment and can be viewed by using the url: http://gaia1.isis.rmit.edu.au:8890/rif_cs/ .The experiment name is highlighted using a red square.



The RIF-CS is harvested using the OAI-PMH protocol configured in the following url: `http://gaia1.isis.rmit.edu.au:8894/oai/` automatically everyday.

CHAPTER 3

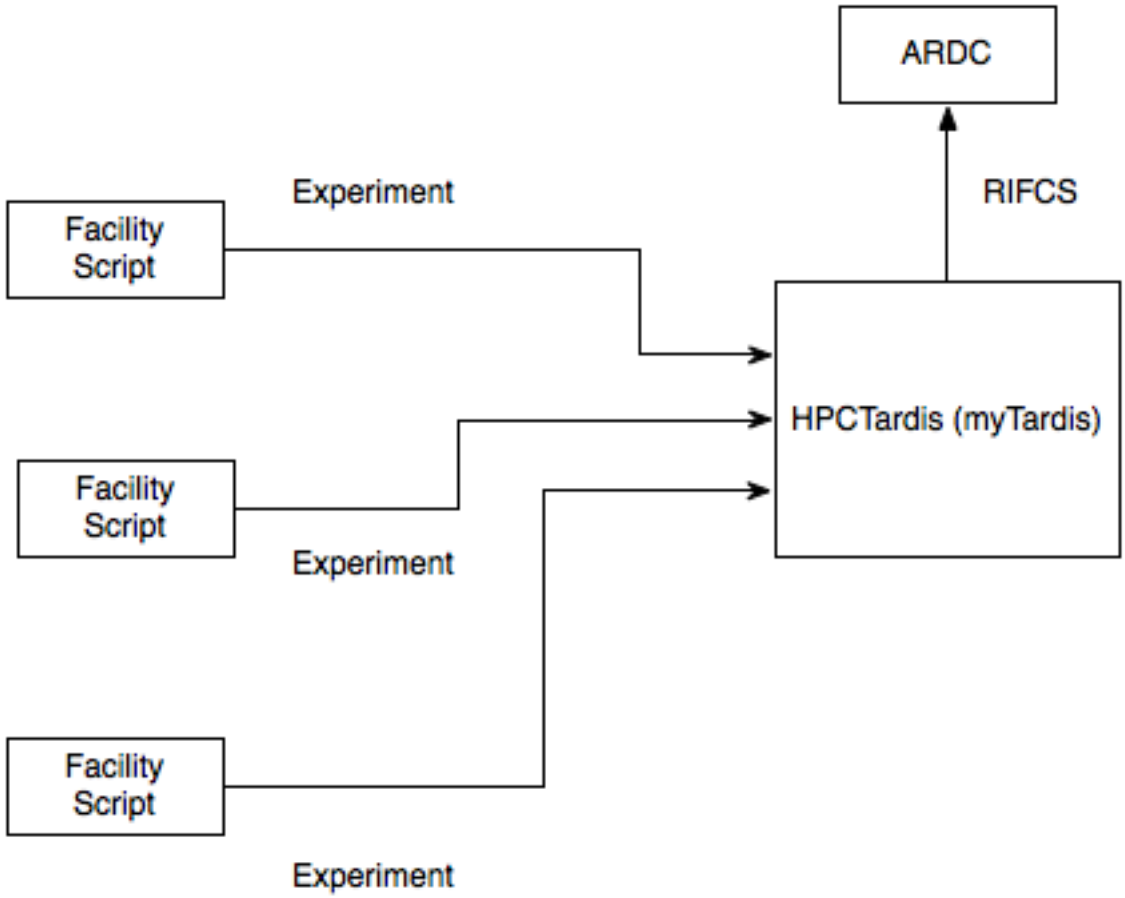
Developer's Manual

This document describes the architecture and implementation of the HPCTardis/MyTardis system. The documentation includes two main parts: the first describes the features of the HPCTardis system for curation of high performance computing datasets; and the second describes the myTardis system on which it is built.

3.1 Introduction

HPCTardis includes two main components: a set of scripts to be deployed on HPC facilities, and a customised and extended version of the myTardis system that adds metadata extraction and ANDS publishing functionality

The diagram below shows the major components of a HPCTardis installation



3.2 Facility Script

The Facility Script consists of two following Bash Scripts :

- `execsum.sh`
- `fta.sh`

The `execsum.sh` script is initated by the user to select datasets or data to be transfered. The `fta.sh` script is initiated by the `execsum.sh` to transfer the data to HPCTardis.

execsum.sh

The script uses a `dialog` utility package in linux to provide a graphical user interface for the user to to select the data from the current or any required directory from the HPC facilities. The script consist of the following major functions to extract adequate information from data:

- `pack_file`

This function checks the valid entry for the entered search directory and the selects the package related files in that particular directory. It lets the users to unselect the automatically selected files.

- `file_desc`

This fucntion creates description for the selected files. The description is created using the canned text. The canned text is made up of various metadata terms required by the users. Therefore, this function is called in the end rather in

the beginning of the updated version of the script.

- `file_exp`

This function extracts the experiment name from the selected data files from the particular package. If the required file for extracting the experiment name is not found, it lets the user to type in the experiment name.

- `file_project`

This function extracts the project name from the selected data files from the particular package. If the required file for extracting the project name is not found, it lets the user to type in the experiment name.

- `exec_summary`

This consolidated all the extracted metadata and form a executive summary for the selected datasets. It also creates a *metadata.<package name>* file using the metadata extracted from the selected datasets. Based on the *metadata.<package name>*, all the selected data files are packaged inside the folder created using values facility name, package name, first 10 characters of the experiment name and the counter file, which is stored locally. The packaged folder is placed in the staging area to be transfered by the file transfer agent script.

The following functions are assistive functions for the above mentioned functions:

- `create_dir`

This function creates required directories such as expeortdir, log and tmp to hold the data files for transfer, to store the log files in case of error during the transfer and to hold various tmp files during the execution of the script.

- `create_config`

This function is used to authenticate the user's tardis username/password. It should be noted that this authentication should be completed in order for the script to proceed. The following *curl* command is used for this purpose:

```
curl -F "username=$TUSERNAME" -F "password=$TPASSWD" -F "authMethod=localdb" $URL/apps/hpctardis/prot
```

A *.config* file is created using the username/password, which in turn would be used by the `fta.sh` to transfer packed folder.

- `deletetempfile`

This function deletes all the temprorily created files.

fta.sh

The functionality of this script is to transfer the files from the HPC facilities to the web based data curation portal HPCTardis. The script moves all the packed folder present in the staging area one by one. Each folder would create experiment in the HPCTardis. The protocol is started by using the following command:

```
curl -F "file=@$EXPORTDIR/$dirname/metadata.$metapack;type=text/plain" -F "username=$TUSERNAME" -F "p
```

The response consist of the following variable the tardis directory, the expeirment ID and the file name:

```
`TARDISDIR=`echo $TMP | awk -F@ '{ print $1 }'` `
`EID=`echo $TMP | awk -F@ '{ print $2 }'``
`FNAME=`echo $TMP | awk -F@ '{ print $3 }'`
```

The received file name and the file name stored in the staging area is compared for security. Once verified, the following rsync command transfer the folder to the tardis directory:

```
rsync -rLtoD --no-p  --chmod=Du+rx,g+srx,o=,Fu+rx,g+rx,o=r --rsh="ssh -i $HOME/.ssh/id_hpc -p $PORTNO
```

Once transfered, the following curl would initiates the function in the HPCTardis to add the files in the created experiment:

```
curl  -d "eid=$EID&desc=$PACKAGE&folder=$dirname&username=$TUSERNAME&password=$TPASSWORD&authMethod=
```

Any error during the transfer would be logged in the *fta-script.log* and also in *<package>.log* files in the log directories.

3.3 HPCTardis/MyTardis

The MyTardis architecture, framework and methodology is described in detail in *Architecture*

We now describe the key additional features of the HPCTardis system: experiment transfer, metadata extraction and ANDS publishing.

HPCTardis additional features are implemented using an application addon at `tardis/apps/hpctardis` which hooks into the API and functionality provided by the myTardis instance at `tardis/tardis_portal`.

3.3.1 Experiment Transfer Protocol

Under Construction

Here we describe the interaction protocol for experiments into HPCtardis

3.3.2 Metadata Extraction

The metadata extraction function analyses datafiles from experiments and extracts domain specific parameter sets according to package specific schemas. See *Overview* for more details.

Configuration

This system is controlled specifically by `tardis/apps/hpctardis/metadata.py`

This package provides

- an engine for matching and extracting metadata.
- a set of utility functions for extracting specific forms of metadata.
- a ruleset data structure to express mappings from datafiles to executions of utility functions and general python code to extract metadata.

Extraction Engine

There are number of different scenarios for metadata extraction in hpctardis handled by this engine:

- The *Create Experiment* Page

  When the user creates a new experiment, the metadata extraction engine is run on each file in each data set individually.

- The *Edit Experiment* Page

  For new files added to an existing experiment, extraction will be repeated. However, existing extracted and edited metadata will not be altered.

- Extraction on ingestion from facility

  The facility protocol triggers metadata on received data at a dataset level.

- Manual editing

  By using the edit/add metadata buttons in the *view experiment* page, individual metadata entries may be added or edited.

- Management Command

  These shell-level commands re-extract metadata and overwrite any existing metadata values and any local changes made by the user.

  Re-extract metadata for specific experiment:

  ```
  bin/django extractmetadata experimentnumber
  ```

  Re-extract metadata for ALL experiments. Use with caution!:

  ```
  bin/django extractmetadata all
  ```

The Rulesets

The list data structure `tardis.apps.hpctardis.metadata.ruleset` defines all available metadata extraction rules for a deployment. The initial package provides a basic set of rules for extracting metadata from VASP, SIESTA, GULP, CRYSTAL simulation packages.

Note that these rules are not definitive for these package and should only form a basis for site specific customisation to requirements of specific researcher groups.

The Grammar for the `ruleset` is as follows:

```
ruleset = [RULESET,
        RULESET,
        ...]

RULEST = (SCHEMA_NS, SCHEMA_NAME):RULES

RULES = [RULE,RULE,...]

RULE = (RULENAME,(DATAFILE_REGEX,), EXTRACTRULE)

RULENAME = the metadata parameter name

DATAFILE_REGEX= the python regex for triggering datafile names in current dataset for the rule

EXTRACTRULE = string of python expression which should return a tuple of ``(value, unit)``.
This rule can contain built-in python functions as well as one or more of the HPCTardis
utility functions to extract required metadata from datafiles.
```

The first time the metadata extraction engine is run and rules are triggered, then schemas for the parameter sets are created in hpctardis and these schemas are used from then onwards for all subsequent extractions. To change an existing ruleset, then specific schema would have to be edited or deleted from the admin tool.

The parameter values created by these rules are strings in the `EXTRACTRULE`, however are interpreted in Tardis of one of two types: STRING and NUMERIC The choice context dependent, such that if the extracted value can be parsed as number, the resulting parameter will be NUMERIC, else it will be STRING in HPCTardis.

See also:

`tardis.apps.hpctardis.metadata.ruleset` in the basic installation for example annotated ruleset examples

Ruleset auxiliary functions

The `EXTRACTRULE` in the ruleset grammer above is a python expression that returns a tuple of value string and unit string. This python expression can contain standard python built-in functions and operators. However, for the standard set of definitions we provide a set of reusable functions useful for extracting metadata files. Additional functions can be written using these as a guide.

The key argument to auxiliary functions is the *context* parameter, which bundles information such as experiment number and file matching regex. See `tardis.apps.hpctardis.metadata._process_datafiles.data_context` for more information.

`tardis.apps.hpctardis.metadata.`**`get_file_line`**(*context*, *lineno*)

Returns the content of relative linenumber. Assumes no unit value and only works for smallish files

Parameters

- **context** – package of parameter data
- **lineno** – the line number of the file to extract

Returns value unit tuple

`tardis.apps.hpctardis.metadata.`**`get_file_lines`**(*context*, *linestart*, *lineend*)

Returns the content of file in the line range. Returns no unit value and only works for smallish local files

Parameters

- **context** – package of parameter data
- **linestart** – the begin of the range of lines to extract
- **lineend** – the end of the range of lines to extract

Returns value unit tuple, where value is newline separated string

`tardis.apps.hpctardis.metadata.`**`get_file_regex`**(*context*, *regex*, *return_max*, ***kwargs*)

Returns the content of the file that matches regex as value, unit pair

Parameters

- **context** – package of parameter data
- **regex** – regex with named groups value and unit
- **return_max** – if true, context filename regex must contain single group. If multiple files match this group, then only one with largest numeric is used.
- **kwargs** – if 'nextline' in kwargs and true then return whole line after matching regex

Returns value unit tuple

`tardis.apps.hpctardis.metadata.`**`get_file_regex_all`**(*context*, *regex*)

Returns the content of the file that matches regex as a list of value, unit pairs

Parameters

- **context** – package of parameter data
- **regex** – regex with named groups value and unit

Returns list of <value,unit> tuples

`tardis.apps.hpctardis.metadata.`**`get_regex_lines`**(*context*, *startregex*, *endregex*)

Returns the file content of lines that match between two matching regular expressions.

Returns blank unit value and only works for smallish files

Parameters

- **context** – package of parameter data
- **startregex** – starting regex for extraction
- **endregex** – ending regex for extraction

Returns value unit tuple

`tardis.apps.hpctardis.metadata.`**`get_regex_lines_vallist`**(*context*, *startregex*, *endregex*)

Returns the file content of lines that match between two matching regular expressions as a list of lines

Parameters

- **context** – package of parameter data
- **startregex** – starting regex for extraction
- **endregex** – ending regex for extraction

Returns list of strings

`tardis.apps.hpctardis.metadata.`**`get_final_iteration`**(*context*)

Returns the final iteration number from a VASP run

Parameters **context** – package of parameter data

Returns value unit tuple

`tardis.apps.hpctardis.metadata.`**`get_constant`**(*context*, *val*, *unit*)

Create a constant value unit pair

Parameters

- **val** – value of the constant
- **unit** – the unit of the constant

Returns value unit tuple

3.3.3 ANDS Publishing

An experiment in hpctardis can be be in two states: *private* and *public*:

- A private experiment is accessible only to the owner of the experiment unless additional experiment ACLs are created (see *auth – Auth Framework*)
- A public experiment is enabled for read-only access to authorised and external users to experiment metadata. If `settings.PRIVATE_DATAFILES` is `True`, then download permission for experiments and datafiles is also *disabled* and redirected to a contact page. Otherwise, downloading of datasets is enabled. Furthermore public experiments and associated party and activity links are accessible at http://127.0.0.1/rif_cs for direct ingestion of the RIF_CS XML.

An experiment is made public through the *publish_experiment* page available from the the *experiment details* page.

An experiment goes through a number of steps to become public:

- The owner must acknowledge legal agreement and select at *least* one of available activity records. They may also choose to associate any party records and their relations.
- For each selected activity the system looks for a party record associated with it through 'ManagedBy' relation. The email location field for that person is queried and an email is sent to this email address.
- The authoriser selects the validation link in the received email.

- When *all* authorising parties have acknowledged then the experiment is made public and records are created in the RIF-CS feed.

Implementation Notes

The `tardis.apps.hpctardis.publish` package handles aspects of publishing in HPCTardis.

- `tardis.apps.hpctardis.publish.rif_cs_profile.profiles.default.xml` is the default template for rending collection records in HPCTardis. Edit as needed.
- `tardis.apps.hpctardis.publish.rif_cs_profile.rif_cs.xml` is the template for rending RIF_CS, specifically party and activity records in HPCTardis. Edit as needed.

The state of the publishing of an experiment handled by the `PublishAuthorisation` model, accesible via the admin tool.

Activity, Party and associated relation records are created using the admin tool.

To restore an experiment to private from public, change the flag in the `Experiment` model using the admin tool.

Management Commands

Resend any outstanding validation emails for an experiment:

```
bin/django publish resend experimentnumber
```

Check all validation requests and if all have been approved, promote experiment to public:

```
bin/django publish promote experimentnumber
```

This command is useful after changing status of experiment in PublishAuthorisation model using the admin tool. For example, to manually authorise an experiment without email, edit the corresponding entry PublishAuthorisations to APPROVAL state, then this command will set the public state for the experiment.

3.4 MyTARDIS

MyTARDIS is a multi-institutional collaborative venture that facilitates the archiving and sharing of data and metadata collected at major facilities such as the Australian Synchrotron and ANSTO and within Institutions.

An example of the benefit of a system such a MyTARDIS in the protein crystallography community is that while the model coordinates and (less often) the structure factors (processed experimental data) are stored in the community Protein Data Bank (PDB) the raw diffraction data is often not available. There are several reasons why this is important, which can be summarised as:

- The availability of raw data is extremely useful for the development of improved methods of image analysis and data processing.
- Fostering the archival of raw data at an institutional level is one the best ways of ensuring that this data is not lost (laboratory archives are typically volatile).

3.4.1 Homepage

You can get a copy of MyTARDIS from http://code.google.com/p/mytardis/

3.5 Quick Start

3.5.1 Prerequisites

Redhat:

```
sudo yum install cyrus-sasl-ldap cyrus-sasl-devel openldap-devel libxslt libxslt-devel libxslt-python
```

Debian/Ubuntu:

```
sudo apt-get install libsasl2-dev libldap libldap2-dev libxslt1.1 libxslt1-dev python-libxslt1
```

3.5.2 Download

To get the current trunk:

```
svn co https://mytardis.googlecode.com/svn/trunk/ mytardis
cd mytardis
```

3.5.3 Building

MyTARDIS is using the buildout buildsystem to handle dependencies and create the python class path:

```
python bootstrap.py
./bin/buildout
```

MyTARDIS can now be executed in it's simplist form using:

```
./bin/django runserver
```

This will execute django using the builtin SQLite DBMS and only be accessable on *localhost*.

3.5.4 Prerequisites

- Linux or similar server
- Python 2.6

3.5.5 Installation

Unpack the distribution or pull from the git repository

3.5.6 Configuration

This page describes how to install MaVRec as a local server, using SQLite3 as a backend with initial data.

Install dependencies (from Ubuntu, use Yum for RedHat etc.):

```
sudo apt-get install build-essential python-dev
sudo apt-get install subversion
```

Bootstrap build environment:

```
cd smra
python2.6 boostrap.py
```

Get required system python dependencies (internet required):

```
bin/buildout
```

Run testcases to verify success:

```
bin/django test
```

Copy prototypical settings file for local version:

```
cp smra/settings_changeme.py smra/settings.py
```

If required, modify standard Django `smra/settings.py` file to change database etc. Documentation in `settings_changeme.py`

The configure MaVRec for interactive use, modify the file `bin/django` and replace:

```
djangorecipe.manage.main('smra.test_settings')
```

with:

```
djangorecipe.manage.main('smra.settings')
```

Setup database and initial data and create an admin user:

```
bin/django syncdb --migrate
```

Start the development server:

```
bin/django runserver
```

System should now be running at http://127.0.0.1:8000

3.6 Contents

3.6.1 Install

Prerequisites

Redhat:

```
sudo yum install cyrus-sasl-ldap cyrus-sasl-devel openldap-devel libxslt libxslt-devel libxslt-pytho
```

Debian/Ubuntu:

```
sudo apt-get install libssl-dev libsasl2-dev libldap-2.4-2 libldap2-dev libxslt1.1 libxslt1-dev pytho
```

Configuration

Configuring MyTARDIS is done through a standard Django *settings.py* file there are some extra configuration options that are specific to MyTARDIS.

`tardis.settings_changeme.`**`FILE_STORE_PATH`**

The location of the file store.

Database

`tardis.settings_changeme.`**`DATABASE_ENGINE`**
The database server engine that will be used to store the MyTARDIS metadata, possible values are *postgresql_psycopg2*, *postgresql*, *mysql*, *sqlite3* or *oracle*.

`tardis.settings_changeme.`**`DATABASE_NAME`**
The name of the database to used to store the data, this is the path to the database if you are using the SQLite storage engine.

`tardis.settings_changeme.`**`DATABASE_USER`**
The user name used to authenticate to the database. If you are using SQLite this field is not used.

`tardis.settings_changeme.`**`DATABASE_PASSWORD`**
The password used to authenticate to the database. If you are using SQLite this field is not used.

`tardis.settings_changeme.`**`DATABASE_HOST`**
The host name of the machine hosting the database service. If this is empty then localhost will be used. If you are using SQLite then this field is ignored.

`tardis.settings_changeme.`**`DATABASE_PORT`**
The port the database is running on. If this is empty then the default port for the database engine will be used. If you are using SQLite then this field is ignored.

LDAP

For further information see *LDAP authentication*

Repository

`tardis.settings_changeme.`**`FILE_STORE_PATH`**
The path to the MyTARDIS repository. This i where files will be copied to once they are ingested into the system.

`tardis.settings_changeme.`**`STAGING_PATH`**
The path to the staging path. This is where new files to be included in datasets will be sourced.

Filters

`tardis.settings_changeme.`**`POST_SAVE_FILTERS`**
This contains a list of post save filters that are execute when a new data file is created.

The **POST_SAVE_FILTERS** variable is specified like:

```
POST_SAVE_FILTERS = [
    ("tardis.tardis_portal.filters.exif.EXIFFilter", ["EXIF", "http://exif.schema"]),
    ]
```

For further details please see the *filters – Filter Framework* section.

See also:

http://www.buildout.org The Buildout homepage.

3.6.2 Database Maintenance

initialising

When creating a new database the `syncdb` command will need to be called to initialise the schema and instert the initial data fixtures.

Usage

```
./bin/django syncdb
```

migrating

Some of the upgrades to MyTARDIS will require that the database schema be upgraded to match the internal datamodel. This tool migrates data from old database schemas to the current one. It detects which version of the database you are currently running and will automatically migrate to the current version.

Usage

```
./bin/django migrate
```

backup

The backupdb command allows to backup and to restore of the MyTARDIS database. The command uses the corresponding database tool to facilitate this task. Currently implemented are PostgreSQL and MySQL. In case of backup, a directory called `backups` is created (if it does not exists) in the working directory. In case of restore, the database for storing the tablespace must already exist before loading the backup file into the database.

Usage

```
./bin/django backupdb
```

-r FILE, **--restore**=FILE

-v VERBOSITY, **--verbosity**=VERBOSITY

--settings=SETTINGS

--pythonpath=PYTHONPATH

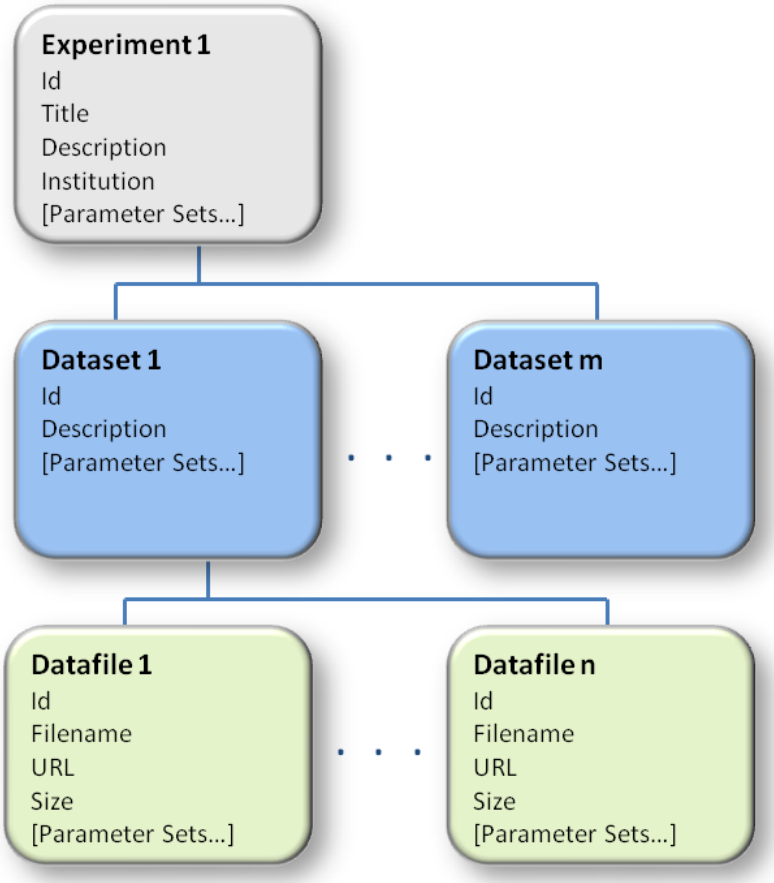
--traceback

--version

-h, --help

3.6.3 Overview

At the simplest level, the experimental data is simply a collection of files (Datafiles), which are grouped in to Datasets, which are grouped in to Experiments:



(Please note that the schema is only partially shown in the diagram above)

At each level, Experiment, Dataset and Datafile, administrator defined parameters may be added, grouped in to Parameter Sets.

Tardis doesn't impose any interpretation on what is considered an Experiment or Dataset. Examples of how datasets may be grouped are: by sample, by instrument settings, or as a time sequence, e.g. artificially aging a material and investigating the effects.

3.6.4 Schema and Parameter Sets

MyTARDIS stores metadata as *Parameters*, which are grouped in to *Parameter Sets*, which are defined by a *Schema*.

Managing Schema

MyTARDIS currently relies on the in-built Django administrative interface for managing schema. The administrative interface is accesible from a link similar to:

```
http://localhost/admin/
```

Schema definitions are the combination of two tables, Schema and ParameterName.

The Schema record identifies the schema (*namespace*), with a ParameterName record for each attribute of the schema. Other attributes of the parameter include the type, e.g. string, number, URL, date, enumerated list, etc., whether the parameter is searchable or not, and if it is an enumerated list, the valid choices.

Note: Creating a proper schema administration page is scheduled to be completed by December 2011.

3.6.5 Ingesting

There are three mechanisms for ingesting metadata and data into MyTARDIS:

1. User Interface The User Interface is appropriate for ingesting a single experiment by the end user with a relatively small amount of data.
2. Staging Area The Staging Area is appropriate for ingesting a single experiment by the end user with larger amounts of data.
3. Batch Ingestion Batch ingestion is typically used by facilities automatically ingesting all metadata from one or more instruments into MyTARDIS.

MyTARDIS supports 2 different XML schemas for importing metadata. One method is METS and the other is using a MyTARDIS specific XML format. METS is the preferred format because it is supported by systems other that MyTARDIS so will provide more versatility in the long run.

METS

The Meta-data Encoding and Transmission Standard was recommended by Monash Librarians and ANU in 2008 as the XML description format for MyTARDIS datasets.

Ingestion Script

Metadata may be easily ingested using a simple script and POST request:

```
#!/bin/bash

file="$1"
username="localdb_admin"
password="secret"
host="http://localhost:8000"
owner="$username"

curl -F username=$username -F password=$password -F xmldata=@${file} -F experiment_owner=$owner "$hos
```

To use this script paste it into a new file called, e.g. *register.sh*, *chmod +x register.sh* then can call it using *./register.sh file.xml*. There are several example XML and METS files within the tardis test suite.

Post Processing

MyTARDIS takes advantage of the Django signal framework to provide post processing of files. The only default post processing step that is enabled by default operates on newly created Dataset Files.

Staging Hook

The staging hook is responsible for moving files from the staging area to the data store. It operates as a `django.db.models.signals.post_save` signal and only triggers in a newly created file.

The staging hook is only triggerd on files that have a protocol of *staging* which signifies that the file is in the in the TARDIS staging area.

EXIF Metadata extraction

class `tardis.tardis_portal.filters.exif.`**`EXIFFilter`**(*name*, *schema*, *tagsToFind*=[], *tagsToExclude*=[])

This filter provides simple metadata extraction of EXIF tags from images.

If a white list is specified then it takes precidence and all other tags will be ignored.

Parameters

- **name** (*string*) – the short name of the schema.
- **schema** (*string*) – the name of the schema to load the EXIF data into.
- **tagsToFind** (*list of strings*) – a list of the tags to include.
- **tagsToExclude** (*list of strings*) – a list of the tags to exclude.

`getExif`(*filename*)

Return a dictionary of the metadata.

`getParamaters`(*schema*, *metadata*)

Return a list of the paramaters that will be saved.

`getSchema`()

Return the schema object that the paramaterset will use.

`saveExifMetadata`(*instance*, *schema*, *metadata*)

Save all the metadata to a Dataset_Files paramamter set.

See also:

http://www.loc.gov/standards/mets/ Metadata Encoding and Tranmission Standard

3.6.6 `auth` – Auth Framework

The main purpose of the reworked Auth system is to allow per experiment permissions to exist allowing a richer web experience. Because of this the permissions are applied on a per experiment basis with a few predefined roles.

read read permission allows individuals and groups access to view an experiment.

write write permissions cover addition of new datasets and datafiles and also deletion of datafile.

delete delete permission allows deletion of datasets and experiments.

Roles are applied through the web using the *Control Panel* and can be applied to either users or groups.

To make an experiment public requires and explicit publish action.

In the *settings.py* user providers are activated by specifying them within the **USER_PROVIDERS** variable:

```
USER_PROVIDERS = ('tardis.tardis_portal.auth.localdb_auth.DjangoUserProvider',)
```

In the *settings.py* group providers are activated by specifying them within the **GROUP_PROVIDERS** variable:

```
GROUP_PROVIDERS = ('tardis.tardis_portal.auth.localdb_auth.DjangoGroupProvider',
                   'tardis.tardis_portal.auth.vbl_auth.VblGroupProvider',)
```

AuthService Objects

class tardis.tardis_portal.auth.**AuthService**(*settings=<django.conf.LazySettings object at 0x4065790>*)

The AuthService provides an interface for querying the auth(n|z) framework within MyTARDIS. The auth service works by reading the class path to plugins from the settings file.

Parameters **settings** (`django.conf.settings`) – the settings object that contains the list of user and group plugins.

authenticate(*authMethod*, ***credentials*)

Try and authenticate the user using the auth type he/she specified to use and if authentication didn't work using that method, try each Django AuthProvider.

Parameters

- **authMethod** (*string*) – the shortname of the auth method.
- ****credentials** – the credentials as expected by the auth plugin

getGroups(*request*)

Return a list of tuples containing pluginname and group id

Parameters **request** (`django.http.HttpRequest`) – a HTTP Request instance

getGroupsForEntity(*entity*)

Return a list of the groups an entity belongs to

Parameters **entity** (*string*) – the entity to earch for, user or group.

The groups will be reurned as a list similar to:

```
[{'name': 'Group 456', 'id': '2'},
{'name': 'Group 123', 'id': '1'}]
```

getUser(*user_dict*)

Return a user model based on the user dict.

This function is responsible for creating the user within the Django DB and returning the resulting user model.

searchEntities(*filter*)

Return a list of users and/or groups

searchGroups(***kw*)

Return a list of users and/or groups

Parameters

- **id** – the value of the id to search for
- **name** – the value of the displayname to search for
- **max_results** – the maximum number of elements to return
- **sort_by** – the attribute the users should be sorted on
- **plugin** – restrict the search to the specific group provider

searchUsers(*filter*)

Return a list of users and/or groups

Auth Plugins

- `tardis.tardis_portal.auth.localdb_auth`
- `tardis.tardis_portal.auth.ldap_auth`
- `tardis.tardis_portal.auth.ip_auth`

3.6.7 `filters` – Filter Framework

Filters are called once an object has been saved to the database, they build on the Django signal infrastrcture.

In the *settings.py* file filters are activated by specifying them within the **POST_SAVE_FILTERS** variable:

```
POST_SAVE_FILTERS = [
    ("tardis.tardis_portal.filters.exif.EXIFFilter", ["EXIF", "http://exif.schema"]),
    ]
```

The format they are specified in is:

```
(<filter class path>, [args], {kwargs})
```

Where **args** and **kwargs** are both optional.

Filter Plugins

- `tardis.tardis_portal.filters.exif.make_filter`

EXIF Filter

`tardis.tardis_portal.filters.exif.`**`make_filter`**(*name='', schema='', tagsToFind=[], tagsToExclude=[]*)

This filter provides simple metadata extraction of EXIF tags from images.

If a white list is specified then it takes precidence and all other tags will be ignored.

Parameters

- **name** (*string*) – the short name of the schema.
- **schema** (*string*) – the name of the schema to load the EXIF data into.
- **tagsToFind** (*list of strings*) – a list of the tags to include.
- **tagsToExclude** (*list of strings*) – a list of the tags to exclude.

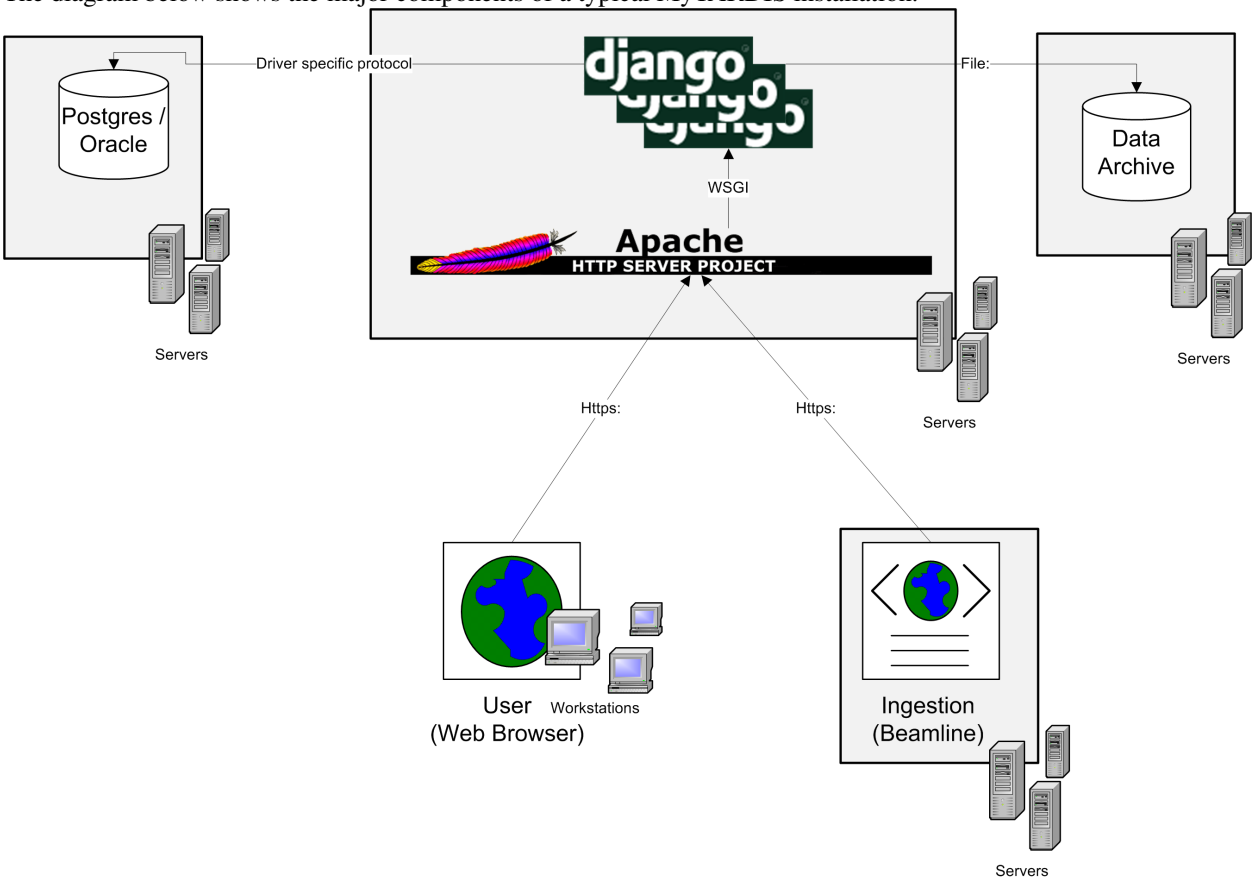
3.6.8 Architecture

This page describes the architecture of MyTARDIS.

MyTARDIS is built on the Django web framework, which itself is built on Python, thus MyTARDIS follows the architectural model of Django.

Component Architecture

The diagram below shows the major components of a typical MyTARDIS installation.



Web Server MyTARDIS is typically deployed with the standard Apache Http + WSGI + Django + Python stack.

RDBMS Any of the Django supported databases may be used, typically Postgres.

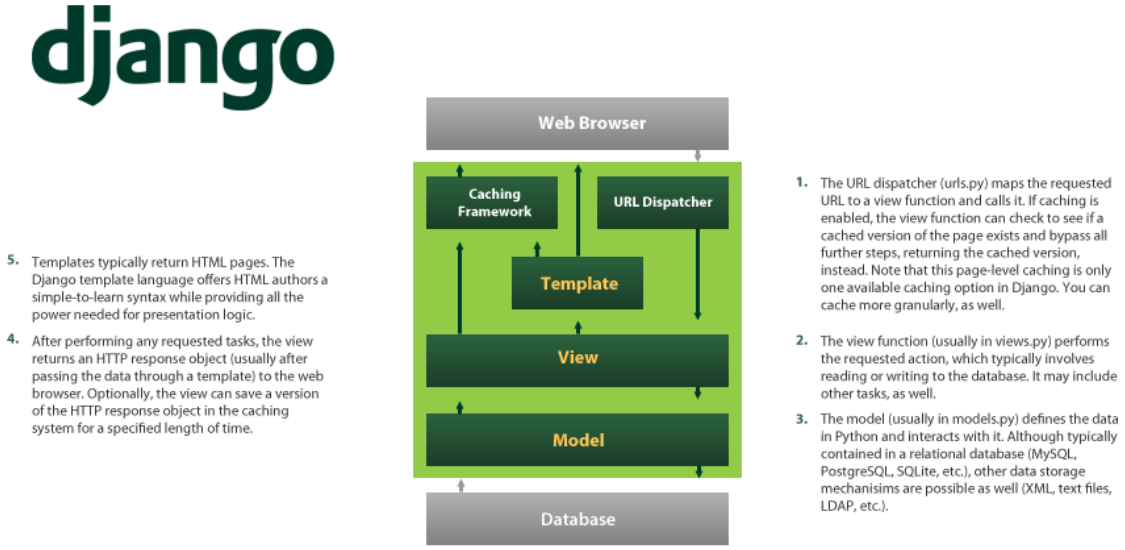
Data Archive Normal file server, e.g. NFS, SAMBA.

Browser Typically Firefox

Ingestion Ingestion is normally site specific.

Functional Architecture

MyTARDIS follows the standard Django architecture, as shown below [JCDA]:



The major functional modules in MyTARDIS are:

Auth Authentication and Authorisation.

Download Data download functionality.

Filters Processing of metadata as data is uploaded, e.g. extract EXIF metadata from JPEG files.

Management Additional commands for the Django CLI. The backup utility is implemented as a Django command.

Migrations South database migration code.

Publish RIF-CS metadata publication.

Information on the individual modules is available from the *modindex*.

Scalability Model

At the component / module level, performance and Scalability within MyTARDIS is achieved by:

- Allowing long lived or compute intensive operations to be hosted on separate machines from the main web server.
  - E.g. by running multiple web servers and sending long lived operations, such as ingestion, to a server reserved for such operations.
- Performance and Scalability of the database is achieved by a combination of 1) optimising the SQL requests issued by MyTARDIS, and 2) database specific scalability, please refer to the appropriate database documentation.
- Performance and Scalability of the web server is provided through the normal mechanisms for apache / wsgi hosted python applications, e,g,:
  - Increasing individual Server capability
    - Individual Server performance / utilization may be managed by controlling the number of python (django) processes active at any one time.

  - Deploying multiple web servers
    * Multiple web servers may be deployed using standard mechanisms, e.g. hardware load balancers. State (session) information is distributed using Django's standard session model.
- The Data Archive is a normal file system, e.g. NFS, SAMBA, etc., with performance and scalability dependent on the implementation and deployment.
- Extraction and formatting of metadata for ingestion is up to the client and may be distributed across any number of machines.

SQL Scalability Notes

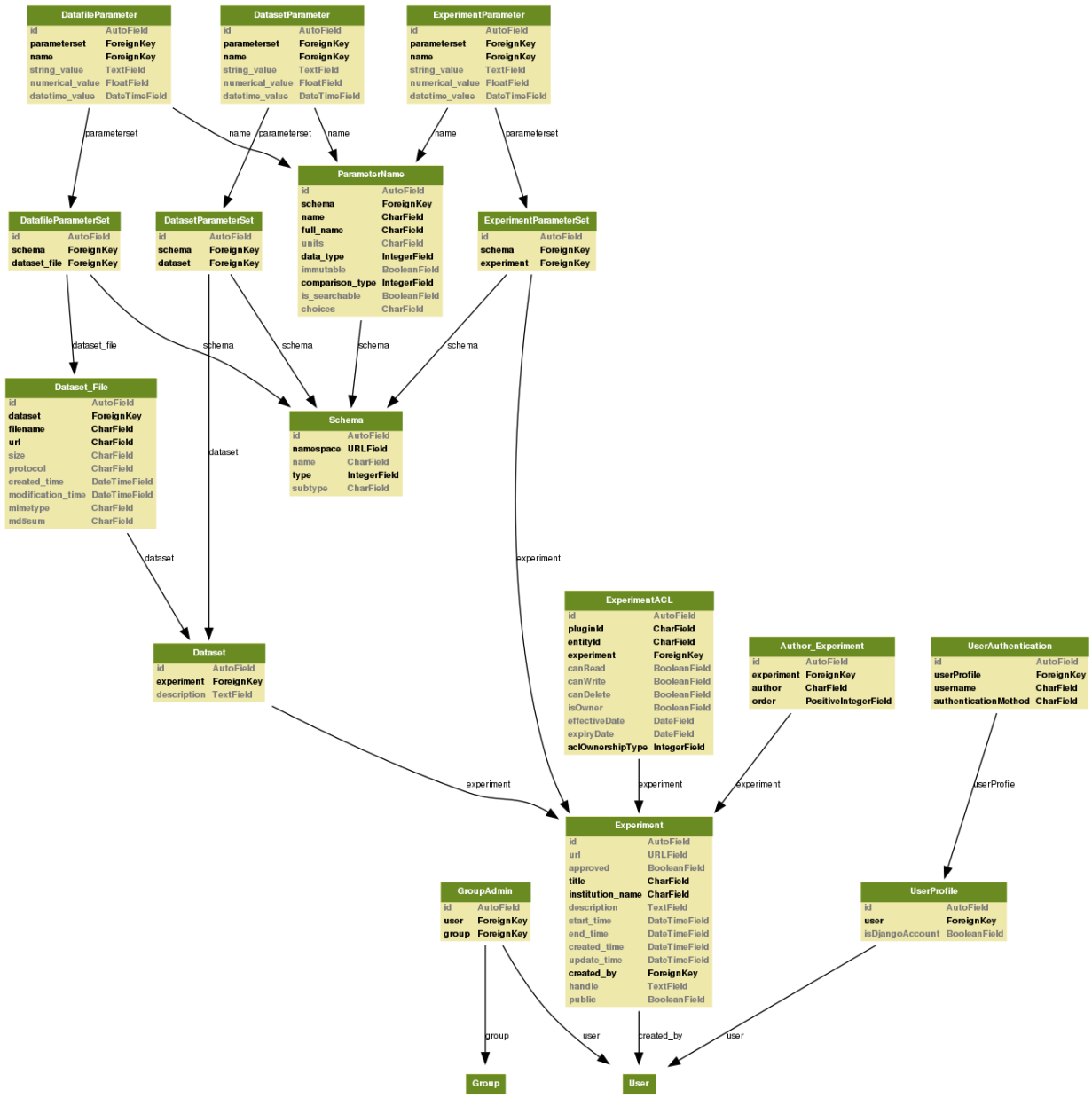
The datafileparameter table is expected to be the single largest table by an order of magnitude, in the hundreds of millions of records (at the Australian Synchrotron).

For Postgres, this will probably be addressed by table partioning. In this case using a date field to partition is often recommended. Dates may be introduced as part of the support for anotations, raw data, versioned data and derived data.

Persistance and Data Architecture

An introduction to how Tardis models Experiments, Datasets and Datafiles is provided in the *Overview*.

Django provides an Object-Relational-Model that maps Django Models to the underlying relational database. The models as defined in Django are:



An additional model diagram including the base Django models is also available: `ref/models-graphviz-all`.

The underlying SQL schema generated by Django is not normally referenced directly as it is managed through Django's ORM, however a version of the schema is available: `ref/sql-graphviz`.

Security Model

Authentication

Access to data catalogued by MyTARDIS may be either public, i.e. anonymous access allowed, or private, requiring authentication. The Authentication Framework is documented in *auth – Auth Framework*.

Authorisation

MyTARDIS includes an extensible authorisation engine, documented in *auth – Auth Framework*.

Web Server Security

The Django framework provides security against:

- Cross Site Request Forgery protection
- SQL Injection Attacks

Class Diagram

Unless the application has a very rich business model (which MyTARDIS does not), a Class diagram doesn't convey much information in Django applications (it ends up being a flat list of classes). To understand the context in which the MyTARDIS classes are created, please refer to the Django documentation, http://docs.djangoproject.com/ on the responsibilities of models, views, managers and templates.

3.6.9 Changelog

2.0 - Unreleased

- Auth/Auth redesign [Gerson, Uli, Russel]
  - Authorisation. Support for several pluggable authorisation plugins (Django internal, LDAP, VBL). The added AuthService middleware provides a mechanism to query all available auth modules to determine what group memberships a users has.
  - Alternative authorisation. Rule based experiment access control engine was implemented with the following access attributes for indivdual users and groups: canRead, canWrite, canDelete, isOwner. Additionally, a time stamp can be specified for each access rule.

    Further information can be found at the wiki: Authorisation Engine design
- Metadata Editing [Steve, Grischa]
- New METS parser & METS exporter [Gerson]
- Dist/Buildout infrastructure [Russell]
- Through the web creation and editing of experiments [Steve, Russell]
- Through the web upload of files [Steve]
- Download protocol handler [Russel, Uli]
- Logging framework [Uli]
- Django 1.3

1.07 - 01/06/2010

- Publish to tardis.edu.au interface created, though not implemented, pending legal text

1.06 - 15/03/2010

- Parameter import interface for creation of new parameter/schema definitions
- iPhone Interface

1.05 - 01/03/2010

- Images as parameters supported
- Data / metadata transfer from synchrotron is now 'threaded' using asynchronous web service transfers.

1.0 - 01/02/2010

- MyTARDIS created from existin MyTARDIS python / django codebase
- Allows private data to be stored
- Open key/value parameter model, replacing current crystallography one
- Internal data store for data
- LDAP Login
- Pagination of files
- Creation of synchrotron-tardis from MyTARDIS codebase including specific code for the VBL login service and data transfer to MyTARDIS deployments.
- Web server changed to apache and mod_wsgi

0.5 - 2009

- Re-wrote federated index (python / django)
- Federated stores are now simple web server based with optional FTP access
- Runs on Jython / Tomcat

0.1 - 2007

- Federated index (php) running on Apache HTTP Server
- Crystallography data deposition and packaging tools for Fedora Commons (java swing desktop)
- Search Interface via web

3.6.10 Contributing

Development

Source

The MyTARDIS project is hosted on Google Code and uses an subversion repository for source code control.

Creating a Patch with SVN

SVN users who whish to contribute code please checkout a copy of the current source:

```
svn checkout http://mytardis.googlecode.com/svn/trunk/ mytardis-read-only
```

You'll need to make sure SVN tracks any new files you add using:

```
svn add path/to/new/file
```

Make the patch using:

```
svn diff > feature_name_or_bugfix.diff
```

and then add an issue to the issue tracker

Generating Documentation

Documentation is done in sphinx and can be built using the commands provided by the sphinx buildout recipe:

```
./bin/sphinxbuilder
```

Hudson

Hudson is a continuous intergration server that is used within the MyTARDIS development process to help maintain the quality of the project. Within Hudson we run use the following script to execute the build and run the tests.

```
#!/bin/bash
rm dist/*
[ -d egg-cache ] || mkdir -p egg-cache
export PYTHON_EGG_CACHE=`pwd`/egg-cache
python setup.py clean
python bootstrap.py
./bin/buildout
./bin/django-admin.py test --with-xunit --with-coverage --cover-package=tardis.tardis_portal --with-x
python setup.py sdist
```

Enable the Publish JUnit test result report in the Post-Build Actions section and use specify the nost tests results output:

```
nosetest.xml
```

To enable reporting of the code coverage you'll need to install the Hudson Cobertura plugin, once that is installed you can specify the location of the coverage.xml file:

```
**/coverage.xml
```

See also:

Hudson Extensible continuous integration server

3.7 Indices and tables

- *genindex*
- *modindex*

- *search*

CHAPTER 4

Storage Migration

This section describes the steps required to move the HPCTardis collections from one facility to another, specifically how to dump or migrate the data store form one database backend to another and the structure of the file store of experimental data.

4.1 Metadata Migration

All metadata for myTardis is kept in an external SQL database. Configuration of that database can be found in `settings.py`.

4.1.1 Database Specific Migration

For migrating from specific database backend to the same backend use the appropriate migration tool for the specific backend.

For example, for MySQL the `mysqhotcopy` or `mysqldump` tools would be appropriate.

4.1.2 Generic Data Migration

For migrating data between different databases, HPCTardis uses the django dumpdata command to produce a database agnostic file

To create a snapshot of the database:

```
bin/django dumpdata --indent=1  > fulldump.json
```

This will create a large JSON format dump file for all information in the database. This file can be edited and filtered as needed.

To recreate and load the database in a new database (after syncdb):

```
bin/django loaddata fulldump.json
```

See also:

django-admin dumpdata - https://docs.djangoproject.com/en/dev/ref/django-admin/#django-admin-dumpdata

4.1.3 Schema Migration

HPCTardis uses the South schema and migration system for migration to new versions of `models.py`

If the database schema is changed a schema migration file must be created. Then the following command should move the current database to the new schema:

```
bin/django syncdb --migrate
```

See also:

http://south.aeracode.org/ - South Data and Schema migration tool

http://www.mysql.com - MySQL

4.2 Dataset Migration

Experiments, datasets and datafile tables in the django database correspond to assets in the associated *TardisStore*. The location of that store is set in the `settings.py` file using the variable `FILE_STORE_PATH`.

The format for data at this location is

```
EXPERIMENT_ID/DATASET_ID*/DATA
```

Where `EXPERIMENT_ID` is the experiment primary key, `DATASET_ID` is a set of directories named with the dataset primary key and `DATA` is the directory structure of individual files and directories corresponding to the datafiles.

As all tardis dataset information is stored using a standard filesystem migrating of that data would require only copying the filesystem using standard methods.

CHAPTER 5

`views` – View level functions and utilities

CHAPTER 6

models – Models

6.1 NameParts Objects

class tardis.apps.hpctardis.models.**NameParts**(**args*, ***kwargs*)

The component parts of a name used in parties and activities

exception DoesNotExist

exception NameParts.**MultipleObjectsReturned**

NameParts.**activityrecord_set**

NameParts.**objects = <django.db.models.manager.Manager object at 0x5119350>**

NameParts.**reverse**

6.2 PartyRecord Objects

class tardis.apps.hpctardis.models.**PartyRecord**(**args*, ***kwargs*)

A record for an ANDS party

Attribute key the full ANDS URI identifier

Attribute type the designation of the type of party

Attribute partyname the nameparts representation of the party name

Attribute repos a foreign key to the class smra.smra_portal.models.Repository

Attribute schemas a many-to-many key to the class smra.smra_portal.models.Schema via class smra.smra_portal.models.MediaObjectParameterSet

exception DoesNotExist

exception PartyRecord.**MultipleObjectsReturned**

PartyRecord.**activitypartyrelation_set**

PartyRecord.**activityrecord_set**

PartyRecord.**get_email_addresses**()

PartyRecord.**get_full_record**()

PartyRecord.**get_fullname**()

PartyRecord.**get_type_display**(**moreargs*, ***morekwargs*)

PartyRecord.**objects** = **<django.db.models.manager.Manager object at 0x510ca10>**

PartyRecord.**partydescription_set**

PartyRecord.**partylocation_set**

PartyRecord.**partyname**

PartyRecord.**publishauthorisation_set**

6.3 PartyLocation Objects

class tardis.apps.hpctardis.models.**PartyLocation**(**args*, ***kwargs*)

A location for a party

Attribute type ANDS designation of type of location

Attribute value the value for the location

Attribute party the party that the location is associated with

exception DoesNotExist

exception PartyLocation.**MultipleObjectsReturned**

PartyLocation.**get_type_display**(**moreargs*, ***morekwargs*)

PartyLocation.**objects** = **<django.db.models.manager.Manager object at 0x509fa50>**

PartyLocation.**party**

6.4 PartyDescription Objects

class tardis.apps.hpctardis.models.**PartyDescription**(**args*, ***kwargs*)

A description for a party

Attribute type ANDS designation of type of description

Attribute value the value for the description

Attribute party the party that the location is associated with

exception DoesNotExist

exception PartyDescription.**MultipleObjectsReturned**

PartyDescription.**get_type_display**(**moreargs*, ***morekwargs*)

PartyDescription.**objects** = **<django.db.models.manager.Manager object at 0x51f82d0>**

PartyDescription.**party**

6.5 ActivityRecord Objects

class tardis.apps.hpctardis.models.**ActivityRecord**(**args*, ***kwargs*)

A record for an ANDS activity

Attribute key the full ANDS URI identifier

Attribute indent an additional ANDS URI identifier

Attribute type the designation of the type of party

Attribute activityname the nameparts representation of the activity name

Attribute description deprecated, do not use.

Attribute parties the associated parties for this activity

Attribute subject comma delimited list of subject names or codes

Attribute group the origin of the activity

exception `DoesNotExist`

exception `ActivityRecord.MultipleObjectsReturned`

`ActivityRecord.activitydescription_set`

`ActivityRecord.activitylocation_set`

`ActivityRecord.activityname`

`ActivityRecord.activitypartyrelation_set`

`ActivityRecord.get_type_display`(**moreargs*, ***morekwargs*)

`ActivityRecord.objects` = **<django.db.models.manager.Manager object at 0x51f8c50>**

`ActivityRecord.parties`

`ActivityRecord.publishauthorisation_set`

6.6 `ActivityDescription` Objects

class `tardis.apps.hpctardis.models.ActivityDescription`(**args*, ***kwargs*)

A description for a activity

Attribute type ANDS designation of type of description

Attribute value the value for the description

Attribute party the party that the location is associated with

exception `DoesNotExist`

exception `ActivityDescription.MultipleObjectsReturned`

`ActivityDescription.get_type_display`(**moreargs*, ***morekwargs*)

`ActivityDescription.objects` = **<django.db.models.manager.Manager object at 0x51f7110>**

`ActivityDescription.party`

6.7 `ActivityLocation` Objects

class `tardis.apps.hpctardis.models.ActivityLocation`(**args*, ***kwargs*)

A location for a activity

Attribute type ANDS designation of type of location

Attribute value the value for the location

Attribute activity the activity that the location is associated with

exception **DoesNotExist**

exception ActivityLocation.**MultipleObjectsReturned**

ActivityLocation.**activity**

ActivityLocation.**objects** = **<django.db.models.manager.Manager object at 0x51f75d0>**

6.8 ActivityPartyRelation Objects

class tardis.apps.hpctardis.models.**ActivityPartyRelation**(**args*, ***kwargs*)

The relation between an activity and a party :attribute activity: the source :attribute party: the destination :attribute relation: the relationship between the above

exception **DoesNotExist**

exception ActivityPartyRelation.**MultipleObjectsReturned**

ActivityPartyRelation.**activity**

ActivityPartyRelation.**get_relation_display**(**moreargs*, ***morekwargs*)

ActivityPartyRelation.**objects** = **<django.db.models.manager.Manager object at 0x51f7c50>**

ActivityPartyRelation.**party**

6.9 PublishAuthorisation Objects

class tardis.apps.hpctardis.models.**PublishAuthorisation**(**args*, ***kwargs*)

Information of authorisation of collection to authoriser party

Attribute auth_key the crypto key used for identifying authorisers replies

Attribute experiment the collection to be authorised

Attribute authoriser the full name of the authoriser

Attribute email the email address of the authoriser

Attribute status the state of the experiment publication

Attribute party_record the authorising party

Attribute activity_record the activity the authoriser represents

APPROVED_PUBLIC = **2**

exception **DoesNotExist**

PublishAuthorisation.**EXPIRED** = **3**

exception PublishAuthorisation.**MultipleObjectsReturned**

PublishAuthorisation.**PENDING_APPROVAL** = **1**

PublishAuthorisation.**PRIVATE** = **0**

PublishAuthorisation.**activity_record**

PublishAuthorisation.**experiment**

PublishAuthorisation.**get_status_display**(**moreargs*, ***morekwargs*)

PublishAuthorisation.**objects** = **<django.db.models.manager.Manager object at 0x52545d0>**

`PublishAuthorisation.`**`party_record`**

`PublishAuthorisation.`**`publishauthevent_set`**

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*

[JCDA] Thanks to Jeff Croft for the diagram.

t