DAX Documentation

Release 2.11.1-dev0

Benjamin Yvernault, Brian Boyd, Stephen Damon, Andrew Plassa

Sep 19, 2023

CONTENTS

DAX is Distributed Automation for XNAT

DAX allows you to:

- store analyzed imaging data on XNAT (datatypes)
- extract information from XNAT via scripts (Xnat_tools)
- run pipelines on your data in XNAT via a cluster (processors)

CHAPTER ONE

QUICK INSTALL

Create a python3 virtual environment with dax and all dependencies.

```
python3 -m venv daxvenv
source daxvenv/bin/activate
pip install dax
```

Configure an environment variable named XNAT_HOST set to the full url of your xnat server. This can be incuded in your .bashrc/.bash_profile file.

```
export XNAT_HOST=https://central.xnat.org
```

Configure your credentials in a file named ".netrc" in your home directory.

```
machine <SERVER>
login <USER>
password <PASSWORD>
```

Here SERVER is the server name only. For example, central.xnat.org, not https://xnat.website.com/xnat.

CHAPTER

TWO

VERSIONS AND INSTALLATION

Our currently running versions of dax are:

- Dax 2 - 2.2.1 - As of July 8, 2021
  - Used for most purposes
- LDAX latest - 0.7.10 - As of October 7, 2020
  - Legacy Dax - Please use DAX 2

These can be verified with

```
dax version
# or
pip freeze | grep dax
# or
python3 -m pip freeze | grep dax
```

To install please reference our Install Page

Contents:

2.1 Installing DAX in a Virtual Environment

2.1.1 Table of Contents

Setup

To install miniconda3 go to https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html . Follow the procedure described on the miniconda site to install for your OS. It is very important that you follow the directions closely and not forget to source conda. The main idea is to download the Python 3.7 or newer bash file and open the terminal (using 3.8 and MacOS as an example here). Run the following where the file was downloaded:

```
bash Miniconda3-latest-MacOSX-x86_64.sh
```

Follow the prompts until miniconda is installed. Now, source conda and add the path to .bash_profile. Then close and reopen terminal. To display a list of installed packages:

```
conda list
```

Create the Virtual Environment

DAX is to be installed only on virtual environments on Python 3. To create a new environment in Miniconda with Python 3.8:

```
conda create -n daxvenv python=3.8
```

which can then be activated or deactivated with:

```
conda activate daxvenv    # Activation of environment
conda deactivate          # Deactivation of environment
```

After activating the new environment, git version 2.11+ should be installed.

- For ACCRE users, refer to the instructions here: https://dax.readthedocs.io/en/latest/requirements_for_dax_on_accre.html
- Otherwise, install git using these instructions: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Install DAX

Once the virtual environment with Python 3 is created and the correct version of git is installed, you'll need to install dax itself

```
(daxvenv) $ pip install dax
```

Configure an environment variable named XNAT_HOST set to the full url of your xnat server. This can be incuded in your startup file (such as .bashrc or .bash_profile).

```
(daxvenv) $ export XNAT_HOST=https://central.xnat.org
```

Configure your credentials in a file named ".netrc" in your home directory.

```
machine <SERVER>
login <USER>
password <PASSWORD>
```

Here SERVER is the server name only. For example, central.xnat.org, not https://xnat.website.com/xnat. Make sure that the xnat_host is formatted similarly to 'xnat.website.com' NOT 'https://xnat.website.com/xnat' The full url WILL NOT WORK properly.

Verify Installation

Next, run XnatCheckLogin, which will verify that you can log on successfully.

```
(daxvenv) $ XnatCheckLogin
================================================================
Checking your settings for XNAT
No host specified, using XNAT_HOST environment variable.
Checking login for host=https://central.xnat.org
Checking connection:host=https://central.xnat.org, user=someusername
--> Good login.
================================================================
```

2.2 Installation of fs:fsData and proc:genProcData

Prerequisites:

- install an XNAT instance https://wiki.xnat.org/documentation/getting-started-with-xnat

2.2.1 On XNAT VM:

1) Make a BACKUP of your $XNAT_HOME, postgres db, and tomcat deployment
2) Stop tomcat
3) Copy plugins to XNAT

Copy the files dax-plugin-fsData-X.Y.Z.jar and dax-plugin-genProcData-X.Y.Z.jar to ${XNAT_HOME}/plugins

The plugins folder is located in the dax package at the path dax/misc/xnat-plugins/files. You can download the files from github repository: https://github.com/VUIIS/dax .

4) Start tomcat and confirm that plugins are installed

2.2.2 ON XNAT webapp:

1) Log onto XNAT as admin
2) click Administer > Data types
3) click Setup Additional Data Type
4) for fs:fsData (NOTE: the fs:fsData datatype is deprecated. Install only if the need is known to exist)

4.a) select fs:fsData and valid without adding anything at first.

4.b) Come back to the new types and edit the fields:

```
enter "FreeSurfer" in both Singular Name and Plural Name field
enter "FS" in Code field
```

4.c) Edit the "Available Report Actions" by adding delete if you want to be able to delete assessor with the following values:

```
Remove Name: delete
Display Name: Delete
Grouping:
Image: delete.gif
Popup:
Secure Access: delete
Feature:
Additional Parameters:
Sequence: 4
```

4.d) click submit and then accept defaults for subsequent screens

5) for proc:genProcData

5.a) select proc:genProcData and valid without adding anything at first.

5.b) Come back to the new types and edit the fields:

```
enter "Processing" in both Singular Name and Plural Name field
enter "Proc" in Code field
```

5.c) Edit the "Available Report Actions" by adding delete if you want to be able to delete assessor with the following values:

```
Remove Name: delete
Display Name: Delete
Grouping:
Image: delete.gif
Popup:
Secure Access: delete
Feature:
Additional Parameters:
Sequence: 4
```

5.d) click submit and then accept defaults for subsequent screens

You are now ready to use the two assessor types fs:fsData and proc:genProcData

2.3 Source Documentation

2.3.1 `dax` – Root package

2.3.2 `dax.task` – Task class

Task object to generate / manage assessors and cluster.

class dax.task.**ClusterTask**(*assr_label*, *upload_dir*, *diskq*)

Class Task to generate/manage the assessor with the cluster

batch_path()

Method to return the path of the PBS file for the job

Returns
A string that is the absolute path to the PBS file that will be submitted to the scheduler for execution.

`build_commands()`

Call the get_cmds method of the class Processor.

Parameters
`jobdir` – Fully qualified path where the job will run on the node. Note that this is likely to start with /tmp on most grids.

Returns
A string that makes a command line call to a spider with all args.

`build_task()`

Method to build a job

`check_date()`

Sets the job created date if the assessor was not made via dax_build

`check_job_usage()`

The task has now finished, get the amount of memory used, the amount of
walltime used, the jobid of the process, the node the process ran on, and when it started from the scheduler. Set these values locally

Returns
None

`check_running()`

Check to see if a job specified by the scheduler ID is still running

Parameters
`jobid` – The ID of the job in question assigned by the scheduler.

Returns
A String of JOB_RUNNING if the job is running or enqueued and JOB_FAILED if the ready flag (see read_flag_exists) does not exist in the assessor label folder in the upload directory.

`commands(`*jobdir*`)`

Call the get_cmds method of the class Processor.

Parameters
`jobdir` – Fully qualified path where the job will run on the node. Note that this is likely to start with /tmp on most grids.

Returns
A string that makes a command line call to a spider with all args.

`get_createdate()`

Get the date an assessor was created

Returns
String of the date the assessor was created in "%Y-%m-%d" format

`get_job_status()`

Get the status of a job given its jobid as assigned by the scheduler

Parameters
`jobid` – job id assigned by the scheduler

Returns
string from call to cluster.job_status or UNKNOWN.

`get_job_usage()`

Get the amount of memory used, the amount of walltime used, the jobid
of the process, the node the process ran on, and when it started from the scheduler.

Returns
List of strings. Memory used, walltime used, jobid, node used, and start date

`get_jobid()`
Get the jobid of an assessor as stored in local cache

Returns
string of the jobid

`get_jobnode()`
Gets the node that a process ran on

Returns
String identifying the node that a job ran on

`get_jobstartdate()`
Get the date that the job started

Returns
String of the date that the job started in "%Y-%m-%d" format

`get_memused()`
Get the amount of memory used for a process

Returns
String of how much memory was used

`get_processor_name()`
Get the name of the Processor for the Task.

Returns
String of the Processor name.

`get_processor_version()`
Get the version of the Processor.

Returns
String of the Processor version.

`get_qcstatus()`
Get the qcstatus

`get_status()`
Get the procstatus

Returns
The string of the procstatus

`get_statuses()`
Get the procstatus, qcstatus, and job id of an assessor

`get_walltime()`
Get the amount of walltime used for a process

Returns
String of how much walltime was used for a process

`is_open()`

Check to see if a task is still in "Open" status as defined in
OPEN_STATUS_LIST.

Returns
True if the Task is open. False if it is not open

`launch(`*force_no_qsub=False*`)`

Method to launch a job on the grid

Raises
cluster.ClusterLaunchException if the jobid is 0 or empty as returned by pbs.submit() method

Returns
True if the job failed

`outlog_path()`

Method to return the path of outlog file for the job

Returns
A string that is the absolute path to the OUTLOG file.

`processor_spec_path()`

Method to return the path of processor file for the job

Returns
A string that is the absolute path to the file.

`reproc_processing()`

Raises
NotImplementedError

Returns
None

`set_createdate(`*date_str*`)`

Set the date of the assessor creation to user passed value

Parameters
`date_str` – String of the date in "%Y-%m-%d" format

Returns
String of today's date in "%Y-%m-%d" format

`set_createdate_today()`

Set the date of the assessor creation to today

Returns
String of todays date in "%Y-%m-%d" format

`set_jobid(`*jobid*`)`

Set the job ID of the assessor

Parameters
`jobid` – The ID of the process assigned by the grid scheduler

Returns
None

`set_jobnode`(*jobnode*)

Set the value of the the node that the process ran on on the grid

Parameters
`jobnode` – String identifying the node the job ran on

Returns
None

`set_jobstartdate`(*date_str*)

Set the date that the job started on the grid based on user passed
value

Parameters
`date_str` – Datestring in the format "%Y-%m-%d" to set the job starte date to

Returns
None

`set_launch`(*jobid*)

Set the date that the job started and its associated ID. Additionally, set the procstatus to JOB_RUNNING

Parameters
`jobid` – The ID of the process assigned by the grid scheduler

Returns
None

`set_memused`(*memused*)

Set the amount of memory used for a process

Parameters
`memused` – String denoting the amount of memory used

Returns
None

`set_proc_and_qc_status`(*procstatus*, *qcstatus*)

Set the procstatus and qcstatus of the assessor

`set_qcstatus`(*qcstatus*)

Set the qcstatus of the assessor

Parameters
`qcstatus` – String to set the qcstatus to

Returns
None

`set_status`(*status*)

Set the procstatus of an assessor on XNAT

Parameters
`status` – String to set the procstatus of the assessor to

Returns
None

`set_walltime`(*walltime*)

Set the value of walltime used for an assessor

Parameters
`walltime` – String denoting how much time was used running the process.

Returns
None

`undo_processing`()

Unset the job ID, memory used, walltime, and jobnode information
for the assessor on XNAT

Except
pyxnat.core.errors.DatabaseError when attempting to delete a resource

Returns
None

`update_status`()

Update the status of a Cluster Task object.

Returns
the “new” status (updated) of the Task.

`upload_outlog_dir`()

Method to return the path of outlog file for the job

Returns
A string that is the absolute path to the OUTLOG file.

`upload_pbs_dir`()

Method to return the path of dir for the PBS

Returns
A string that is the directory path for the PBS dir

`class dax.task.Task`(*processor*, *assessor*, *upload_dir*)

Class Task to generate/manage the assessor with the cluster

`check_date`()

Sets the job created date if the assessor was not made through
dax_build

Returns
Returns if get_createdate() is != ‘’, sets date otherwise

`check_job_usage`()

The task has now finished, get the amount of memory used, the amount of
walltime used, the jobid of the process, the node the process ran on, and when it started from the scheduler. Set these values on XNAT

Returns
None

`check_running`(*jobid=None*)

Check to see if a job specified by the scheduler ID is still running

Parameters
`jobid` – The ID of the job in question assigned by the scheduler.

Returns
A String of JOB_RUNNING if the job is running or enqueued and JOB_FAILED if the ready flag (see read_flag_exists) does not exist in the assessor label folder in the upload directory.

`commands`(*jobdir*)

Call the get_cmds method of the class Processor.

Parameters
`jobdir` – Fully qualified path where the job will run on the node. Note that this is likely to start with /tmp on most grids.

Returns
A string that makes a command line call to a spider with all args.

`get_createdate`()

Get the date an assessor was created

Returns
String of the date the assessor was created in "%Y-%m-%d" format

`get_job_status`(*jobid=None*)

Get the status of a job given its jobid as assigned by the scheduler

Parameters
`jobid` – job id assigned by the scheduler

Returns
string from call to cluster.job_status or UNKNOWN.

`get_job_usage`()

Get the amount of memory used, the amount of walltime used, the jobid
of the process, the node the process ran on, and when it started from the scheduler.

Returns
List of strings. Memory used, walltime used, jobid, node used, and start date

`get_jobid`()

Get the jobid of an assessor as stored on XNAT

Returns
string of the jobid

`get_jobnode`()

Gets the node that a process ran on

Returns
String identifying the node that a job ran on

`get_jobstartdate`()

Get the date that the job started

Returns
String of the date that the job started in "%Y-%m-%d" format

`get_memused()`

Get the amount of memory used for a process

Returns
String of how much memory was used

`get_processor_name()`

Get the name of the Processor for the Task.

Returns
String of the Processor name.

`get_processor_version()`

Get the version of the Processor.

Returns
String of the Processor version.

`get_qcstatus()`

Get the qcstatus of the assessor

Returns
A string of the qcstatus for the assessor if it exists. If it does not, it returns DOES_NOT_EXIST. The else case returns an UNKNOWN xsiType with the xsiType of the assessor as stored on XNAT.

`get_status()`

Get the procstatus of an assessor

Returns
The string of the procstatus of the assessor. DOES_NOT_EXIST if the assessor does not exist

`get_statuses`(*cached_sessions=None*)

Get the procstatus, qcstatus, and job id of an assessor

Returns
Serially ordered strings of the assessor procstatus, qcstatus, then jobid.

`get_walltime()`

Get the amount of walltime used for a process

Returns
String of how much walltime was used for a process

`is_open()`

Check to see if a task is still in "Open" status as defined in
OPEN_STATUS_LIST.

Returns
True if the Task is open. False if it is not open

`launch`(*jobdir*, *job_email=None*, *job_email_options='FAIL'*, *job_rungroup=None*, *xnat_host=None*, *writeonly=False*, *pbsdir=None*, *force_no_qsub=False*)

Method to launch a job on the grid

Parameters

- **`jobdir`** – absolute path where the data will be stored on the node
- **`job_email`** – who to email if the job fails

- `job_email_options` – grid-specific job email options (e.g., fails, starts, exits etc)
- `job_rungroup` – grid-specific group to run the job under
- `xnat_host` – set the XNAT_HOST in the PBS job
- `writeonly` – write the job files without submitting them
- `pbsdir` – folder to store the pbs file
- `force_no_qsub` – run the job locally on the computer (serial mode)

Raises
cluster.ClusterLaunchException if the jobid is 0 or empty as returned by pbs.submit() method

Returns
True if the job failed

`outlog_path()`

Method to return the path of outlog file for the job

Returns
A string that is the absolute path to the OUTLOG file.

`pbs_path`(*writeonly=False*, *pbsdir=None*)

Method to return the path of the PBS file for the job

Parameters

- `writeonly` – write the job files without submitting them in TRASH
- `pbsdir` – folder to store the pbs file

Returns
A string that is the absolute path to the PBS file that will be submitted to the scheduler for execution.

`ready_flag_exists()`

Method to see if the flag file <UPLOAD_DIR>/<ASSESSOR_LABEL>/READY_TO_UPLOAD.txt exists

Returns
True if the file exists. False if the file does not exist.

`reproc_processing()`

If the procstatus of an assessor is REPROC on XNAT, rerun the assessor.

Returns
None

`set_createdate`(*date_str*)

Set the date of the assessor creation to user passed value

Parameters
`date_str` – String of the date in "%Y-%m-%d" format

Returns
String of today's date in "%Y-%m-%d" format

`set_createdate_today()`

Set the date of the assessor creation to today

Returns
String of todays date in "%Y-%m-%d" format

`set_jobid`(*jobid*)

Set the job ID of the assessor on XNAT

Parameters
`jobid` – The ID of the process assigned by the grid scheduler

Returns
None

`set_jobnode`(*jobnode*)

Set the value of the the node that the process ran on on the grid

Parameters
`jobnode` – String identifying the node the job ran on

Returns
None

`set_jobstartdate`(*date_str*)

Set the date that the job started on the grid based on user passed
value

Parameters
`date_str` – Datestring in the format "%Y-%m-%d" to set the job starte date to

Returns
None

`set_jobstartdate_today`()

Set the date that the job started on the grid to today

Returns
call to set_jobstartdate with today's date

`set_launch`(*jobid*)

Set the date that the job started and its associated ID on XNAT. Additionally, set the procstatus to JOB_RUNNING

Parameters
`jobid` – The ID of the process assigned by the grid scheduler

Returns
None

`set_memused`(*memused*)

Set the amount of memory used for a process

Parameters
`memused` – String denoting the amount of memory used

Returns
None

`set_proc_and_qc_status`(*procstatus*, *qcstatus*)

Set the procstatus and qcstatus of the assessor

Parameters

- `procstatus` – String to set the procstatus of the assessor to
- `qcstatus` – String to set the qcstatus of the assessor to

Returns
None

`set_qcstatus`(*qcstatus*)

Set the qcstatus of the assessor

Parameters
`qcstatus` – String to set the qcstatus to

Returns
None

`set_status`(*status*)

Set the procstatus of an assessor on XNAT

Parameters
`status` – String to set the procstatus of the assessor to

Returns
None

`set_walltime`(*walltime*)

Set the value of walltime used for an assessor on XNAT

Parameters
`walltime` – String denoting how much time was used running the process.

Returns
None

`undo_processing`()

Unset the job ID, memory used, walltime, and jobnode information
for the assessor on XNAT

Except
pyxnat.core.errors.DatabaseError when attempting to delete a resource

Returns
None

`update_status`()

Update the satus of a Task object.

Returns
the "new" status (updated) of the Task.

`class dax.task.XnatTask`(*processor*, *assessor*, *upload_dir*, *diskq*)

Class Task to generate/manage the assessor with the cluster

`batch_path`()

Method to return the path of the PBS file for the job

Returns
A string that is the absolute path to the PBS file that will be submitted to the scheduler for execution.

`build_commands`(*assr*, *sessions*, *jobdir*, *resdir*)

Call the build_cmds method of the class Processor.

Parameters
`jobdir` – Fully qualified path where the job will run on the node. Note that this is likely to start with /tmp on most grids.

Returns
A string that makes a command line call to a spider with all args.

`build_task`(*assr*, *sessions*, *jobdir*, *job_email=None*, *job_email_options='FAIL'*, *job_rungroup=None*, *xnat_host=None*)

Method to build a job

`check_job_usage`()

The task has now finished, get the amount of memory used, the amount of
walltime used, the jobid of the process, the node the process ran on, and when it started from the scheduler. Set these values on XNAT

Returns
None

`check_running`()

Check to see if a job specified by the scheduler ID is still running

Parameters
`jobid` – The ID of the job in question assigned by the scheduler.

Returns
A String of JOB_RUNNING if the job is running or enqueued and JOB_FAILED if the ready flag (see read_flag_exists) does not exist in the assessor label folder in the upload directory.

`get_job_status`()

Get the status of a job given its jobid as assigned by the scheduler

Parameters
`jobid` – job id assigned by the scheduler

Returns
string from call to cluster.job_status or UNKNOWN.

`launch`()

Method to launch a job on the grid

`outlog_path`()

Method to return the path of outlog file for the job

Returns
A string that is the absolute path to the OUTLOG file.

`processor_spec_path`()

Method to return the path of processor file for the job

Returns
A string that is the absolute path to the file.

`set_launch`(*jobid*)

Set the date that the job started and its associated ID on XNAT. Additionally, set the procstatus to JOB_RUNNING

Parameters
`jobid` – The ID of the process assigned by the grid scheduler

Returns
None

update_status()

Update the satus of an XNAT Task object.

Returns
the "new" status (updated) of the Task.

2.3.3 `dax.spiders` – Spider class

2.3.4 `dax.processors` – Processor class

Processor class define for Scan and Session.

class dax.processors.**AutoProcessor**(*xnat*, *yaml_source*, *user_inputs=None*)

Auto Processor class for AutoSpider using YAML files

get_assessor_input_types()

Enumerate the assessor input types for this. The default implementation returns an empty collection; override this method if you are inheriting from a non-yaml processor. :return: a list of input assessor types

get_cmds(*assr*, *jobdir*)

Method to generate the spider command for cluster job.

Parameters

- **assessor** – pyxnat assessor object
- **jobdir** – jobdir where the job's output will be generated

Returns
command to execute the spider in the job script

get_proctype()

Return the processor name for this processor. Override this method if you are inheriting from a non-yaml processor. :return: the name of the processor type

parse_session(*csess*, *sessions*, *pets=None*)

Method to run the processor parser on this session, in order to calculate the pattern matches for this processor and the sessions provided :param csess: the active session. For non-longitudinal studies, this is the session that the pattern matching is performed on. For longitudinal studies, this is the 'current' session from which all prior sessions are numbered for the purposes of pattern matching :param sessions: the full, time-ordered list of sessions that should be considered for longitudinal studies. :return: None

class dax.processors.**Processor**(*walltime_str*, *memreq_mb*, *spider_path*, *version=None*, *ppn=1*, *env=None*, *suffix_proc=''*, *xsitype='proc:genProcData'*, *job_template=None*)

Base class for processor

build_cmds(*cobj*, *dir*)

Build the commands that will go in the PBS/SLURM script :raises: NotImplementedError if not overridden from base class. :return: None

default_settings_spider(*spider_path*)

Get the default spider version and name

Parameters
spider_path – Fully qualified path and file of the spider

Returns
None

`get_assessor_input_types()`

Enumerate the assessor input types for this. The default implementation returns an empty collection; override this method if you are inheriting from a non-yaml processor. :return: a list of input assessor types

`get_proctype()`

Return the processor name for this processor. Override this method if you are inheriting from a non-yaml processor. :return: the name of the processor type

`set_spider_settings`(*spider_path*, *version*)

Method to set the spider version, path, and name from filepath

Parameters

- **`spider_path`** – Fully qualified path and file of the spider
- **`version`** – version of the spider

Returns
None

`should_run()`

Responsible for determining if the assessor should shouw up in session.

Raises
NotImplementedError if not overridden.

Returns
None

2.3.5 `dax.log` – Logging utility

`dax.log.setup_critical_logger`(*name*, *logfile*)

Sets up the critical logger

Parameters

- **`name`** – Name of the logger
- **`logfile`** – file to store the log to. sys.stdout if no file define

Returns
logger object

`dax.log.setup_debug_logger`(*name*, *logfile*)

Sets up the debug logger

Parameters

- **`name`** – Name of the logger
- **`logfile`** – file to store the log to. sys.stdout if no file define

Returns
logger object

`dax.log.setup_error_logger`(*name*, *logfile*)

Sets up the error logger

Parameters

- **name** – Name of the logger
- **logfile** – file to store the log to. sys.stdout if no file define

Returns
logger object

`dax.log.`**`setup_info_logger`**`(`*name*, *logfile*`)`

Sets up the info logger

Parameters

- **name** – Name of the logger
- **logfile** – file to store the log to. sys.stdout if no file define

Returns
logger object

`dax.log.`**`setup_warning_logger`**`(`*name*, *logfile*`)`

Sets up the warning logger

Parameters

- **name** – Name of the logger
- **logfile** – file to store the log to. sys.stdout if no file define

Returns
logger object

2.3.6 dax.bin – Responsible for launching, building and updating a Task

File containing functions called by dax executables

`dax.bin.`**`build`**`(`*settings_path*, *logfile*, *debug*, *projects=None*, *sessions=None*, *mod_delta=None*, *proj_lastrun=None*, *start_sess=None*`)`

Method that is responsible for running all modules and putting assessors
into the database

Parameters

- **settings_path** – Path to the project settings file
- **logfile** – Full file of the file used to log to
- **debug** – Should debug mode be used
- **projects** – Project(s) that need to be built
- **sessions** – Session(s) that need to be built

Returns
None

`dax.bin.`**`check_default_keys`**`(`*yaml_file*, *doc*`)`

Static method to raise error if key not found in dictionary from yaml file. :param yaml_file: path to yaml file defining the processor :param doc: doc dictionary extracted from the yaml file `dax.bin.`**`launch_jobs`**`(`*settings_path*, *logfile*, *debug*, *projects=None*, *sessions=None*, *writeonly=False*, *pbsdir=None*, *force_no_qsub=False*`)`

Method to launch jobs on the grid

Parameters

- **`settings_path`** – Path to the project settings file
- **`logfile`** – Full file of the file used to log to
- **`debug`** – Should debug mode be used
- **`projects`** – Project(s) that need to be launched
- **`sessions`** – Session(s) that need to be updated
- **`writeonly`** – write the job files without submitting them
- **`pbsdir`** – folder to store the pbs file
- **`force_no_qsub`** – run the job locally on the computer (serial mode)

Returns
None

`dax.bin.`**`load_from_file`**`(`*filepath*, *args*, *logger*, *singularity_imagedir=None*, *job_template=None*`)`

Check if a file exists and if it's a python file :param filepath: path to the file to test :return: True the file pass the test, False otherwise

`dax.bin.`**`raise_yaml_error_if_no_key`**`(`*doc*, *yaml_file*, *key*`)`

Method to raise an execption if the key is not in the dict :param doc: dict to check :param yaml_file: YAMLfile path :param key: key to search

`dax.bin.`**`read_yaml_settings`**`(`*yaml_file*, *logger*`)`

Method to read the settings yaml file and generate the launcher object.

Parameters
`yaml_file` – path to yaml file defining the settings

Returns
launcher object

`dax.bin.`**`set_logger`**`(`*logfile*, *debug*`)`

Set the logging depth

Parameters

- **`logfile`** – File to log output to
- **`debug`** – Should debug depth be used?

Returns
logger object

`dax.bin.`**`undo_processing`**`(`*assessor_label*, *logger=None*`)`

Unset job information for the assessor on XNAT, Delete files, set to run.

Returns
None

`dax.bin.`**`update_tasks`**`(`*settings_path*, *logfile*, *debug*, *projects=None*, *sessions=None*`)`

Method that is responsible for updating a Task.

Parameters

- **settings_path** – Path to the project settings file
- **logfile** – Full file of the file used to log to
- **debug** – Should debug mode be used
- **projects** – Project(s) that need to be launched
- **sessions** – Session(s) that need to be updated

Returns
None

2.3.7 `dax.XnatUtils` – Collection of utilities for upload/download and general access

XnatUtils contains functions to interface with XNAT using Pyxnat.

`class dax.XnatUtils.AssessorHandler(`*label*`)`

Class to intelligently deal with the Assessor labels. Make the splitting of the strings easier.

`get_proctype()`

Get the proctype from the assessor label

Returns
The proctype for the assessor

`get_project_id()`

Get the project ID from the assessor label

Returns
The XNAT project label

`get_scan_id()`

Get the scan ID from teh assessor label

Returns
The scan id for the assessor label

`get_session_label()`

Get the session label from the assessor label

Returns
The XNAT session label

`get_subject_label()`

Get the subject label from the assessor label

Returns
The XNAT subject label

`is_valid()`

Check to see if we have a valid assessor label (aka not None)

Returns
True if valid, False if not valid

`select_assessor(`*intf***`)`**

Run Interface.select() on the assessor label

Parameters
`intf` – pyxnat.Interface object

Returns
The pyxnat EObject of the assessor

`class dax.XnatUtils.CachedImageAssessor(`*intf*, *assr_element*, *parent*`)`

Class to cache the XML information for an assessor on XNAT

`get(`*name*`)`

Get the value of a variable associated with the assessor

Parameters
`name` – Variable name to get the value of

Returns
Value of the variable, otherwise ‘’.

`get_in_resources()`

Get a list of dictionaries of info for the CachedResource objects
for “in” type

Returns
List of dictionaries of info for the CachedResource objects for “in” type

`get_out_resources()`

Get a list of dictionaries of info for the CachedResource objects
for “out” type

Returns
List of dictionaries of info for the CachedResource objects for “out” type

`get_resources()`

Makes a call to get_out_resources.

Returns
List of dictionaries of info for the CachedResource objects for “out” type

`in_resources()`

Get a list of CachedResource objects for “in” type

Returns
List of CachedResource objects for “in” type

`info()`

Get a dictionary of information associated with the assessor

Returns
None

`label()`

Get the label of the assessor

Returns
String of the assessor label

`out_resources()`

Get a list of CachedResource objects for “out” type

Returns
List of CachedResource objects for “out” type

`parent()`

Get the parent element of the assessor (session)

Returns
The session element XML string

`class dax.XnatUtils.CachedImageScan(`*intf*, *scan_element*, *parent*`)`

Class to cache the XML information for a scan on XNAT

`get(`*name*`)`

Get the value of a variable associated with a scan.

Parameters
`name` – Name of the variable to get the value of

Returns
Value of the variable if it exists, or ‘’ otherwise.

`get_resources()`

Get a list of dictionaries of info for each CachedResource.

Returns
List of dictionaries of infor for each CachedResource.

`info()`

Get lots of variables assocaited with this scan.

Returns
Dictionary of infomation about the scan.

`label()`

Get the ID of the scan

Returns
String of the scan ID

`parent()`

Get the parent of the scan

Returns
XML String of the scan parent

`resources()`

Get a list of the CachedResource (s) associated with this scan.

Returns
List of the CachedResource (s) associated with this scan.

`session()`

Get the session associated with this object :return: session asscoiated with this object

`class dax.XnatUtils.CachedImageSession(`*intf*, *proj*, *subj*, *sess*`)`

Enumeration for assessors function, to control what assessors are returned `assessors`(*select=(0,)*)

Get a list of CachedImageAssessor objects for the XNAT session

Returns
List of CachedImageAssessor objects for the session.

`full_object`()

Return a the full pyxnat Session object of this sessions

Returns
pyxnat Session object

`get`(*name*)

Get the value of a variable name in the session

Parameters
`name` – The variable name that you want to get the value of

Returns
The value of the variable or '' if not found.

`get_resources`()

Return a list of dictionaries that correspond to the information
for each resource

Returns
List of dictionaries

`has_shared_project`()

Get the project if shared.

Returns
project_shared_id if shared, None otherwise

`info`()

Get a dictionary of lots of variables that correspond to the session

Returns
Dictionary of variables

`label`()

Get the label of the session

Returns
String of the session label

`resources`()

Get a list of CachedResource objects for the session

Returns
List of CachedResource objects for the session

`scans`()

Get a list of CachedImageScan objects for the XNAT session

Returns
List of CachedImageScan objects for the session.

`session()`

Get the session associated with this object :return: session asscoiated with this object

`class dax.XnatUtils.CachedResource(`*element*, *parent*`)`

Class to cache resource XML info on XNAT

`get(`*name*`)`

Get the value of a variable associated with the resource

Parameters
`name` – Variable name to get the value of

Returns
The value of the variable, ‘’ otherwise.

`info()`

Get a dictionary of information relating to the resource

Returns
dictionary of information about the resource.

`label()`

Get the label of the resource

Returns
String of the label of the resource

`parent()`

Get the resource parent XML string

Returns
The resource parent XML string

`class dax.XnatUtils.InterfaceTemp(`*xnat_host=None*, *xnat_user=None*, *xnat_pass=None*, *smtp_host=None*, *timeout_emails=None*, *xnat_timeout=300*, *xnat_retries=4*, *xnat_wait=600*`)`

Extends the pyxnat.Interface class to make a temporary directory, write the
cache to it and then blow it away on the Interface.disconnect call() NOTE: This is deprecated in pyxnat 1.0.0.0

Using netrc to get username password if not given.

`authenticate()`

Authenticate to XNAT.

Connect to XNAT and try to Disconnect the JSESSION before reconnecting. Raise XnatAuthentification-Error if it failes.

Returns
True or False

`connect()`

Connect to XNAT.

`disconnect()`

Tell XNAT to disconnect this session

`get_assessor_out_resources`(*projectid*, *subjectid*, *sessionid*, *assessorid*)

Gets a list of all of the resources for an assessor associated to a
session/subject/project requested by the user.

Parameters

- `(string)` (*assessorid*) – ID of a project on XNAT
- `(string)` – ID/label of a subject
- `(string)` – ID/label of a session
- `(string)` – ID/label of an assessor to get resources for

Returns
List of resources for the assessor

`get_assessor_path`(*project*, *subject*, *session*, *assessor*)

Given project, subject, session, assessor (strings), returns assessor path (string)

`get_assessor_resource_path`(*project*, *subject*, *session*, *assessor*, *resource*)

Given project, subject, session, assessor, resource (strings), returns assessor resource path (string)

`get_assessors`(*projectid*, *subjectid*, *sessionid*)

List all the assessors that you have access to based on passed
session/subject/project.

Parameters

- `(string)` (*sessionid*) – ID of a project on XNAT
- `(string)` – ID/label of a subject
- `(string)` – ID/label of a session

Returns
List of all the assessors

`get_experiment_path`(*project*, *subject*, *session*)

Given project, subject, session (strings), returns session path (string)

`get_project_assessors`(*projectid*)

List all the assessors that you have access to based on passed project.

Parameters
`(string)` (*projectid*) – ID of a project on XNAT

Returns
List of all the assessors for the project

`get_project_path`(*project*)

Given project (string), returns project path (string)

`get_project_scans`(*project_id*, *include_shared=True*)

List all the scans that you have access to based on passed project.

Parameters

- `(string)` (*projectid*) – ID of a project on XNAT
- `(boolean)` (*include_shared*) – include the shared data in this project

Returns
List of all the scans for the project

`get_resources`(*project_id*)
Given project (string), return list of project's resources

`get_scan_path`(*project*, *subject*, *session*, *scan*)
Given project, subject, session, scan (strings), returns scan path (string)

`get_scan_resource_path`(*project*, *subject*, *session*, *scan*, *resource*)
Given project, subject, session, scan, resource (strings), returns scan resource path (string)

`get_scan_resources`(*projectid*, *subjectid*, *sessionid*, *scanid*)

Gets a list of all of the resources for a scan associated to a
session/subject/project requested by the user.

Parameters

- **(string)** (*scanid*) – ID of a project on XNAT
- **(string)** – ID/label of a subject
- **(string)** – ID/label of a session
- **(string)** – ID of a scan to get resources for

Returns
List of resources for the scan

`get_scans`(*projectid*, *subjectid*, *sessionid*)

List all the scans that you have access to based on passed
session/subject/project.

Parameters

- **(string)** (*sessionid*) – ID of a project on XNAT
- **(string)** – ID/label of a subject
- **(string)** – ID/label of a session

Returns
List of all the scans

`get_session_resources`(*projectid*, *subjectid*, *sessionid*)

Gets a list of all of the resources for a session associated to a
subject/project requested by the user

Parameters

- **(string)** (*sessionid*) – ID of a project on XNAT
- **(string)** – ID/label of a subject
- **(string)** – ID/label of a session to get resources for

Returns
List of resources for the session

`get_sessions`(*projectid=None*, *subjectid=None*)

List all the sessions either:

1) that you have access to

or

2) in a single project (and single subject) based on kargs

Parameters

- `projectid` – ID of a project on XNAT
- `subjectid` – ID/label of a subject

Returns
List of sessions

`get_sessions_minimal`(*projectid*)

Parameters
`projectid` – ID of a project on XNAT

Returns
List of sessions

`get_sgp_assessor_path`(*project*, *subject*, *assessor*)

Given project, subject, assessor (strings), returns assessor path (string)

`get_subject_path`(*project*, *subject*)

Given project, subject (strings), returns subject path (string)

`get_subject_resources`(*project_id*, *subject_id*)

Given project and subject (strings), return list of subject's resources

`get_subjects`(*project_id*)

Given project_id (string), return list of subjects in project

`list_project_assessor_types`(*projectid*)

List all the assessors that you have access to based on passed project.

Parameters
`(string)` (*projectid*) – ID of a project on XNAT

Returns
List of all the assessors for the project

`list_project_assessors`(*projectid*)

List all the assessors that you have access to based on passed project.

Parameters
`(string)` (*projectid*) – ID of a project on XNAT

Returns
List of all the assessors for the project

`select_assessor`(*project*, *subject*, *session*, *assessor*)

Given project, subject, session, assessor (strings), returns assessor object `select_assessor_resource`(*project*, *subject*, *session*, *assessor*, *resource*)

Given project, subject, session, assessor, resource (strings), returns assessor resource object

`select_experiment`(*project*, *subject*, *session*)

Given project, subject, session (strings), returns session (experiment object) Same as select_session

`select_project`(*project*)

Given project (string), returns project object

`select_scan`(*project*, *subject*, *session*, *scan*)

Given project, subject, session, scan (strings), returns scan object

`select_scan_resource`(*project*, *subject*, *session*, *scan*, *resource*)

Given project, subject, session, scan, resource (strings), returns scan resource object

`select_session`(*project*, *subject*, *session*)

Given project, subject, session (strings), returns session (experiment object) Same as select_experiment

`select_sgp_assessor`(*project*, *subject*, *assessor*)

Given project, subject, assessor (strings), returns assessor object

`select_subject`(*project*, *subject*)

Given project, subject (strings), returns subject object

`class dax.XnatUtils.SpiderProcessHandler`(*script_name*, *suffix*, *project=None*, *subject=None*, *experiment=None*, *scan=None*, *alabel=None*, *assessor_handler=None*, *time_writer=None*, *host=None*)

Class to handle the uploading of results for a spider.

`add_file`(*filepath*, *resource*)

Add a file in the assessor in the upload directory based on the
resource name as will be seen on XNAT

Parameters

- `filepath` – Full path to a file to upload
- `resource` – The resource name it should appear under in XNAT

Returns
None

`add_folder`(*folderpath*, *resource_name=None*)

Add a folder to the assessor in the upload directory.

Parameters

- `folderpath` – Full path to the folder to upoad
- `resource_name` – Resource name chosen (if different than basename)

Raises

- `shutil.Error` – Directories are the same
- `OSError` – The directory doesn't exist

Returns
None

`add_pdf`(*filepath*)
Add the PDF and run ps2pdf on the file if it ends with .ps

Parameters
`filepath` – Full path to the PDF/PS file

Returns
None

`add_snapshot`(*snapshot*)
Add in the snapshots (for quick viewing on XNAT)

Parameters
`snapshot` – Full path to the snapshot file

Returns
None

`clean`(*directory*)
Clean directory if no error and pdf created

Parameters
`directory` – directory to be cleaned

`done`()

Create a flag file that the assessor is ready to be uploaded and set
the status as READY_TO_UPLOAD

Returns
None

`file_exists`(*fpath*)
Check to see if a file exists

Parameters
`fpath` – full path to a file to assert it exists

Returns
True if it exists, False if it doesn't

`folder_exists`(*fpath*)
Check to see if a folder exists

Parameters
`fpath` – Full path to a folder to assert it exists

Returns
True if it exists, False if it doesn't

`print_copying_statement`(*label*, *src*, *dest*)
Print a line that data is being copied to the upload directory

Parameters

- `label` – The XNAT resource label
- `src` – Source directory or file
- `dest` – Destination directory or file

Returns
None

`print_err`(*msg*)
Print error message using time writer if set, print otherwise

Parameters
`msg` – Message to print

Returns
None

`print_msg`(*msg*)
Prints a message using TimedWriter or print

Parameters
`msg` – Message to print

Returns
None

`set_assessor_status`(*status*)
Set the status of the assessor based on passed value

Parameters
`status` – Value to set the procstatus to

Except
All catchable errors.

Returns
None

`set_error`()
Set the flag for the error to 1

Returns
None

2.4 DAX Manager

2.4.1 Table of Contents:

About

DAX Manager is a non-required tool hosted in REDCap which allows you to quickly generate settings files that can be launched with DAX. This alleviates the need to manual write settings files and makes updating scan types, walltimes, etc a much quicker and streamlined process.

How to set it up

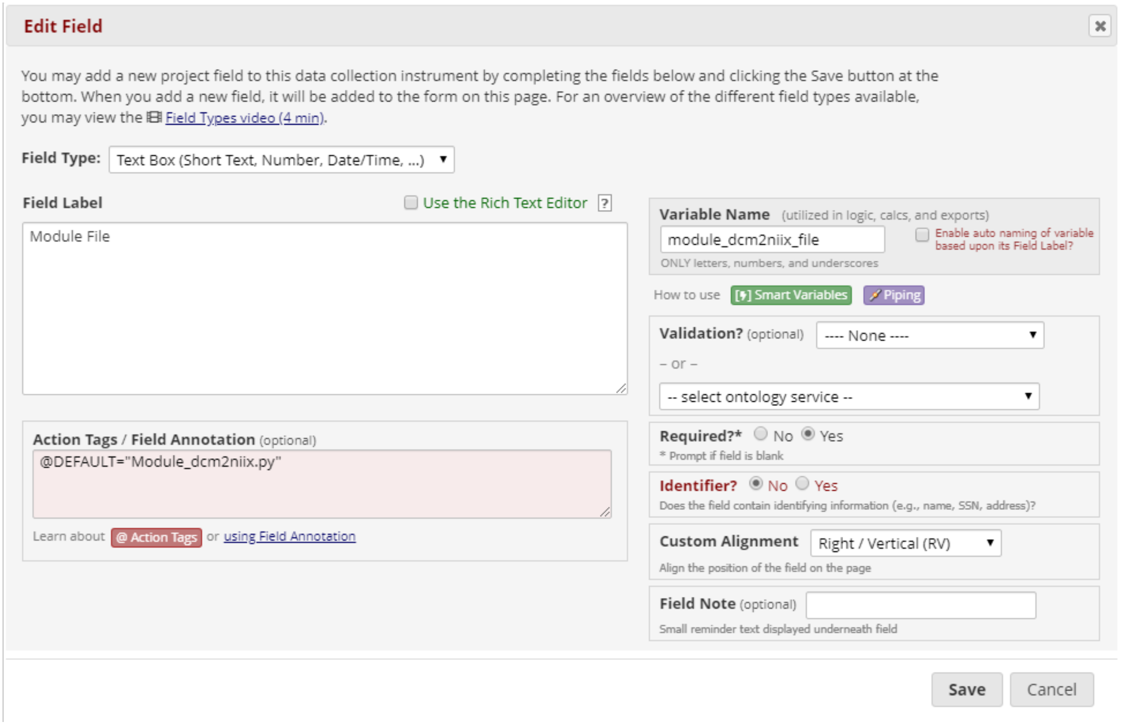
The main instrument should be called General and contains a lot of standard variables that are required for DAX to interface with DAX Manager appropriately. For convenience, a copy of the latest data dictionary has been included and can be downloaded here for reference. It is suggested to use this version even if you do not plan on running all of the spiders because it is currently being used in production

https://github.com/VUIIS/dax/blob/master/docs/files/dax_manager/XNATProjectSettings_DataDictionary_2016-01-21.csv

2.4.2 DAX 1

How to add a Module in DAX 1

Variables used in a module must all start with the FULL module name. For example, consider "Module dcm2niix". All of the variables for this module must start with "**module_dcm2niix_**". There are 2 required variables. The first is the "Module File" variable. This variable for "Module dcm2niix" would be called "module_dcm2niix_file". The "Action Tags / Field Annotation" should be @DEFAULT="MODULE_NAME". See below for an example.


Edit Field

You may add a new project field to this data collection instrument by completing the fields below and clicking the Save button at the bottom. When you add a new field, it will be added to the form on this page. For an overview of the different field types available, you may view the Field Types video (4 min).

Field Type: Text Box (Short Text, Number, Date/Time, ...)

Field Label: Module File

Use the Rich Text Editor

Variable Name (utilized in logic, calcs, and exports): module_dcm2niix_file

Enable auto naming of variable based upon its Field Label?

ONLY letters, numbers, and underscores

How to use Smart Variables Piping

Validation? (optional): ---- None ----

– or –

-- select ontology service --

Action Tags / Field Annotation (optional): @DEFAULT="Module_dcm2niix.py"

Learn about @ Action Tags or using Field Annotation

Required?* No [✓ Yes]

* Prompt if field is blank

Identifier? [✓ No] Yes

Does the field contain identifying information (e.g., name, SSN, address)?

Custom Alignment: Right / Vertical (RV)

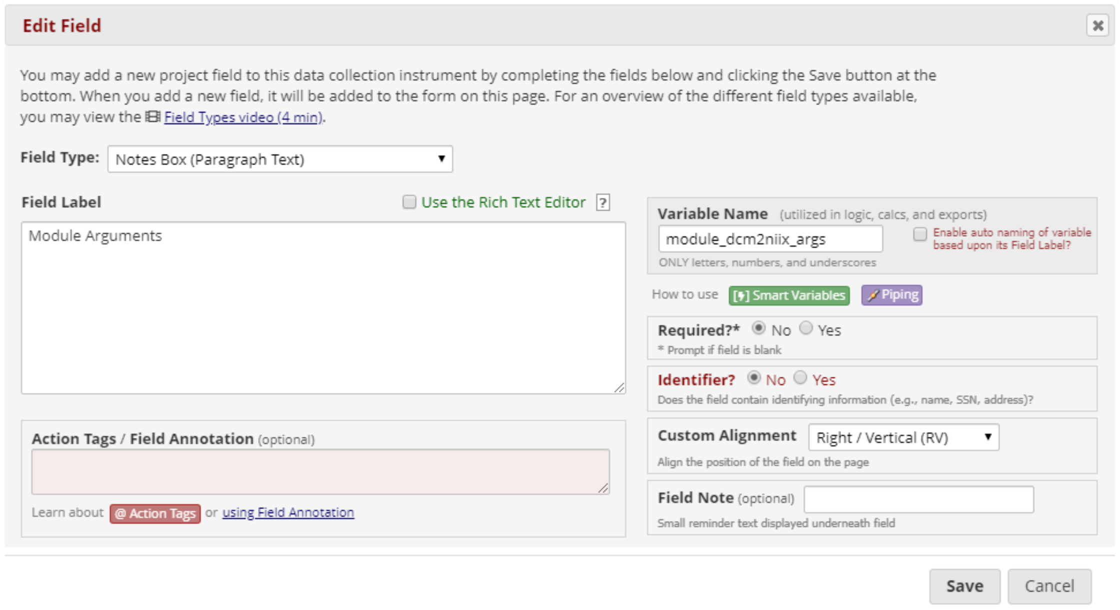
Align the position of the field on the page

Field Note (optional):

Small reminder text displayed underneath field

Save Cancel

The second required variable is the "Module Arguments" variable. In the case of "Module dcm2niix", this variable would be called "module_dcm2niix_args". See below for an example.


Edit Field

You may add a new project field to this data collection instrument by completing the fields below and clicking the Save button at the bottom. When you add a new field, it will be added to the form on this page. For an overview of the different field types available, you may view the Field Types video (4 min).

Field Type: Notes Box (Paragraph Text)

Field Label: Module Arguments

Use the Rich Text Editor

Action Tags / Field Annotation (optional)

Learn about @ Action Tags or using Field Annotation

Variable Name (utilized in logic, calcs, and exports): module_dcm2niix_args

Enable auto naming of variable based upon its Field Label?

ONLY letters, numbers, and underscores

How to use [$] Smart Variables Piping

Required?* [✓ No] Yes

* Prompt if field is blank

Identifier? [✓ No] Yes

Does the field contain identifying information (e.g., name, SSN, address)?

Custom Alignment: Right / Vertical (RV)

Align the position of the field on the page

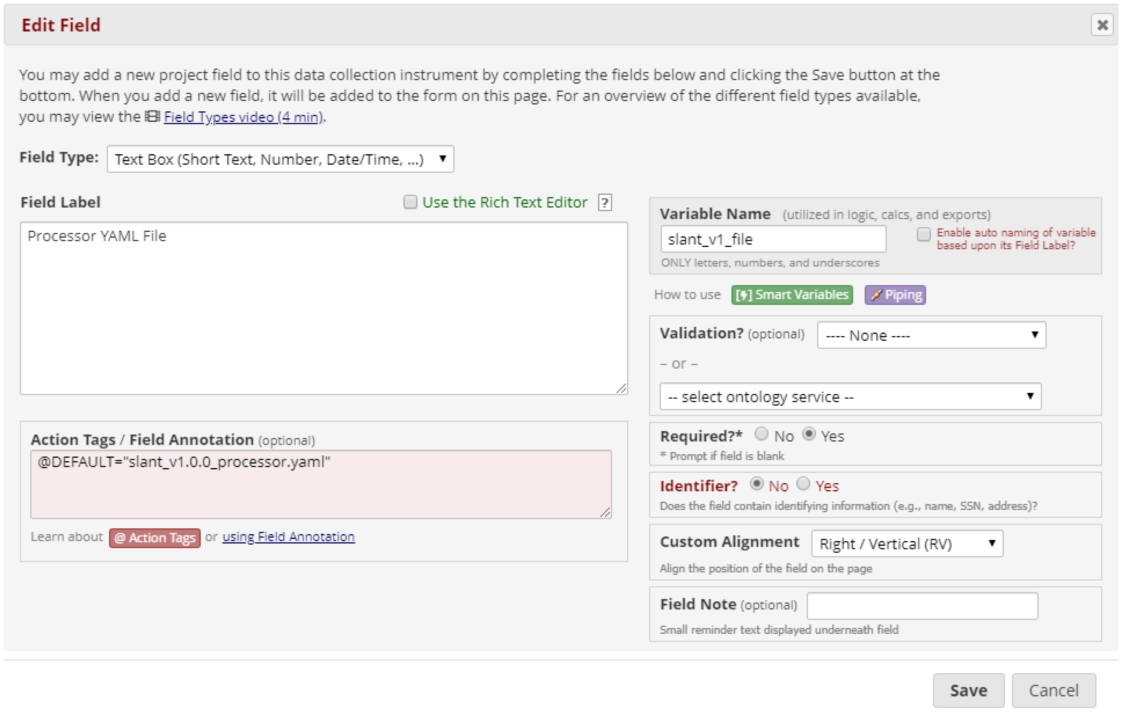
Field Note (optional)

Small reminder text displayed underneath field

Save Cancel

How to add a Process in DAX 1

Processes are setup very similarly to Modules. There are 2 required variables, "Processor YAML File" and "Processor Arguments". The variable names use slighly different naming conventions as Modules. For example, consider "Processor slant_v1". The "Processor YAML File" variable should be named "slant_v1_file" and the "Action Tags / Field Annotation" field should contain the full name of the processor (@DEFAULT="slant_v1.0.0_processor.yaml"). See below for an example.


Edit Field

You may add a new project field to this data collection instrument by completing the fields below and clicking the Save button at the bottom. When you add a new field, it will be added to the form on this page. For an overview of the different field types available, you may view the Field Types video (4 min).

Field Type: Text Box (Short Text, Number, Date/Time, ...)

Field Label: Processor YAML File

Use the Rich Text Editor

Action Tags / Field Annotation (optional): @DEFAULT="slant_v1.0.0_processor.yaml"

Learn about @ Action Tags or using Field Annotation

Variable Name (utilized in logic, calcs, and exports): slant_v1_file

Enable auto naming of variable based upon its Field Label?

ONLY letters, numbers, and underscores

How to use [$] Smart Variables Piping

Validation? (optional): ---- None ----

– or –

-- select ontology service --

Required?* No [✓ Yes]

* Prompt if field is blank

Identifier? [✓ No] Yes

Does the field contain identifying information (e.g., name, SSN, address)?

Custom Alignment: Right / Vertical (RV)

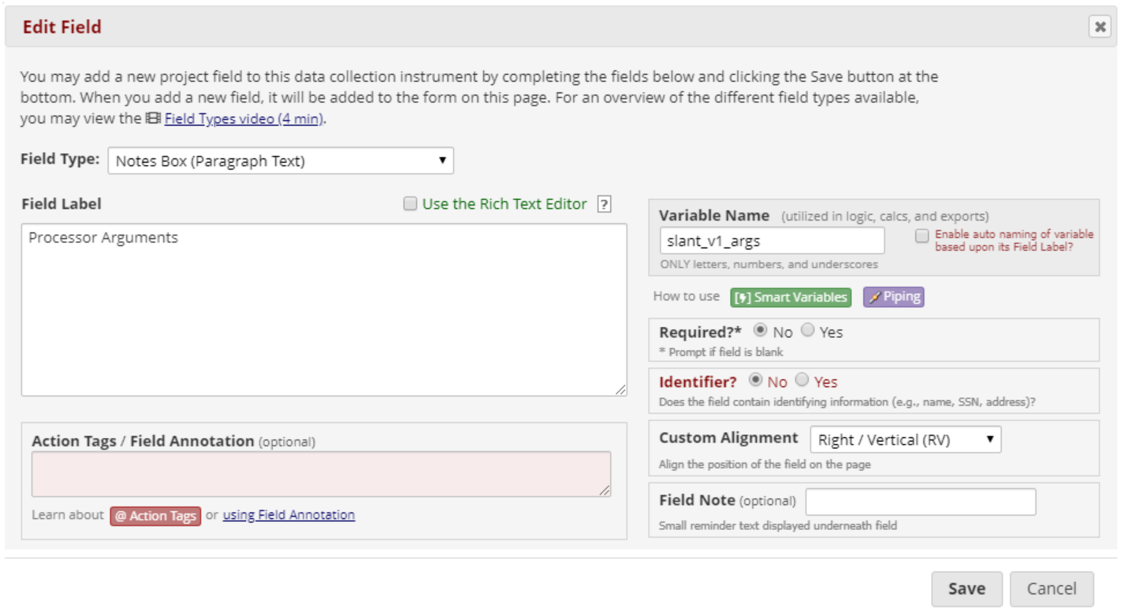
Align the position of the field on the page

Field Note (optional)

Small reminder text displayed underneath field

Save Cancel

The second required variable, "Processor Arguments" follows the same naming conventions. See below for an example.


Edit Field

You may add a new project field to this data collection instrument by completing the fields below and clicking the Save button at the bottom. When you add a new field, it will be added to the form on this page. For an overview of the different field types available, you may view the Field Types video (4 min).

Field Type: Notes Box (Paragraph Text)

Field Label: Processor Arguments

Use the Rich Text Editor

Variable Name (utilized in logic, calcs, and exports): slant_v1_args

Enable auto naming of variable based upon its Field Label?

ONLY letters, numbers, and underscores

How to use Smart Variables Piping

Required?* [✓ No] Yes

* Prompt if field is blank

Identifier? [✓ No] Yes

Does the field contain identifying information (e.g., name, SSN, address)?

Custom Alignment: Right / Vertical (RV)

Align the position of the field on the page

Action Tags / Field Annotation (optional)

Learn about @ Action Tags or using Field Annotation

Field Note (optional)

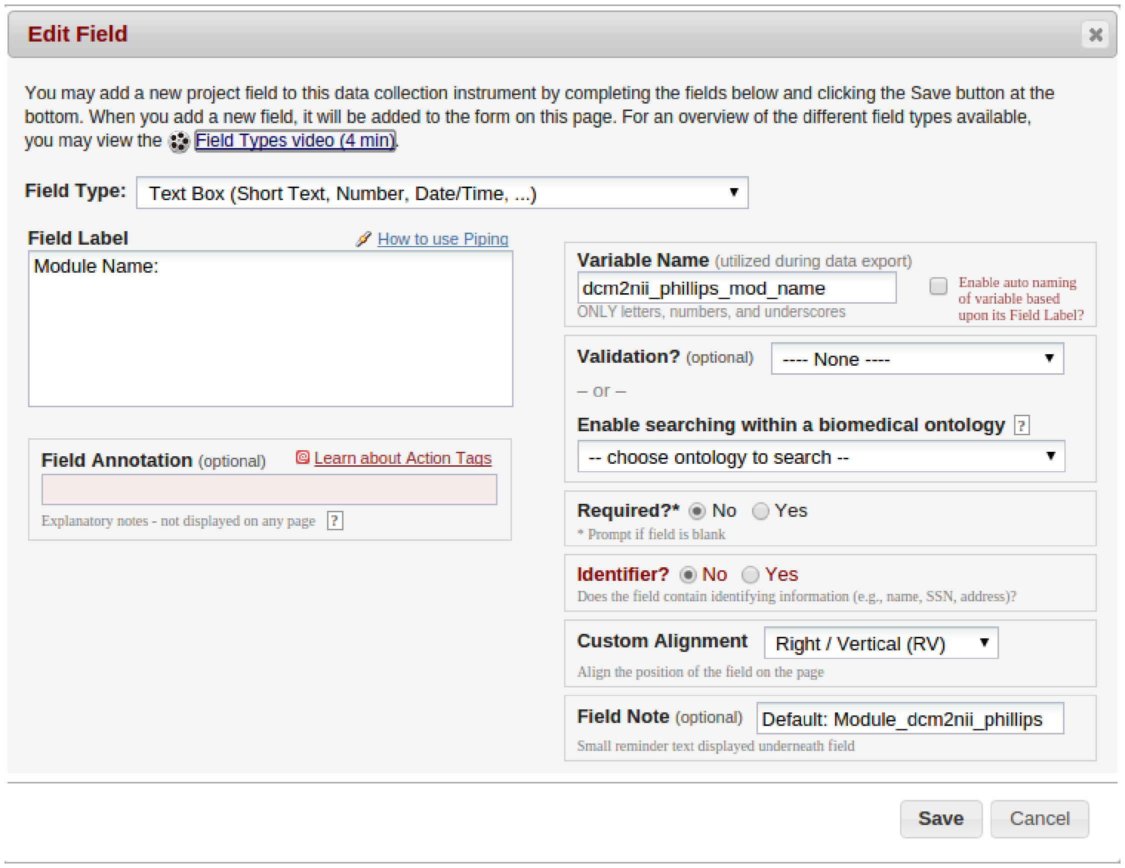
Small reminder text displayed underneath field

Save Cancel

2.4.3 LDAX

How to add a Module in LDAX

Variables used in a module must all start with the text immediately AFTER Module. For example, consider "Module dcm2nii philips". All of the variables for this module must start with "**dcm2nii_philips_**". One required variable is the "on" variable. This variable, again, in the case of "Module dcm2nii philips", would be called "dcm2nii_philips_on". This is used to check to see if the module related to this record in REDCap should be run for your project or not. It must also be of the yes/no REDCap type. If you do not have this variable included, you will get errors when you run dax_manager. The second required variable is the "Module name" variable. In the case of "Module dcm2nii philips", this variable is called "dcm2nii_philips_mod_name". This relates to the class name of the python module file. This information is stored in the REDCap "Field Note" (See below).


Edit Field

You may add a new project field to this data collection instrument by completing the fields below and clicking the Save button at the bottom. When you add a new field, it will be added to the form on this page. For an overview of the different field types available, you may view the Field Types video (4 min).

Field Type: Text Box (Short Text, Number, Date/Time, ...)

Field Label: Module Name:

Field Annotation (optional)

Explanatory notes - not displayed on any page

Variable Name (utilized during data export): dcm2nii_phillips_mod_name

ONLY letters, numbers, and underscores

Enable auto naming of variable based upon its Field Label?

Validation? (optional): ---- None ----

– or –

Enable searching within a biomedical ontology: -- choose ontology to search --

Required?* [✓ No] Yes

* Prompt if field is blank

Identifier? [✓ No] Yes

Does the field contain identifying information (e.g., name, SSN, address)?

Custom Alignment: Right / Vertical (RV)

Align the position of the field on the page

Field Note (optional): Default: Module_dcm2nii_phillips

Small reminder text displayed underneath field

This variable must be a REDCap Text Box type (as do all other variables at this point). This must be entered in the following format: "Default: <Module_Class_Name>". All other variables that are used must also start with the "**dcm2nii_philips_**" prefix and must match those of the module init.

Additionally, for the sake of user-friendliness, all variables should use REDCap's branching logic to only appear if the module is "on". It is important to note that in all cases, the REDCap "Field Label" is not used in any automated fashion, but should be something obvious to the users.

How to add a Process in LDAX

Just like in the case of Modules, Processes follow a close formatting pattern. Similarly, all process variables should start with the text immediately after "Process ". For this example, consider "Process Multi_Atlas". Just like in the case of the modules, the first variable should be a REDCap yes/no and should be called "multi_atlas_on". The remainder of the variables should all be of REDCap type "Text Box". The next required variable is the "Processor Name" variable which must be labeled with the "<Process Name>_proc_name" suffix. In the case of "Process Multi_Atlas", this is called "multi_atlas_proc_name". Just like in the case of the Module, the class name of the processor should be entered in the REDCap Field Note after "Default: ".

There are several other required variables which will be enumerated below (suffix listed first):

1. _suffix_proc - Used to determine what the processor suffix (if any should be)
2. _version - The version of the spider (1.0.0, 2.0.1 etc)
3. _walltime - The amount of walltime to use for the spider when executed on the grid
4. _mem_mb - The amount of ram to request for the job to run. Note this should be in megabytes

5. _scan_types - If writing a ScanProcessor, this is required. If writing a SessionProcessor, this is not required. This, in the case of a ScanProcessor, is used to filter out the scan types that the processor will accept to run the spider on.

Just like in the case of a Module, all variables other than the "on" variable should use REDCap branching logic to only be visible when the process is "on".

2.5 Contributors

DAX is a multi-institution collaborative effort of the following labs:

MASI at Vanderbilt University, Nashville, Tennessee, USA

Center for Cognitive Medicine at Vanderbilt University Medical Center, Nashville, Tennessee, USA

TIG at UCL (University College London), London, UK

2.6 How To Contribute

We encourage all collaborations! However, we follow a pull-request work flow to help facilitate a simplified code-review process. If you would like to contribute, we kindly request that any of your work be done in a branch. Rules for branching and merging are outlined below:

1. Branches - The scope of your branch should be narrow. Do not make a branch only for changing documentation, and then refactor how task.py works. These should be two totally separate branches.
2. Testing - You should test your branch before making a pull request. Do not make a pull request with untested code.
3. Committing - Use helpful commit messages. Do not use messages like "updates", "bug fix", and "updated a few files" etc. Please make these commit messages at least somewhat helpful. Use lots of commits, do not make 1 bulk commit of all of the changes that you make. This practice makes it hard for others to review.
4. Pull request - When you are ready to make a pull request, please try to itemize all of the changes that you made in at least moderate depth. This will alert everyone reviewing the code of possible things to check to make sure that you didn't break anything.
5. Merging - Do NOT merge your own pull request. Contributors should review each and every pull request before merging into the master branch. Please allow at least a few days before commenting and asking for status. If the depth of changes is deep, please allow at least a few weeks.
6. Master branch - NEVER commit to the master branch directly unless there is a serious bug fix.

If you are unfamiliar with branches in github, please see the link below:

Working with Branches

2.7 FAQ

These FAQs assume that you have read the XNAT documentation and or are familiar with navigating through the web UI. If you are not, you can read the XNAT documentation here.

1. **What is DAX?**
   DAX is an open source project that uses the pyxnat wrapper for the REST api to automate pipeline running on a DRMAA compliant grid.
2. **What are Modules?**
   Modules are a special class in DAX. They represent, generally, a task that should not be performed on the grid. The purpose for this was to not fill up the grid queue with jobs that take 20-30 seconds. Examples of such tasks could be converting a DICOM to a NIfTI file, changing the scan type, archiving a session from the prearchive, or performing skull-stripping. As you can see, these tasks can all be considered "light-weight" and thus probably don't have a place on the grid.
3. **What are Spiders?**
   Spiders are a python script. The purpose of the script is to download data from XNAT, run an image processing pipeline, and then prepare the data to be uploaded to XNAT. Spiders are run on the grid because they can take hours to days.
4. **How do I know the EXACT command line call that was made?**
   The PBS resource contains the script that was submitted to the grid scheduler for execution. You can view this file for the exact command line call(s) that were executed on the grid.
5. **I think I found a bug, what should I do?**
   The easiest way to get a bug fixed is to post as much information as you can on the DAX github issue tracker. If possible, please post the command line call you made (with any sensitive information removed) and the stack trace or error log in question.
6. **I have an idea of something I want to add. How do I go about adding it?**
   Great! We'd love to see what you have to include! Please read the guidelines on how to contribute.

2.8 DAX Processors

2.8.1 About

DAX pipelines are defined by creating YAML text files. If you are not familiar with YAML, start here: https://learnxinyminutes.com/docs/yaml/.

A processor YAML file defines the Environment, Inputs, Commands, and Outputs of your pipeline.

2.8.2 Processor Repos

There are several existing processors that can be used without modification. The processors in these repositories can also provide valuable examples.

https://github.com/VUIIS/dax_yaml_processor_examples

https://github.com/VUIIS/yaml_processors (private, internal use only)

2.8.3 Overview

The processor file defines how a script to run a pipeline should be created. DAX will use the processor to generate scripts to be submitted to your cluster as jobs. The script will contain the commands to download the inputs from XNAT, run the pipeline, and prepare the results to be uploaded back to XNAT (the actual uploading is performed by DAX via *dax upload*).

2.8.4 A "Simple" Example

```
---
moreauto: true
inputs:
  default:
    container_path: MRIQA_v1.0.0.simg
  xnat:
    scans:
      - name: scan_t1
        types: MPRAGE
        resources:
          - resource: NIFTI
            ftype: FILE
            varname: t1_nifti
outputs:
  - path: stats.txt
    type: FILE
    resource: STATS
  - path: report.pdf
    type: FILE
    resource: PDF
  - path: DATA
    type: DIR
    resource: DATA
command: >-
  singularity
  run
  --contain
  --cleanenv
  --home $INDIR
  --bind $INDIR:/dev/shm
  --bind $INDIR:/tmp
  --bind $INDIR:/INPUTS
  --bind $OUTDIR:/OUTPUTS
  {container_path}
  --t1_nifti /INPUTS/{t1_nifti}
attrs:
  walltime: '36:00:00'
  memory: 8192
```

2.8.5 Parts of the Processor YAML

All processor YAML files should start with these two lines:

```
---
moreauto: true
```

The primary components of a processor YAML file are:

- inputs
- outputs
- command
- attrs

Each of these components is required.

2.8.6 inputs

The **inputs** section defines the files and parameters to be prepared for the pipeline. Currently, the only subsections of inputs supported are **defaults** and **xnat**.

The **defaults** subsection can contain paths to local resources such as singularity containers, local codebases, local data to be used by the pipeline. It can essentially contain any value that needs to be passed directly to the **command** template (see below).

The **xnat** section defines the files, directories or values that are extracted from XNAT and passed to the command. Currently, the subsections of **xnat** that are supported are **scans**, **assessors**, **attrs**, and **filters**. Each of these subsections contains an array with a specific set of fields for each item in the array.

xnat scans

Each **xnat scans** item requires a **types** field. The **types** field is used to match against the scan type attribute on XNAT. The value can be a single string or a comma-separated list. Wildcards are also supported.

By default, any scan that matches will be included. You can exclude scans with a quality of *unusable* on XNAT by including the field **needs_qc** with value of *True*. The default is to run anything, i.e. a **needs_qc** value of *False*. Note that *questionable* is treated the same as *usable*, so they'll always run.

The **resources** subsection of each xnat scan should contain a list of resources to download from the matched scan. Each resource requires fields for **ftype** and **var**.

ftype specifies what type to downloaded from the resource, either *FILE*, *DIR*, or *DIRJ*. *FILE* will download individual files from the resource. *DIR* will download the whole directory from the resource with the hierarchy maintained. *DIRJ* will also download the directory but strips extraneous intermediate directories from the produced path as implemented by the *-j* flag of unzip.

The **var** field defines the tag to be replaced in the **command** string template (see below).

The optional **fmatch** field defines a regular expression to apply to filter the list of filenames in the resource.

xnat assessors

Each xnat assessor item requires a **proctype** field. The **proctype** field is used to match against the assessor proctype attribute on XNAT. The value can be a single string or a comma-separated list. Wildcards are also supported.

By default, any assessor that matches **proctype** will be included. However if **needs_qc** is set to *True*, assessors with a qcstatus of "Needs QA", "Bad", "Failed", "Poor", or "Do Not Run" will be excluded.

The **resources** subsection of each xnat assessor should contain a list of resources to download from the matched scan. Each resource requires fields for **ftype** and **var**.

The **ftype** specifies what type to downloaded from the resource, either *FILE*, *DIR*, or *DIRJ*. *FILE* will download individual files from the resource. *DIR* will download the whole directory from the resource with the hierarchy maintained. *DIRJ* will also download the directory but strips extraneous intermediate directories from the produced path as impelemented by the "-j" flag of unzip.

The **var** field defines the tag to be replaced in the **command** string template (see below).

Optional fields for a resource are fmatch, fdest and fcount. fmatch defines a regular expression to apply to filter the list of filenames in the resource. fcount can be used to limit the number of files matched. By default, only 1 file is downloaded. The inputs for some containers are expected to be in specific locations with specific filenames. This is accomplished using the **fdest** field. The file or directory gets copied to /INPUTS and renamed to the name specified in **fdest**.

xnat attrs

You can evaluate attributes at the subject, session, or scan level. Any fields that are accessible via the XNAT API can be queried. Each **attrs** item should contain a **varname**, **object**, and **attr**. **varname** specifies the tag to be replaced in the **command** string template. **object** is the XNAT object type to query and can be either *subject*, *session*, or *scan*. **attr** is the XNAT field to query. If the object type is *scan*, then a scan name from the xnat scans section must be included with the **ref** field.

For example:

```yaml
attrs:
    - varname: project
      object: session
      attr: project
```

This will extract the value of the project attribute from the session object and replace {project} in the command template.

xnat filters

filters allows you to filter a subset of the cartesian product of the matched scans and assessors. Currently, the only filter implemented is a match filter. It will only create the assessors where the specified list of inputs match. This is used when you want to link a set of assessors that all use the same initial scan as input.

For example:

```yaml
filters:
    - type: match
      inputs: scan_t1,assr_freesurfer/scan_t1
```

This will tell DAX to only run this pipeline where the value for scan_t1 and assr_freesurfer/scan_t1 are the same scan.

outputs

The **outputs** section defines a list files or directories to be uploaded to XNAT upon completion of the pipeline. Each output item must contain fields **path**, **type**, and **resource**. The **path** value contains the local relative path of the file or directory to be uploaded. The type of the path should either be *FILE* or *DIR*. The **resource** is the name of resource of the assessor created on XNAT where the output is to be uploaded.

For every processor, a *PDF* output with **resource** named PDF is required and must be of type *FILE*.

command

The **command** field defines a string template that is formatted using the values from **inputs**.

Each tag specified inside curly braces ("{}"") corresponds to a field in the **defaults** input section, or to a **var** field from a resource on an input or to a **varname** in the xnat attrs section.

Not all **var** must be used.

attrs

The **attrs** section defines miscellaneous other attributes including cluster parameters. These values replace tags in the jobtemplate.

jobtemplate

The **jobtemplate** is a text file that contains a template to create a batch job script.

2.8.7 Versioning

By default, name and version are parsed from the container file name, based on the format: <NAME>_v<major.minor.revision>.simg where<NAME>_v<major> is the proctype.

The YAML file can override these by using any of the top level fields **procversion**, **procname**, and/or **proctype**. **procversion** specifies the major.minor.revision, e.g. *1.0.2*. **procname** specifies the name only without version, e.g. mprage. **proctype** is the name and major version, e.g. *mprage_v1*. If only **procname** is specified, the version is parsed from the container name. If only **procversion** is specified, the name is parsed from the container name. If **proctype** is specified, it will override everything else to determine proctype.

2.8.8 Notes on Singularity run options

–cleanenv avoids env confusion, and –contain prevents accidentally using code from the host filesystem. However, with –contain, some spiders will need to have specific temp space on the host attached. E.g. for some versions of Freesurfer, –bind ${INDIR}:/dev/shm. For compiled Matlab spiders, we need to provide –home $INDIR to avoid .mcrCache collisions in temp space when multiple spiders are running. And, some cases may require ${INDIR}:/tmp or /tmp:/tmp.

2.9 DAX Processors, version 3

2.9.1 About

DAX pipelines are defined by creating YAML text files. If you are not familiar with YAML, start here: https://learnxinyminutes.com/docs/yaml/.

A processor YAML file defines the Environment, Inputs, Commands, and Outputs of your pipeline.

Version 3 processors have a number of new options and conveniences.

2.9.2 Processor Repos

There are several existing processors that can be used without modification. The processors in these repositories can also provide valuable examples.

https://github.com/VUIIS/dax_yaml_processor_examples

https://github.com/VUIIS/yaml_processors (private, internal use only)

2.9.3 Overview

The processor file defines how a script to run a pipeline should be created. DAX will use the processor to generate scripts to be submitted to your cluster as jobs. The script will contain the commands to download the inputs from XNAT, run the pipeline, and prepare the results to be uploaded back to XNAT (the actual uploading is performed by DAX via *dax upload*).

2.9.4 A Basic Example

```
---
procyamlversion: 3.0.0-dev.0                # Indicates to run as a v3 processor

containers:                                 # Containers we will ref in the command␣
↪section
  - name: EXAMP                               # Reference by this name in command␣
↪section
    path: example_v2.0.0.sif                  # Name/path that is replaced in command␣
↪section
    source: docker://vuiiscci/example:v2.0.0     # Not used, but good practice to set it

requirements:  # Requirements for the cluster node, substituted into SBATCH section of␣
↪job template
  walltime: 0-2  # Time to request - SLURM supports the format DAYS-HOURS
  memory: 16G

inputs:
  vars:   # Keyvalues to substitute in the command, for passing static settings
      - param1: param1value
  xnat:
    attrs:  # Values to extract from xnat at the specified level of the current instance
      - varname:  scanID     # Name to be used to dereference later
```

(continues on next page)

(continued from previous page)

```
        object:   scan       # Source, of: project, subject, session, scan, assessor
        attr:     ID         # Name of the field in xnat
        ref:      scan_fmri  # From which object in inputs, referred to by name
    scans:
      - name: scan_fmri       # the name of this scan to dereference later
        types: fMRI_run*      # the scan types to match on the session in XNAT
        nifti: fmri.nii.gz    # Shortcut to download file in NIFTI resource as fmri.nii.
↪gz
        resources:            # To get files in other resources
          - resource: EDAT        # Name of the resource
            fdest: edat.txt       # Download the file as edat.txt
            varname: edat_txt     # Reference for command string substitution
    assessors:
      - name: assr_preproc
        proctypes: preproc-fmri_v2
        resources:
          - {resource: FILTERED_DATA, fdest: filtered_data.nii.gz}

outputs:
  - pdf: report*.pdf        # Matching file uploaded to PDF resource
  - stats: stats.txt        # Matching file uploaded to STATS resource
  - dir: PREPROC            # Matching directory (PREPROC) uploaded to PREPROC resource
  - path: inputpathname     # General purpose for other outputs
    type: DIR                   # Type is FILE or DIR
    resource: RESOURCENAME      # Store it in resource RESOURCENAME

# Available commands are 'singularity_run' and 'singularity_exec'. These include default
# flags --contain --cleanenv, and mount points for temp space plus INPUTS and OUTPUTS
command:
  type: singularity_run
  extraopts: []       # Appends to default options for the run command
  container: EXAMP    # Name of the container in the list above
  args: >-
    --fmri_file /INPUTS/fmri.nii.gz
    --filtered_file /INPUTS/filtered_data.nii.gz
    --param1 {param1value}
    --scan_id {scanID}
    --edat_txt /INPUTS/{edat_txt}

description: |
  Example description that gets printed to every PDF created by this processor
  1. step 1 does something cool
  2. step 2 does this other thing

# Specify the job template to use (examples: https://github.com/VUIIS/dax_templates/)
job_template: job_template_v3.txt
```

2.9.5 Parts of the Processor YAML

2.9.6 inputs (required)

The **inputs** section defines the files and parameters to be prepared for the pipeline. Currently, the only subsections of inputs supported are **vars** and **xnat**.

The **vars** subsection can store parameters to be passed as pipeline options, such as smoothing kernel size, etc that may be more conveniently coded here to substitute into the command arguments.

The **xnat** section defines the files, directories or values that are extracted from XNAT and passed to the command. Currently, the subsections of **xnat** that are supported are **scans**, **assessors**, **attrs**, and **filters**. Each of these subsections contains an array with a specific set of fields for each item in the array.

xnat scans

Each **xnat scans** item requires a **types** field. The **types** field is used to match against the scan type attribute on XNAT. The value can be a single string or a comma-separated list. Wildcards are also supported.

The **resources** subsection of each xnat scan should contain a list of resources to download from the matched scan.

ftype specifies what type to downloaded from the resource, either *FILE*, *DIR*, or *DIRJ*. *FILE* will download individual files from the resource. *DIR* will download the whole directory from the resource with the hierarchy maintained. *DIRJ* will also download the directory but strips extraneous intermediate directories from the produced path as implemented by the *-j* flag of unzip.

The **varname** field defines tags to be replaced in the **command** string template (see below).

The optional **fmatch** field defines a regular expression to apply to filter the list of filenames in the resource. **fmulti** affects how inputs are handled when there are multiple matching files in a resource. By default, this situation causes an exception, but if **fmulti** is set to *any1*, a single (arbitrary) file is selected from the matching files instead.

By default, any scan that matches will be included as an available input. Several optional settings affect this:

- If **needs_qc** is *True* and **require_usable** is *False* or not specified, assessors that would have a scan as an input will be created, but will not run if the scan is marked *unusable*.
- If **needs_qc** is *True* and **require_usable** is also *True*, assessors that would have a scan as an input will be created, but will not run unless the scan is marked *usable*.
- If **skip_unusable** is *True*, assessors that would have an *unusable* scan as an input will not even be created.
- **keep_multis** may be *all* (the default); *first*; *last*; or an index 1,2,3,… This applies when there are multiple scans in the session that match as possible inputs. Normally all matching scans are used as inputs, multiplying assessors as needed. When *first* is specified, only the first matching scan will be used as an input, reducing the number of assessors built by a factor of the number of matching scans. “First” is defined as alphabetical order by scan ID, cast to lowercase. The exact scan type is not considered; only whether there is a match with the **types** specified.

xnat assessors

Each xnat assessor item requires a **proctype** field. The **proctype** field is used to match against the assessor proctype attribute on XNAT. The value can be a single string or a comma-separated list. Wildcards are also supported.

Any assessor that matches **proctype** will be included as a possible input. However if **needs_qc** is set to *True*, input assessors with a qcstatus of “Needs QA”, “Bad”, “Failed”, “Poor”, or “Do Not Run” will cause the new assessor not to run.

The **resources** subsection of each xnat assessor should contain a list of resources to download from the matched scan.

The **ftype** specifies what type to downloaded from the resource, either *FILE*, *DIR*, or *DIRJ*. *FILE* will download individual files from the resource. *DIR* will download the whole directory from the resource with the hierarchy maintained. *DIRJ* will also download the directory but strips extraneous intermediate directories from the produced path as impelemented by the “-j” flag of unzip.

The **varname** field defines the tag to be replaced in the **command** string template (see below).

Optional fields for a resource are **fmatch** and **fdest**. fmatch defines a regular expression to apply to filter the list of filenames in the resource. The inputs for some containers are expected to be in specific locations with specific filenames. This is accomplished using the **fdest** field. The file or directory gets copied to /INPUTS and renamed to the name specified in **fdest**.

xnat attrs

You can evaluate attributes at the subject, session, or scan level. Any fields that are accessible via the XNAT API can be queried. Each **attrs** item should contain a **varname**, **object**, and **attr**. **varname** specifies the tag to be replaced in the **command** string template. **object** is the XNAT object type to query and can be either *subject*, *session*, or *scan*. **attr** is the XNAT field to query. If the object type is *scan*, then a scan name from the xnat scans section must be included with the **ref** field.

For example:

```yaml
attrs:
    - varname: project
      object: session
      attr: project

# Or equivalently
attrs:
    - {varname: project, object: assessor, attr: project}
```

This will extract the value of the project attribute from the assessor object and replace {project} in the command template.

xnat filters

filters allows you to filter a subset of the cartesian product of the matched scans and assessors. Currently, the only filter implemented is a match filter. It will only create the assessors where the specified list of inputs match. This is used when you want to link a set of assessors that all use the same initial scan as input.

For example:

```yaml
filters:
    - type: match
      inputs: scan_t1,assr_freesurfer/scan_t1
```

This will tell DAX to only run this pipeline where the value for scan_t1 and assr_freesurfer/scan_t1 are the same scan.

outputs

The **outputs** section defines a list files or directories to be uploaded to XNAT upon completion of the pipeline. Each output item must contain fields **path**, **type**, and **resource**. The **path** value contains the local relative path of the file or directory to be uploaded. The type of the path should either be *FILE* or *DIR*. The **resource** is the name of resource of the assessor created on XNAT where the output is to be uploaded.

For every processor, a *PDF* output with **resource** named PDF is required and must be of type *FILE*.

PDF and *STATS* outputs, as well as *DIR* type outputs, have shortcuts as shown in the example.

command

The **command** field defines a string template that is formatted using the values from **inputs**.

Each tag specified inside curly braces ("{}"") corresponds to a field in the **defaults** input section, or to a **var** field from a resource on an input or to a **varname** in the xnat attrs section.

See the example for explanations of the other fields.

jobtemplate

The **jobtemplate** is a text file that contains a template to create a batch job script.

2.9.7 Versioning

Processor name and version are parsed from the processor file name, based on the format <NAME>_v<major.minor.revision>.yaml. <NAME>_v<major> will be used as the proctype.

2.9.8 Notes on singularity options

The default options are *SINGULARITY_BASEOPTS* in dax/dax/processors_v3.py:

```
--contain --cleanenv
--home $JOBDIR
--bind $INDIR:/INPUTS
--bind $OUTDIR:/OUTPUTS
--bind $JOBDIR:/tmp
--bind $JOBDIR:/dev/shm
```

$JOBDIR, $INDIR, $OUTDIR are available at run time, and refer to locations on the filesystem of the node where the job is running.

Singularity has default binds that differ between installations. –contain disables these to prevent cross-talk with the host filesystem. And –cleanenv prevents cross-talk with the host environment. However, with –contain, some spiders will need to have specific temp space on the host attached. E.g. for some versions of Freesurfer, –bind ${INDIR}:/dev/shm. For compiled Matlab spiders, we need to provide –home $INDIR to avoid .mcrCache collisions in temp space when multiple spiders are running. And, some cases may require ${INDIR}:/tmp or /tmp:/tmp. Thus the defaults above.

The entire singularity command is built as:

```
singularity <run|exec> <SINGULARITY_BASEOPTS> <extraopts> <container> <args>
```

2.9.9 Subject-Level Processors

As of version 2.7, dax supports subject-level processors, in addition to session-level. The subject-level processors can include inputs across multiple sessions within the same subject. In the processor yaml, a subject-level processor is implied by including the "sessions" level between inputs.xnat and scans/assessors. Each session requires the attribute types. The types are matched against the XNAT field xnat:imageSessionData/session_type. Currently the match must be exact.

To set the session type of a session, you can use dax/pyxnat:

```
xnat.select_session(PROJ, SUBJ, SESS).attrs.set('session_type', SESSTYPE)
```

Below is an example of a subject-level processor that will include an assessor from two different sessions of session types Baseline and Week12.

```
---
procyamlversion: 3.0.0-dev.0
containers:
  - name: EMOSTROOP
    path: fmri_emostroop_v2.0.0.sif
    source: docker://bud42/fmri_emostroop:v2
requirements:
  walltime: 0-2
  memory: 16G
inputs:
  xnat:
    sessions:
      - types: Baseline
        assessors:
          - name: assr_emostroop_a
            types: fmri_emostroop_v1
            resources:
              - resource: PREPROC
                fmatch: swauFMRI.nii.gz
                fdest:  swauFMRIa.nii.gz
      - types: Week12
        assessors:
          - name: assr_emostroop_c
            types: fmri_emostroop_v1
            resources:
              - resource: PREPROC
                fmatch: swauFMRI.nii.gz
                fdest:  swauFMRIc.nii.gz
outputs:
  - dir: PREPROC
  - dir: 1stLEVEL
command:
  type: singularity_run
  container: EMOSTROOP
  args: BLvsWK12
```

The assessor will be created under the subject on XNAT, at the same level as a session. The proctype of the assessor will be derived from the filename just like session-level processors. The XNAT data type of the assessor, or xsiType, will be proc:subjGenProcData (for session-level assessors the type is proc:genprocData).

2.10 Assessors in VUIIS XNAT

An assessor is processed on XNAT. All files produced by a script using data from one scan / multiple scans / any other process data will be / need to be upload to an assessor.

The VUIIS XNAT is using two kind of assessors :

- proc:genProcData : the generic assessor type
- fs:fsData : the specific FreeSurfer assessor type that we created (deprecated)

We are using these statuses for the assessor:

- NO_DATA : no data exists on the sessions to be able to run
- NEED_INPUTS : input data has not been created yet for a scan, multiple scans or other assessor; sometimes this means the inputs it needs aren't present, other times, this means everything is present but the assessor hasn't built yet
- NEED_TO_RUN : ready to be launched on the cluster (ACCRE). All input data for the process to run exists
- JOB_RUNNING : the assessor is built and the job is running on ACCRE or the job is completed and is waiting to be uploaded
- JOB_FAILED : the job failed on the cluster
- READY_TO_UPLOAD : Job done, waiting for the results to be uploaded to XNAT from the cluster
- UPLOADING : in the process of uploading the resources on XNAT
- READY_TO_COMPLETE : the assessors contains all the files but we still need finish up (this includes getting the walltime and memory used on ACCRE)
- COMPLETE : all finished

There is a QA status that is managed by the project owner. This field defaults to "Needs QA". Other values can be set as desired. If set to "Rerun", the assessor will automatically be deleted and rerun.

2.11 DAX Command Line Tools

2.11.1 Table of Contents

8. *XnatSetup*
9. *XnatProcessUpload*
10. *XnatSubjectUpdate*
11. *RedCapReport*
12. *XnatCheckLogin*
13. *Xnatinfo*
14. *Xnatsessionupdate*
15. *BIDSMapping*
16. *XnatBOND*

2.11.2 List of the Tools

Each tool has a help option and some examples on how to use the tools. You can call each tool with no arguments to see the help.

XnatSetup

You can use the Xnatsetup (see below) command tool to setup your computer with the –basic options. It will do what is below automatically, but if you don't want to do that, it can be setup manually.

This Xnat commands will use two thing :

- install pyxnat and python packages on your computer (Check 'Get started for Spiders' on the wiki)
- set your bashrc with the env variable to connect to Xnat with pyxnat :

```
export XNAT_HOST=http://xnat.vanderbilt.edu/xnat
export XNAT_USER=username
export XNAT_PASS=password
export PATH=/PathToMasimatlab/trunk/xnatspiders/Xnat_tools:$PATH
```

FYI : you can open the bashrc like :

```
vim (or nano or any editor you like) ~/.bashrc
```

and when you are done editing it, use :

```
. ~/.bashrc
```

You will after this be able to call the commands directly on your terminal.

One last thing, the Xnatupload will send you updates (errors and warnings) about the directory you are trying to upload on Xnat. If you want to receive this email, you need to set up two variables in your bashrc :

```
export EMAIL_ADDR=add@gmail.com
export EMAIL_PWS=passwordforthisemail
```

It will use this email address to send you email. It has to be a gmail address.

Xnatsetup is as you can guess a command tool to set up your computer. It will install the python package needed and ask for the variables that need to be set up. There are different kinds of setup :

- basic to be able to use the XNAT command tools
- advance to run spiders on your computer or ACCRE
- redcap to use the spider to send data to redcap
- cci package setup for ACCRE or if you need XnatUtils
- ACCRE setup

```
#######################################################
#                                            XNATSETUP
# XnatSetup is a command tool to set up on your computer the variables to use
#      the tools/spiders.
# Developed by the masiLab Vanderbilt University, TN, USA.
# Operating system : Linux & Mac OS
# Shell : bash
# Requirements : python with pip & git
# Contact : benjamin.c.yvernault@vanderbilt.edu
#     No Arguments given
#     See the help bellow or Use "Xnatquery" -h
#######################################################
Usage: Xnatsetup [options]

What is the script doing : Set up your computer to use xnat.
  *Basic installation (--basic) - Needed to use the Xnat command tools or any of
the next installations : install the python package httplib2, lxml, and pyxnat if not
already install & saving your username/host/password for XNAT.
  *Advance installation (--advance ) - Needed to run the non-specific spiders :
Set up the Upload Directory, set up masimatlab path for Xnatrun, add the xnat
tools to your PATH, and add Spiders.py in your PYTHONPATH .
  *Redcap installation (--redcap ) - Needed to use redcap spiders (send data to
redcap) : Install Pycap / pandas if not install and set up the URL for redcap .
  *API installation (--api ) - Needed to use API package to run spiders on ACCRE
via jobs (Contains XnatUtils) : Install API if not install.
  *ACCRE installation (--accre) - Setup the environment to use the
package/spiders/tools on ACCRE.


Options:
 -h, --help            show this help message and exit
 --basic               Use this options to set up the env variables to use
                       the Xnat tools and have the basic set up.
 --advance             Use this options to set up the env variables to run
                       spiders in general.
 --redcap              Use this options to set up the env variables to use
                       redcap spiders.
 --api                 Use this options to set up the env variables to run
                       spiders on ACCRE via jobs.
 --Accre               Use this options if you are on Accre.
 --NoSudo              Use this options if you don't have sudo access and you
                       still want to install the package (check -d option).
 -d INSTALLDIR, --installdir=INSTALLDIR
                       Use this options to specify a directory where the
                       python package need to be install. It works only if
```

(continues on next page)

(continued from previous page)

```
                        you use --NoSudo option.
 --tutorial             Give you the step for the specific setup you are
                        asking.
```

Contact - benjamin.c.yvernault@vanderbilt.edu

XnatQuery

Xnatquery will show you the tree on xnat. Xnatquery is a tool to query objects on XNAT for each level. You can see which projects you have access to and see the hierarchy of data on your project. It has several options (accessible with -h or –help) :

```
################################################################
#                            XnatQuery                         #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Query through XNAT at the level you want.                #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################

----------------------------------------------------------------
usage: XnatQuery [-h] [--host HOST] [-u USERNAME] [-p PROJECT] [-s SUBJECT]
                 [-e SESSION] [-a ASSESSOR] [-c SCAN] [--all] [--me]

What is the script doing :
   * Query on Xnat at any level.

Examples:
   *Show all the projects you have access to:
        Xnatquery --me
   *Show all projects:
        Xnatquery --all
   *Query a specific level (example scan/assessors for a session):
        Xnatquery -p PID -s 109873 -e 109873
   *Query a specific level with all objects under it :
        Xnatquery -p PID -s 109873 --all

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host for XNAT. Default: env XNAT_HOST.
  -u USERNAME, --username USERNAME
                        Username for XNAT.
  -p PROJECT, --project PROJECT
                        project ID on Xnat or 'all' to see all the project.
  -s SUBJECT, --subject SUBJECT
                        Subject label on Xnat
  -e SESSION, --experiment SESSION
```

(continues on next page)

(continued from previous page)

```
                            Session label on Xnat
  -a ASSESSOR, --assessor ASSESSOR
                            Assessor/Process label on XNAT. E.G: VUSTP-x-VUSTP1-x-VUSTP1a-x-
↪FS
  -c SCAN, --scan SCAN  Scan ID on Xnat.
  --all                     Print all the objects on XNAT from the level you are at.
  --me                      Give the projects ID that you have access.
```

Extra Examples

- To get information on the project

```
Xnatquery -p projectID --info
```

- To get all the subjects in this project

```
Xnatquery -p projectID
```

- To get all the experiments in this project

```
Xnatquery -p projectID -s subject
```

Contact - benjamin.c.yvernault@vanderbilt.edu

XnatCheck

Xnatcheck is a quick way to check directly on your terminal if there is the resource you just created on all your project. You can check if there is a scan type or an assessor type as well with the options -s or -a. Options available (-h or -help):

```
################################################################
#                             XnatCheck                        #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Check XNAT data (subject/session/scan/assessor/resource) #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################

----------------------------------------------------------------
usage: XnatCheck [-h] [--host HOST] [-u USERNAME] [-p PROJECTS]
                 [--filters FILTERS [FILTERS ...]]
                 [--delimiter DELIMITER_FILTER_RESOURSE] [--csv CSV_FILE]
                 [--format FORMAT] [--printfilters] [--printformat]

What is the script doing :
   *Check object on XNAT (subject/session/scan/assessor/resources) specify by the␣
↪options.

How to write a filter string:
 - for resources filters, the string needs to follow this template:
```

(continues on next page)

(continued from previous page)

```
   variable_name=value--sizeoperatorValue--nbfoperatorValue--fpathsoperatorValue
   By default, it will return the assessor that does have the resource if no other␣
↪filter specify
 - for other filters, the string needs to follow this template:
   variable_name=Value
   operator can be different than =. Look at the table in --printfilters

Use --printfilters to see the different variables available

Examples:
   *See format variables:
        Xnatcheck --printformat
   *See filter variables:
        Xnatcheck --printfilters
   *Get list of T1,DTI scans that have a resource called NIFTI:
        Xnatcheck -p PID --filters type=T1,DTI assessor_res=NIFTI
   *Get list of fMRIQA assessors that have a resource called PDF:
        Xnatcheck -p PID --filters proctype=fMRIQA assessor_res=PDF
   *Get list of assessors except fMRIQA that have a resource called PDF :
        Xnatcheck -p PID --filters proctype!=fMRIQA assessor_res=PDF
   *Get list of project sessions that do not have a resource called testing:
        Xnatcheck -p PID --filters session_label=VUSTP1a,VUSTP2b,VUSTP3a session_res!
↪=testing
   *Get list of project fMRIQA and VBMQA that used more than 45mb and less than 1hour:
        Xnatcheck -p PID1,PID2 --filters proctype=fMRIQA,VBMQA procstatus=COMPLETE
↪"memused>45mb" "walltimeused<1:00:00" --format assessor_label,procnode,memused,
↪walltimeused

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host for XNAT. Default: env XNAT_HOST.
  -u USERNAME, --username USERNAME
                        Username for XNAT.
  -p PROJECTS, --project PROJECTS
                        Project(s) ID on XNAT
  --filters FILTERS [FILTERS ...]
                        List of filters separated by a space to apply to the search.
  --delimiter DELIMITER_FILTER_RESOURSE
                        Resource filters delimiter. By default: --.
  --csv CSV_FILE        File path to save the CSV output.
  --format FORMAT       Header for the csv. format: list of variables name comma-
↪separated
  --printfilters        Print available filters.
  --printformat         Print available format for display.
```

Extra Examples

- To return all the scans where there is no NIFTI on the project BLSA

```
Xnatcheck -p BLSA -r NIFTI
```

- To return all the assessors where there is no PDF on the project BLSA

```
Xnatcheck -p BLSA -r PDF -l 1
```

- To return all the subjects/experiments where there is no fMRIQA assessor on the project BLSA

```
Xnatcheck -p BLSA -a fMRIQA
```

- To return all the subjects/experiments where there is no fMRIQA assessor on the project BLSA and check for the one that exists if there is a PDF resource

```
Xnatcheck -p BLSA -a fMRIQA -r PDF
```

Contact - benjamin.c.yvernault@vanderbilt.edu

XnatDownload

Xnatdownload will download all the resources that you asked for in a directory. Xnatdownload provides bulk download of data from XNAT with specific filters applied. It provides options to narrow your download to only what you need. This tool will generate a folder per project in your -d directory with two files: download_commandLine.txt and download_report.csv with the description of what you downloaded. It has several options (accessible with -h or -help) :

```
################################################################
#                                                   XNATDOWNLOAD
#
# Developed by the masiLab Vanderbilt University, TN, USA.
# If issues, email benjamin.c.yvernault@vanderbilt.edu
# Parameters :
#     No Arguments given
#     See the help bellow or Use "Xnatdownload" -h
################################################################
usage: Xnatdownload [-h] [--host HOST] [-u USERNAME] [-p PROJECT]
                    [-d DIRECTORY] [-D] [--subj SUBJECT] [--sess SESSION]
                    [-s SCANTYPE] [-a ASSESSORTYPE] [--WOS WITHOUTS]
                    [--WOP WITHOUTA] [--quality QUALITIES] [--status STATUS]
                    [--qcstatus QCSTATUS] [-c CSVFILE] [--rs RESOURCESS]
                    [--ra RESOURCESA] [--selectionS SELECTIONSCAN]
                    [--selectionP SELECTIONASSESSOR] [--overwrite] [--update]
                    [--fullRegex] [-o OUTPUTFILE] [-i] [-b BIDS_DIR] [-xt]
                    [--bond_dir BOND_DIR]

What is the script doing :
   *Download filtered data from XNAT to your local computer using the different OPTIONS.

Examples:
   *Download all resources for all scans/assessors in a project:
        Xnatdownload -p PID -d /tmp/downloadPID -s all --rs all -a all --ra all
   *Download NIFTI for T1,fMRI:
        Xnatdownload -p PID -d /tmp/downloadPID -s T1,fMRI --rs NIFTI
   *Download only the outlogs for fMRIQA assessors that failed:
        Xnatdownload -p PID -d /tmp/downloadPID -a fMRIQA --status JOB_FAILED --ra OUTLOG
   *Download PDF for assessors that Needs QA:
        Xnatdownload -p PID -d /tmp/downloadPID -a all --qcstatus="Needs QA" --ra OUTLOG
   *Download NIFTI for T1 for some sessions :
```

(continues on next page)

(continued from previous page)

```
        Xnatdownload -p PID -d /tmp/downloadPID --sess 109309,189308 -s all --rs NIFTI
    *Download same data than previous line but overwrite the data:
        Xnatdownload -p PID -d /tmp/downloadPID --sess 109309,189308 -s all --rs NIFTI --
↪overwrite
    *Download data described by a csvfile (follow template) :
        Xnatdownload -d /tmp/downloadPID -c  upload_sheet.csv
    *Transform the XnatDownload data in BIDS format for all sessions, scantype and␣
↪resources:
        Xnatdownload -p PID --sess all -d /tmp/downloadPID -s all --rs all --bids /tmp/
↪BIDS_dataset
    *Transform the XnatDownload data in BIDS format for all sessions, scantype and␣
↪resources with xnat tag:
        Xnatdownload -p PID --sess all -d /tmp/downloadPID -s all --rs all --bids /tmp/
↪BIDS_dataset -xt
    *Transform the XnatDownload data in BIDS format for all sessions, scantype and␣
↪resources with xnat tag and perform bond:
        Xnatdownload -p PID --sess all -d /tmp/downloadPID -s all --rs all --bids /tmp/
↪BIDS_dataset -xt --bond /tmp/BOND_dir

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host for XNAT. Default: using $XNAT_HOST.
  -u USERNAME, --username USERNAME
                        Username for XNAT. Default: using $XNAT_USER.
  -p PROJECT, --project PROJECT
                        Project(s) ID on Xnat
  -d DIRECTORY, --directory DIRECTORY
                        Directory where the data will be download
  -D, --oneDirectory    Data will be downloaded in the same directory. No sub-
                        directory.
  --subj SUBJECT        filter scans/assessors by their subject label. Format:
                        a comma separated string (E.G: --subj VUSTP2,VUSTP3).
  --sess SESSION        filter scans/assessors by their session label. Format:
                        a comma separated string (E.G: --sess VUSTP2b,VUSTP3a)
  -s SCANTYPE, --scantype SCANTYPE
                        filter scans by their types (required to download
                        scans). Format: a comma separated string (E.G : -s
                        T1,MPRAGE,REST). To download all types, set to 'all'.
  -a ASSESSORTYPE, --assessortype ASSESSORTYPE
                        filter assessors by their types (required to download
                        assessors). Format: a comma separated string (E.G : -a
                        fMRIQA,dtiQA_v2,Multi_Atlas). To download all types,
                        set to 'all'.
  --WOS WITHOUTS        filter scans by their types and removed the one with
                        the specified types. Format: a comma separated string
                        (E.G : --WOS T1,MPRAGE,REST).
  --WOP WITHOUTA        filter assessors by their types and removed the one
                        with the specified types. Format: a comma separated
                        string (E.G : --WOP fMRIQA,dtiQA).
  --quality QUALITIES   filter scans by their quality. Format: a comma
                        separated string (E.G: --quality
                        usable,questionable,unusable).
```

(continues on next page)

(continued from previous page)

```
  --status STATUS       filter assessors by their job status. Format: a comma
                        separated string.
  --qcstatus QCSTATUS   filter assessors by their quality control status.
                        Format: a comma separated string.
  -c CSVFILE, --csvfile CSVFILE
                        CSV file with the following header: object_type,projec
                        t_id,subject_label,session_type,session_label,as_label
                        . object_type must be 'scan' or 'assessor' and
                        as_label the scan ID or assessor label.
  --rs RESOURCESS       Resources you want to download for scans. E.g : --rs
                        NIFTI,PAR,REC.
  --ra RESOURCESA       Resources you want to download for assessors. E.g :
                        --ra OUTLOG,PDF,PBS.
  --selectionS SELECTIONSCAN
                        Download from only one selected scan.By default : no
                        selection. E.G : project-x-subject-x-session-x-scan
  --selectionP SELECTIONASSESSOR
                        Download from only one selected processor.By default :
                        no selection. E.G : assessor_label
  --overwrite           Overwrite the previous data downloaded with the same
                        command.
  --update              Update the files from XNAT that have been downloaded
                        with the newest version if there is one (not working
                        yet).
  --fullRegex           Use full regex for filtering data.
  -o OUTPUTFILE, --output OUTPUTFILE
                        Write the display in a file giving to this OPTIONS.
  -i, --ignore          Ignore reading of the csv report file
  -b BIDS_DIR, --bids BIDS_DIR
                        Directory to store the XNAT to BIDS curated data
  -xt, --xnat_tag       Download BIDS data with XNAT subjID and sessID
  --bond_dir BOND_DIR   Download the Key groups and Param groups in BIDS data to BOND_DIR
```

Example

- Downloads in /home/benjamin/temp the resources NIFTI and PDF for all the scan fMRI and the assessor fMRIQA for the project BLSA

```
Xnatdownload -p BLSA -d /home/benjamin/temp/ -a fMRIQA -s fMRI -r NIFTI,PDF
```

Contact - benjamin.c.yvernault@vanderbilt.edu

XnatUpload

Xnatupload will create subject/experiment/scan/resources for a project on XNAT and upload the data into the project from a folder. Xnatupload provides bulk upload of data to a project on XNAT. You need to provide a specific CSV file with the following header:

- object_type,project_id,subject_label,session_type,session_label,as_label,as_type,as_description,quality,resource,fpath

where:

- as_label corresponds to assessor or scan label
- as_type corresponds to proctype or scantype

- as_description corresponds to procstatus or series description for the scan
- quality corresponds to qastatus or quality for scan

It should be similar to this (project in the example is CIBS-TEST):

object_type,project_id,subject_label,session_type,session_label,as_label,as_type,as_description,quality,resource,fpath
scan,CIBS-TEST,CIBS-TEST_01,MR,CIBS-TEST_01,401,BRAIN2_3DT1,BRAIN2_3DT1,usable,NIFTI,/Users/<USER>/Downloads

Methods

Warning: the project must already exist on XNAT. You can add a new project via the XNAT web GUI. Follow one of the three methods to upload:

- Number 1 : all the files are in one directory but they need to be rename like this projectID-x-subjectID-x-experimentID-x-scanID-x-scantype-x-resourcename.extention. Fastest methode but only one file can be upload in a resource.
- Number 2 : you don't need to rename all the files but you need to give a specific structure to your directory : folder/subjectID/experimentID/scanID-x-scantype/ResourceID/ and put the resources corresponding in it. E.G : TempDir/BLSA_0000/BLSA_0000_0/scan2-x-fMRI/NIFTI/nifti.nii.gz. It will not be as fast as the first methode but you can upload more than one file to a resources.
- Option -o : There is a third choice. If you want to upload files to Xnat on a scan and you don't want to create anything, you can use this options -o. It's for only upload. It's using something like the first methodes : put all the files into one folder with a special name : projectID-x-subjectID-x-experimentID-x-scanID-x-resourcename.extention for assessor, assessor_label-resourcename.extension for assessor (Reminder : assessor_label = projectID-x-subjectID-x-experimentID-x-scanID-x-process_name or projectID-x-subjectID-x-experimentID-x-processname).

```
################################################################
#                            XnatUpload                        #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Print a detailed report from XNAT projects.              #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################
IMPORTANT WARNING FOR ALL USERS ABOUT XNAT:
   session_label needs to be unique for each session.
   Two subjects can NOT have the same session_label
----------------------------------------------------------------
usage: XnatUpload [-h] [--host HOST] [-u USERNAME] -c CSV_FILE
                  [--sess SESSION_TYPE] [--report] [--force] [--delete]
                  [--deleteAll] [--noextract] [--printmodality]
                  [-o OUTPUT_FILE] [-b BIDS_DIR] [-p PROJECT]

What is the script doing :
   * Upload data to XNAT following the csv file information.
     csv header:
     object_type,project_id,subject_label,session_type,session_label,
     as_label,as_type,as_description,quality,resource,fpath

IMPORTANT: YOU NEED TO CREATE THE PROJECT ON XNAT BEFORE UPLOADING.
```

(continues on next page)

(continued from previous page)

```
Examples:
   * See Session type:
        Xnatupload --printmodality
   * Simple upload:
        Xnatupload -c upload_sheet.csv
   * Upload everything with a session type:
        Xnatupload -c upload_sheet.csv --sess PET
   * Check the upload:
        Xnatupload -c upload_sheet.csv --report
   * Force upload:
        Xnatupload -c upload_sheet.csv --force
   * Upload with delete resource before uploading:
        Xnatupload -c upload_sheet.csv --delete
   * Upload with delete every resources for the object (SCAN/ASSESSOR) before uploading:
        Xnatupload -c upload_sheet.csv --deleteAll
   * Upload BIDS data to XNAT format for scan
        Xnatupload -b /tmp/bidsDataset -p PID
   * Check BIDS data to XNAT
        Xnatupload -b /tmp/bidsDataset -p PID --report
   * Force upload BIDS data to upload XNAT
        Xnatupload -b /tmp/bidsDataset -p PID --force

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host for XNAT. Default: env XNAT_HOST.
  -u USERNAME, --username USERNAME
                        Username for XNAT.
  -c CSV_FILE, --csv CSV_FILE
                        CSV file with the information for uploading data to XNAT.␣
↪Header: object_type,project_id,subject_label,session_type,session_label,as_label,as_
↪type,as_description,as_quality,resource,fpath
  --sess SESSION_TYPE   Session type on Xnat. Use printmodality to see the options.
  --report              Print a report to verify inputs.
  --force               Force the upload and remove previous resources.
  --delete              Delete resource files prior to upload.
  --deleteAll           Delete all resources in object prior to upload.
  --noextract           Do not extract the zip files on XNAT when uploading a folder.
  --printmodality       Display the different modality available on XNAT for a session.
  -o OUTPUT_FILE, --output OUTPUT_FILE
                        File path to store the script logs.
  -b BIDS_DIR, --bids BIDS_DIR
                        BIDS Directory to convert to XNAT and then upload
  -p PROJECT, --project PROJECT
                        Project for BIDS XNAT upload
```

Extra Examples

- Shows on the terminal what kind of data the command is going to upload and where with method 1

```
Xnatupload -d /Path/to/directory --report --up1
```

- Uploads the files in the directory with the first method

```
Xnatupload -p projectID -d /Path/to/directory -up1 -sess MR
```

- Uploads the files in the directory with the second method

```
Xnatupload -p projectID -d /Path/to/directory --up2 --sess CT
```

- Uploads (only, no creation of subject/exp/scan) all the files from the directory into Xnat even if there is already a resources (options -force)

```
Xnatupload -d /Path/to/directory -o -T 1 --force
```

Contact - benjamin.c.yvernault@vanderbilt.edu

XnatReport

Xnatreport will give you a report on one ore more projects. It will show all the subjects/sessions/scans/assessors/resources for the projects chosen. It has several options (accessible with -h or -help) :

```
################################################################
#                            XnatReport                        #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Print a detailed report from XNAT projects.              #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################


----------------------------------------------------------------
usage: XnatReport [-h] [--host HOST] [-u USERNAME] [-p PROJECTS] [-c CSV_FILE]
                  [--format FORMAT] [--printformat]

What is the script doing :
   * Create a report about Xnat projects.

Examples:
   *Report of a project:
        Xnatreport -p PID
   *Report with a specific format:
        Xnatreport -p PID --format object_type,session_id,session_label,age
   *print the format available:
        Xnatreport --printformat
   *Save report in a csv:
        Xnatreport -p PID -c report.csv

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host for XNAT. Default: env XNAT_HOST.
  -u USERNAME, --username USERNAME
                        Username for XNAT.
```

(continues on next page)

(continued from previous page)

```
  -p PROJECTS, --project PROJECTS
                        List of project ID on Xnat separate by a coma
  -c CSV_FILE, --csvfile CSV_FILE
                        csv fullpath where to save the report.
  --format FORMAT       Header for the csv. format: variables name separated by comma.
  --printformat         Print available variables names for the option --format.
```

Extra Examples

- Creates a report for BLSA and CTONS and will print it on the screen/terminal

```
Xnatreport -p BLSA,CTONS
```

- Sends the report on BLSA and CTONS to your email address as a csv file. You need to set to variables gmail address and password used to sent the email in your bashrc

```
Xnatreport -p BLSA,CTONS --csv -e email@email.com
```

- Writes the report as a ".csv" file that can be open with Excel. If not path specify, /tmp is the place where the report is save. -t will do the same but in a text file

```
Xnatreport -p BLSA,CTONS --csv
```

Contact - benjamin.c.yvernault@vanderbilt.edu

XnatSwitchProcessStatus

XnatSwitchProcessStatus is one of the most powerful and used of the Xnat_tools. It allows the user to switch/set the procstatus (job status) for a specific proctype (type of assessor) in a project. XnatSwitchProcessStatus allows the user to change the status of several type of assessors in a project that have a specific type or just for all of them.

```
################################################################
#                     XnatSwitchProcessStatus                  #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Change assessor job/quality control status.              #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################


----------------------------------------------------------------
usage: XnatSwitchProcessStatus [-h] [--host HOST] [-u USERNAME]
                               [--select SELECT] [-x TXT_FILE] [-p PROJECTS]
                               [--subj SUBJECTS] [--sess SESSIONS] [-s STATUS]
                               [-f FORMER_STATUS] [-t PROCTYPES]
                               [-n NEED_INPUTS] [-d] [--qc] [--printstatus]
                               [--fullRegex] [--restart] [--rerun] [--init]
                               [--rerundiskq]

What is the script doing :
```

(continues on next page)

(continued from previous page)

```
    *Switch/Set the status for assessors on XNAT selected by the proctype.

Examples:
    *See status managed by DAX:
        XnatSwitchProcessStatus --printstatus
    *Set all fMRIQA to a specific status Error for a project:
        XnatSwitchProcessStatus -p PID -s Error -t fMRIQA
    *Set all Multi_Atlas that have the status JOB_FAILED to NEED_TO_RUN to have the␣
↪processes run again:
        XnatSwitchProcessStatus -p PID -f JOB_FAILED -t Multi_Atlas -s NEED_TO_RUN
    *Set all VBMQA to NEED_TO_RUN for a project and delete resources:
        XnatSwitchProcessStatus -p PID -s NEED_TO_RUN -t VBMQA -d
    *Set all VBMQA to NEED_TO_RUN, delete resources, and set linked assessors fMRI_
↪Preprocess to NEED_INPUTS:
        XnatSwitchProcessStatus -p PID -s NEED_TO_RUN -t VBMQA -d -n fMRI_Preprocess
    *Set all dtiQA_v2 qa status to Passed for a project:
        XnatSwitchProcessStatus -p PID -s Passed -t dtiQA_v2 --qc
    *Set FreeSurfer for a specific project/subject to NEED_INPUTS:
        XnatSwitchProcessStatus -p PID --subj 123 -s NEED_INPUTS -t FreeSurfer

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host for XNAT. Default: env XNAT_HOST.
  -u USERNAME, --username USERNAME
                        Username for XNAT.
  --select SELECT       Give the assessor label that you want to change the status.
  -x TXT_FILE, --txtfile TXT_FILE
                        File txt. Each line represents the label of the assessor which␣
↪need to change status.
  -p PROJECTS, --project PROJECTS
                        Project ID on XNAT or list of Project ID
  --subj SUBJECTS       Change Status for only this subject/list of subjects.
  --sess SESSIONS       Change Status for only this session/list of sessions.
  -s STATUS, --status STATUS
                        Status you want to set on the Processes. E.G: 'NEED_TO_RUN'
  -f FORMER_STATUS, --formerStatus FORMER_STATUS
                        Change assessors with this former status. E.G: 'JOB_FAILED'
  -t PROCTYPES, --type PROCTYPES
                        Assessor process type you want the status to changed.
  -n NEED_INPUTS, --Needinputs NEED_INPUTS
                        Assessor process type that need to change to NEED_INPUTS because␣
↪the assessors from -t you changed are inputs to those assessors.
  -d, --deleteR         Delete the resources on the assessor.
  --qc                  Change the quality control status on XNAT.
  --printstatus         Print status used by DAX to manage assessors.
  --fullRegex           Use full regex for filtering data.
  --restart             Restart the assessors by switching the status for all assessors␣
↪found to NEED_TO_RUN and delete previous resources.
  --rerun               Rerun the assessors by switching status to NEED_TO_RUN for␣
↪assessors that failed and delete previous resources.
  --init                Init the assessors by switching status to NEED_INPUTS for␣
↪assessors that have been set to NO_DATA.
```

(continues on next page)

(continued from previous page)

```
  --rerundiskq          Rerun the assessor that have the status JOB_FAILED: switching␣
↪status to NEED_INPUTS from JOB_FAILED and delete previous resources.
```

Extra Examples

- Changes the status for dtiQA_v2 and Freesurfer that have a Failed status to NeedToRun in project BLSA

```
XnatSwitchProcessStatus -p BLSA -f Failed -s NeedToRun -t dtiQA_v2,FreeSurfer
```

- Changes the status for dtiQA_v2 and Freesurfer that have a Failed status to NeedToRun in project BLSA and it will delete all the resources on the assessor

```
XnatSwitchProcessStatus -p BLSA -f Failed -s NeedToRun -t dtiQA_v2,FreeSurfer -d
```

- Changes the status for the specific FreeSurfer assessor in BLSA_0000_00 session to NeedToRun and delete the resources

```
XnatSwitchProcessStatus --select BLSA-x-BLSA_0000-x-BLSA_0000_00-x-FreeSurfer -s␣
↪NeedToRun -d
```

Contact - benjamin.c.yvernault@vanderbilt.edu

XnatProcessUpload

Xnatprocessupload allows you to upload data for an assessor (you can't do it that with Xnatupload). You only need to give the path to the folder where the data are. If the assessor doesn't exist, it will create one. You need to organize the data like this :

1) One folder per assessor you want to upload, the name of the folder needs to be the name of the assessor (Remember: assessor label = projectID-x-subjectID-x-sessionID-x-(scanID if running on a only a scan)-x-processname)
2) Put one folder for each resources you want to upload within the assessor folder with the name folder equal to the resource name.
3) Put the file you want to upload in it.

```
########################################################
#                    XNATPROCESSUPLOAD
#
# Developed by the masiLab Vanderbilt University, TN, USA.
# If issues, email benjamin.c.yvernault@vanderbilt.edu
# Parameters :
#     No Arguments given
#     Use "Xnatprocessupload -h" to see the options
########################################################
Usage: Xnatprocessupload [options]
What is the script doing : Upload Data on Xnat from a Directory as an Assessor.

Options:
 -h, --help            show this help message and exit
 -d FOLDER_PATH, --directory=FOLDER_PATH
                       Directory containing the different assessors folders that you␣
↪want to upload.
 --force               Force the upload.
```

Contact - benjamin.c.yvernault@vanderbilt.edu

XnatSubjectUpdate

Xnatsubjectupdate changes the last update date on XNAT to nothing. It will make the automatic process (in cci package when it's setup) to run again on this subject.

```
########################################################
#                   XNATSUBJECTUPDATE
#
# Developed by the masiLab Vanderbilt University, TN, USA.
# If issues, email benjamin.c.yvernault@vanderbilt.edu
# Parameters :
#     No Arguments given
#     See the help bellow or Use "Xnatsubjectupdate" -h
########################################################
Usage: Xnatsubjectupdate [options]
What is the script doing : Query on Xnat at any level.

Options:
 -h, --help            show this help message and exit
 -p PROJECT_ID, --project=PROJECT_ID
                       One project ID on Xnat.
 -s SUBJECT_LABELS, --subject=SUBJECT_LABELS
                       Subject label on Xnat or list of them.
```

Contact - benjamin.c.yvernault@vanderbilt.edu

RedCapReport

Redcapreport is a powertool to extract data from REDCap. It will download the data and put it into a csv file. You can specify different options to have a precise download.

```
################################################################
#                          RedCapReport                        #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Create REDCap report for a redcap project.               #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################

----------------------------------------------------------------
usage: RedCapReport [-h] -k KEY [-c CSVFILE] [-x TXTFILE] [-p PROJECT]
                    [-s SUBJECT] [-e SESSION] [-a ASSESSOR] [-t PROCTYPE]
                    [-f PROCFILE] [-l LIBRARIES] [-F] [-L] [--all]

What is the script doing :
   *Extract data from REDCap as a csv file.
```

(continues on next page)

(continued from previous page)

```
Examples:
   *Save the data in a csv file: Redcapreport -k KEY -c extract_redcap.csv
   *print the libraries name: Redcapreport -k KEY -L
   *print all fields name and label: Redcapreport -k KEY -F
   *Extract values for all record: Redcapreport -k KEY --all
   *Filter for specific project/subject/session/assessor type:
    Redcapreport -k KEY -p PID -s 109387 -e 109387_1,109387_2 -t FS,TRACULA_v1,dtiQA_v2
   *Extract for specific assessor: Redcapreport -k KEY -p PID -a PID-x-109387-x-109387_1-
↪x-FS
   *Extract for specific libraries type: Redcapreport -k KEY -p PID -l library_name
   *Extract only the fields described in the txt file: Redcapreport -k KEY -x fields.txt

optional arguments:
  -h, --help            show this help message and exit
  -k KEY, --key KEY     API Token for REDCap project.
  -c CSVFILE, --csvfile CSVFILE
                        csv file path where the report will be save.
  -x TXTFILE, --txtfile TXTFILE
                        txt file path with per line, the name of the variable
                        on REDCap you want to extract.
  -p PROJECT, --project PROJECT
                        Extract values for processes for the projects chosen.
                        E.G: project1,project2
  -s SUBJECT, --subject SUBJECT
                        Extract values for processes for the subjects chosen.
                        E.G: subject1,subject2
  -e SESSION, --session SESSION
                        Extract values for processes for the sessions chosen.
                        E.G: session1,session2
  -a ASSESSOR, --assessor ASSESSOR
                        Extract values for processors chosen. E.G:
                        processor1,processor2
  -t PROCTYPE, --proctype PROCTYPE
                        Extract values for processes types chosen. E.G:
                        fMRIQA,dtiQA
  -f PROCFILE, --procfile PROCFILE
                        file path with each line one processor label. Extract
                        values for processes types chosen.
  -l LIBRARIES, --libraries LIBRARIES
                        Extract values for only the libraries specify. Check
                        the project for the libraries name. Switch spaces by
                        '_' and everything lower case. E.G:
                        dti_quality_assurance. By default: all libraries
  -F, --fields          Print all field names and labels
  -L, --printlib        Print all libraries names for the project.
  --all                 Extract values for all records.
```

Contact - benjamin.c.yvernault@vanderbilt.edu

XnatCheckLogin

XnatCheckLogin allows the user to check that environment variables are set appropriately. It will let you know in a few seconds if your logins are good or not.

```
usage: XnatCheckLogin [-h] [--host HOST]
Set and Check the logins for XNAT.
optional arguments:
  -h, --help   show this help message and exit
  --host HOST  Host for XNAT.
```

Xnatinfo

Xnatinfo is the tool to get fast statistics information on a project (number of subjects/sessions/scans/assessors and the status of the assessors). There is only one way to call Xnatinfo:

```
################################################################
#                          Xnatinfo                            #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Display information on a XNAT project.                   #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################

----------------------------------------------------------------
usage: Xnatinfo [-h] [--host HOST] [-u USERNAME] [-x OUTPUT_FILE] [-f] [-r]
                [--ignoreUnusable] [--ignoreScans]
                project

What is the script doing :
   * Generate a report for a XNAT project displaying scans/assessors
     information.

Examples:
    * See the information for project TEST:
        Xnatinfo TEST

positional arguments:
  project               Project ID on XNAT

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host for XNAT. Default: env XNAT_HOST.
  -u USERNAME, --username USERNAME
                        Username for XNAT.
  -x OUTPUT_FILE, --filetxt OUTPUT_FILE
                        Path to a txt file to save the report
  -f, --failed          Add this flag to print out failed jobs
```

(continues on next page)

(continued from previous page)

```
  -r, --running         Add this flag to print out running jobs
  --ignoreUnusable      Ignore print statement of unusable scans
  --ignoreScans         Ignore print statement of scans
```

Xnatsessionupdate

Xnatsessionupdate resets the last update date on XNAT on a session. It will force DAX update scripts to update the session. This tool is for advanced users and managers of projects on XNAT.

```
################################################################
#                        XnatSessionUpdate                     #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Reset sessions to be seen by the nex dax_update.         #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################

----------------------------------------------------------------
usage: XnatSessionUpdate [-h] [--host HOST] [-u USERNAME] -p PROJECTS
                         [-s SESSION] [-n] [-x TXT_FILE] [-a]

What is the script doing :
   * Reset sessions last update date to update the sessions during
     the next dax_update.

Examples:
   *Reset all sessions:
        Xnatsessionupdate -p PID --all
   *Reset some sessions :
        Xnatsessionupdate -p PID -s 109374,109348
   *Reset for the sessions that have assessors NEED_INPUTS:
        Xnatsessionupdate -p PID -n

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host for XNAT. Default: env XNAT_HOST.
  -u USERNAME, --username USERNAME
                        Username for XNAT.
  -p PROJECTS, --project PROJECTS
                        Projects ID on Xnat.
  -s SESSION, --session SESSION
                        Session label on Xnat or list of them.
  -n, --needinputs      Change the subject last update date for all the subject with␣
↪processes that have a job status equal to NEED_INPUTS.
  -x TXT_FILE, --txtfile TXT_FILE
                        File txt with at each line the label of the assessor or just the␣
↪Session label where the Subject date need to be changed. E.G for label: project-x-
```

(continues on next page)

(continued from previous page)

```
↪subject-x-experiment-x-scan-x-process_name.
  -a, --all             Change for all sessions.
```

BIDSMapping

BIDSMapping tool allows the user to create, update or replace rules/mapping at the project level on XNAT. These rules are essential as they entail the link between scan type or series description on XNAT to the BIDS datatype, task type and repetition time. XnatToBids function uses these mapping at the project to transform XNAT data into the BIDS compliant data with BIDS filenames and folder structure.

```
################################################################
#                      BIDSMAPPING                             #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Upload rules/mapping to Project level on XNAT.           #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################

usage: use "BIDSMapping --help" for more information

What is the script doing :
   *Uploads BIDS datatype, tasktype and repitition time mapping to XNAT project level␣
↪using the different OPTIONS.

Examples:
   *Create a new datatype mapping for scan_type of XNAT scans:
        BIDSMapping -p PID --xnatinfo scan_type --type datatype --create /tmp/projectID_
↪datataype.csv
   *The correct format for /tmp/projectID_datataype.csv
        scan_type,datatype
        Resting State,func
   *Create a new datatype mapping for series_description of XNAT scans:
        BIDSMapping -p PID --xnatinfo series_description --type datatype --create /tmp/
↪projectID_datataype.csv
   *Create a new tasktype mapping for scan_type of XNAT scans:
        BIDSMapping -p PID --xnatinfo scan_type --type tasktype --create /tmp/projectID_
↪tasktype.csv
   *Replace tasktype mapping for scan_type of XNAT scans: (It removes the old mapping␣
↪and upload the new mapping)
        BIDSMapping -p PID --xnatinfo scan_type --type tasktype --replace /tmp/projectID_
↪tasktype.csv
   *Update tasktype mapping for scan_type of XNAT scans: (This is ONLY add new mapping␣
↪rules, CANT remove rules use --replace to remove and add mapping rules)
        BIDSMapping -p PID --xnatinfo scan_type --type tasktype --update /tmp/projectID_
↪tasktype.csv
   *Create default datatype mapping for scan_type of XNAT scans: (There is no default␣
↪for series_description use --create)
```

(continues on next page)

(continued from previous page)

```
        BIDSMapping -p PID --xnatinfo scan_type --type datatype --create_default
   *Download the current mapping on XNAT:
        BIDSMapping -p PID --xnatinfo scan_type --type datatype --download /tmp/download.
↪csv
   *Download the scan_types on project on XNAT:
        BIDSMapping -p PID --template /tmp/scan_type_template.csv

optional arguments:
  -h, --help            show this help message and exit
  --host HOST           Host for XNAT. Default: using $XNAT_HOST.
  -u USERNAME, --username USERNAME
                        Username for XNAT. Default: using $XNAT_USER.
  -o LOGFILE, --logfile LOGFILE
                        Write the display/output in a file given to this OPTIONS.
  -p PROJECT, --project PROJECT
                        Project to create/update BIDS mapping file
  -t TYPE, --type TYPE  The type of mapping either datatype, tasktype or repetition_time_
↪sec
  -x XNATINFO, --xnatinfo XNATINFO
                        The type of xnat info to use for mapping either scan_type or␣
↪series_description
  -c CREATE, --create CREATE
                        Create the given BIDS new mapping file at project level. (EG. --
↪create <mappingfile>.csv)
                        Default create creates the default mapping at project file. (EG.␣
↪--create)
                        csvfile EG:
                        scan_type,datatype
                        T1W/3D/TFE,anat
                        Resting State,func
  -cd, --create_default
                        Default create creates the default mapping at project file. (EG.␣
↪--create_default)
  -ud UPDATE, --update UPDATE
                        Update the existing BIDS mapping file at project level. (EG. --
↪update <mappingfile>.csv)
                        This option can only add rules
  -rp REPLACE, --replace REPLACE
                        Replace the existing BIDS mapping file at project level. (EG. --
↪replace <mappingfile>.csv)
                        This option can remove and add new rules
  -rv REVERT, --revert REVERT
                        Revert to an old mapping from a specific date/time. (EG: --
↪revert 10-17-19-21:32:15
                        or --revert 10-17-19). Check the LOGFILE at project level for␣
↪the date
  -d DOWNLOAD, --download DOWNLOAD
                        Downloads the current BIDS mapping file (EG: --download
↪<foldername>)
  -tp TEMPLATE, --template TEMPLATE
                        Default mapping template (EG: --template <template file>)
```

For a walkthrough tutorial of BIDSMapping check out https://dax.readthedocs.io/en/latest/BIDS_walkthrough.html

Contact - praitayini.kanakaraj@vanderbilt.edu

XnatBOND

XnatBOND takes in a BIDS directory and detects the Key and Parameter Groups. This tool can be used to Modifying Key and Parameter Group Assignment. For more details on the package used look at https://bids-bond.readthedocs.io/en/latest/readme.html

```
################################################################
#                             XnatBond                         #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Generate and alternate key params in BIDS using BOND     #
# Examples:                                                    #
#     Check the help for examples by running --help            #
################################################################

usage: XnatBOND [-h] --bids_dir BIDS_DIR [-b BOND_DIR] [-m keyparam_edited keyparam_
↪files new_keyparam_prefix] [-o LOGFILE]

What is the script doing :
        *Generate the csv files that have the summary of key groups and param groups␣
↪from the
        bidsdata and modify them in the bids data.

Examples:
        *Generate orginial key and parameter groups:
                XnatBOND --bids_dir BIDS_DIR --bond_dir BOND_DIR
        *Update the key and parameter groups:
                XnatBOND --bids_dir BIDS_DIR --modify_keyparam

optional arguments:
-h, --help            show this help message and exit
--bids_dir BIDS_DIR   BIDS data directory.
-b BOND_DIR, --bond_dir BOND_DIR
                      BOND data directory.
-m keyparam_edited keyparam_files new_keyparam_prefix, --modify_keyparam keyparam_edited␣
↪keyparam_files new_keyparam_prefix
                      Values to modify the keyparam in bids.
-o LOGFILE, --logfile LOGFILE
                      Write the display/output in a file given to this OPTIONS.
```

2.12 DAX Executables

2.12.1 Table of Contents

DAX Packages

We have been developing a high throughput pipeline processing and quality assurance environment based on Washington University's XNAT platform. This system has been deployed as the primary data archival platform for all VUIIS studies. This pipeline has been implemented in a python package called Distributed Automation for XNAT (DAX). Data processing occurs on the Vanderbilt Advanced Computing Center for Research and Education (ACCRE). DAX has been developed with a series of settings making the package portable on any batch scripting system. Each customized module is a spider that performs an image processing task using a variety of open source software.

DAX is available on github at https://github.com/VUIIS/dax and be installed with "pip install dax".

2.12.2 How Does it Work?

DAX consists of three main executables that communicates with an XNAT system to process and archived imaging data. XNAT has an object implemented as a child of a session that is called an Assessor that corresponds to processed data. By reading the database on a project basis, the different executables will generate the assessors, check for inputs, run the scripts on the cluster as a job, check on the status of the jobs, and upload data back to XNAT. DAX will also maintain data on REDCap. DAX uses a settings files to specify various customizations of the DAX installation and to specify which processes each project should run and any customizations to the processes.

2.12.3 DAX Settings

Inside the package DAX, there is a dax_settings.py file. This file contains variables that can be set by the user such as the commands used by your cluster, the different paths (the upload directory, root job, etc. . .), email settings, or REDCap settings for dax_manager.

By default, the package is set to use the settings used by Vanderbilt University. It's set for SLURM cluster.

2.12.4 How to Write a ProjectSettings.yaml File

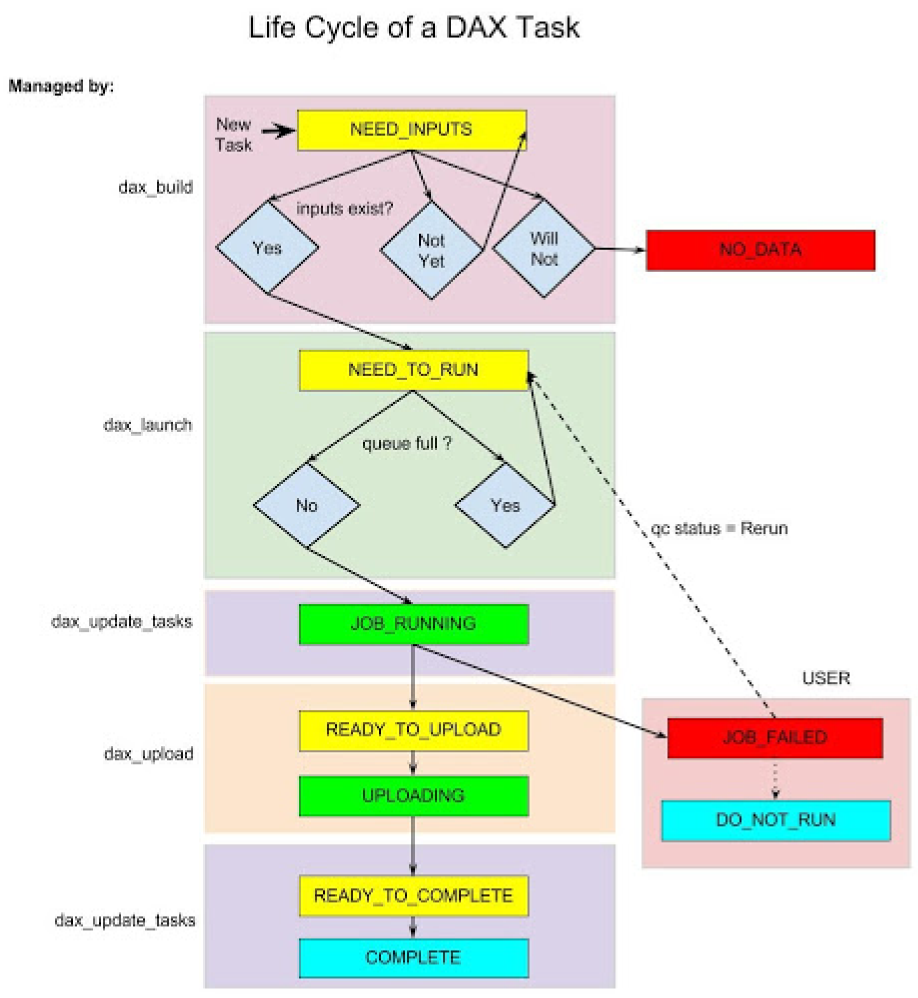
Two of the DAX executables will need a ProjectSettings.py file to run. This file is a python script providing the description of each modules/processors that need to run for a project or a list of projects. You can learn on how to write a ProjectSettings.yaml file here: Writing a settings file.

DAX Executables

The main executables in the DAX package are:

- dax build
- dax update
- dax launch
- dax upload
- dax manager

See image below to understand the role of each executable:



2.12.5 DAX Build

dax build will build all the projects in your ProjectSettings.yaml file. It will check each session of your project and run the different modules (e.g: converting dicom to nifti, generating preview, extracting physlog, etc. . .) and generates the assessors from the processors set in the ProjectSettings.yaml file.

2.12.6 DAX Update

dax update handles assessors for all the projects in your ProjectSettings.yaml file. It will get the list of all the assessors that are "open", meaning with a status from the list below and update each assessors status.

Open assessors status:

- NEED_TO_RUN
- UPLOADING
- JOB_RUNNING
- READY_TO_COMPLETE
- JOB_FAILED

2.12.7 DAX Launch

It will submit jobs to the cluster for each assessors that have the status NEED_TO_RUN.

2.12.8 DAX Upload

Each job on the cluster will not upload data directly to XNAT but copies the data to a temporary folder on the computer. dax upload will read each processed data from this folder and will upload them on XNAT under an assessor that was previously created by dax build.

2.12.9 DAX Manager

dax manager allows users to manage multiple projects from REDCap (https://redcap.vanderbilt.edu). It will automatically generate a ProjectSettings.yaml file from the REDCap database and will run dax build/update/launch/upload from those files.

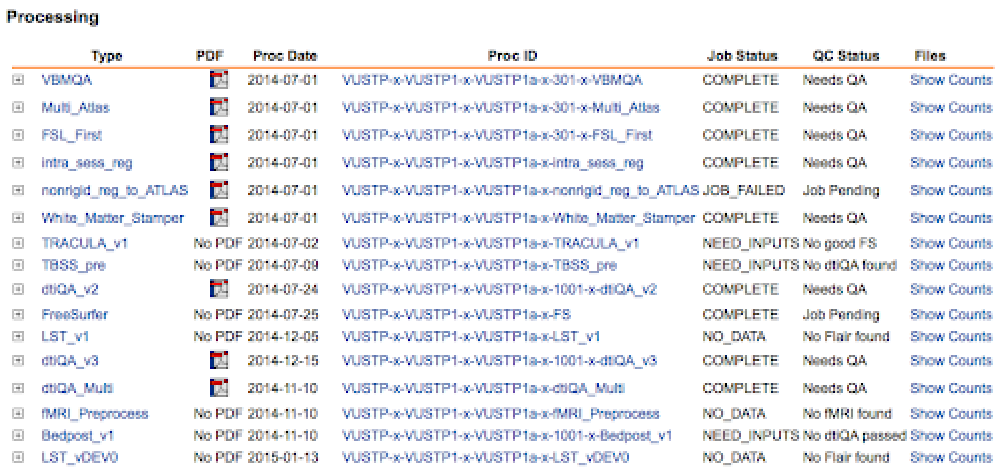
On the REDCap project, each record corresponds to a project. Each library is a module or a processor that can be enabled and customized by the user.

2.13 Manage a Project

2.13.1 Table of Contents

Check Why an Assessor Failed

Each assessor has a procstatus. If you look at a session view and specifically at the assessor list, you can see the column Procstatus.png (see below):



Processing

	Type	PDF	Proc Date	Proc ID	Job Status	QC Status	Files
⊞	VBMQA	📄	2014-07-01	VUSTP-x-VUSTP1-x-VUSTP1a-x-301-x-VBMQA	COMPLETE	Needs QA	Show Counts
⊞	Multi_Atlas	📄	2014-07-01	VUSTP-x-VUSTP1-x-VUSTP1a-x-301-x-Multi_Atlas	COMPLETE	Needs QA	Show Counts
⊞	FSL_First	📄	2014-07-01	VUSTP-x-VUSTP1-x-VUSTP1a-x-301-x-FSL_First	COMPLETE	Needs QA	Show Counts
⊞	intra_sess_reg	📄	2014-07-01	VUSTP-x-VUSTP1-x-VUSTP1a-x-intra_sess_reg	COMPLETE	Needs QA	Show Counts
⊞	nonrigid_reg_to_ATLAS	📄	2014-07-01	VUSTP-x-VUSTP1-x-VUSTP1a-x-nonrigid_reg_to_ATLAS	JOB_FAILED	Job Pending	Show Counts
⊞	White_Matter_Stamper	📄	2014-07-01	VUSTP-x-VUSTP1-x-VUSTP1a-x-White_Matter_Stamper	COMPLETE	Needs QA	Show Counts
⊞	TRACULA_v1	No PDF	2014-07-02	VUSTP-x-VUSTP1-x-VUSTP1a-x-TRACULA_v1	NEED_INPUTS	No good FS	Show Counts
⊞	TBSS_pre	No PDF	2014-07-09	VUSTP-x-VUSTP1-x-VUSTP1a-x-TBSS_pre	NEED_INPUTS	No dtiQA found	Show Counts
⊞	dtiQA_v2	📄	2014-07-24	VUSTP-x-VUSTP1-x-VUSTP1a-x-1001-x-dtiQA_v2	COMPLETE	Needs QA	Show Counts
⊞	FreeSurfer	No PDF	2014-07-25	VUSTP-x-VUSTP1-x-VUSTP1a-x-FS	COMPLETE	Job Pending	Show Counts
⊞	LST_v1	No PDF	2014-12-05	VUSTP-x-VUSTP1-x-VUSTP1a-x-LST_v1	NO_DATA	No Flair found	Show Counts
⊞	dtiQA_v3	📄	2014-12-15	VUSTP-x-VUSTP1-x-VUSTP1a-x-1001-x-dtiQA_v3	COMPLETE	Needs QA	Show Counts
⊞	dtiQA_Multi	📄	2014-11-10	VUSTP-x-VUSTP1-x-VUSTP1a-x-dtiQA_Multi	COMPLETE	Needs QA	Show Counts
⊞	fMRI_Preprocess	No PDF	2014-11-10	VUSTP-x-VUSTP1-x-VUSTP1a-x-fMRI_Preprocess	NO_DATA	No fMRI found	Show Counts
⊞	Bedpost_v1	No PDF	2014-11-10	VUSTP-x-VUSTP1-x-VUSTP1a-x-1001-x-Bedpost_v1	NEED_INPUTS	No dtiQA passed	Show Counts
⊞	LST_vDEV0	No PDF	2015-01-13	VUSTP-x-VUSTP1-x-VUSTP1a-x-LST_vDEV0	NO_DATA	No Flair found	Show Counts

An assessor with the status JOB_FAILED means that the script failed to run on the cluster. To understand why, the user can look at the OUTLOG file under the assessor. If the file is not present, you can check the Uploading queue on your gateway running dax in the OUTLOG folder. When you have located the file, you can see the error generated by the script and try to solve them.

Set/Reset Assessors to Run

If you need to set an assessor to run or reset a large number of assessors to run because they failed, you can use XnatSwitchProcessStatus. We are going to reset all the dtiQA_v2 assessors on our test project VUSTP to NEED_TO_RUN because we want them to rerun:

- XnatSwitchProcessStatus -p VUSTP -s NEED_TO_RUN -t dtiQA_v2 -d

-d means that we want to delete the previous resources. In an other example, we want to run again all the fMRIQA that failed because we fixed the problem:

- XnatSwitchProcessStatus -p VUSTP -s NEED_TO_RUN -t fMRIQA -f JOB_FAILED -d

Sometimes, an assessor is used as an input for an other assessor (TRACULA uses FreeSurfer outputs). If you rerun a FreeSurfer for example on the subject number 1, you might want to set the TRACULA to NEED_INPUTS to wait for FreeSurfer to have the valid inputs to rerun as well. To do so, you can use the options -n following by the proctype:

- XnatSwitchProcessStatus -p VUSTP –subj VUSTP1 -s NEED_TO_RUN -t FS -d -n TRACULA_v1

You should be able now to restart all the jobs you want/need on XNAT.

Run an XnatCheck on Your Project

Xnatcheck is useful to get a list of assessors from XNAT that fit specific criteria. For example, you want to get the list of all the assessors that failed to restart, you can use the following command:

- Xnatcheck -p VUSTP –filters procstatus=JOB_FAILED

The result is the following:

```
################################################################
# XNATCHECK #
# #
# Usage: #
# Check XNAT data (subject/session/scan/assessor/resource) #
# Parameters : #
# Project(s) -> VUSTP #
# Resource Delimiter -> -- #
# filters String -> ['procstatus=JOB_FAILED'] #
################################################################
===================================================================
INFO: Creating your filters from the options.
* regular filter: procstatus = JOB_FAILED

INFO: extracting information from XNAT:
WARNING: extracting information from XNAT for a full project might take some time.
Please be patient.

- VUSTP
INFO: Number of XNAT object found after filters:
-------------------------------------------
| Project ID | Number of Objects |
-------------------------------------------
| VUSTP | 18 |
-------------------------------------------


object_type,project_id,subject_label,session_type,session_label,as_label,as_type,
      as_description,as_quality
assessor,VUSTP,VUSTP1,MR,VUSTP1a,
      VUSTP-x-VUSTP1-x-VUSTP1a-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP3,MR,VUSTP3a,
      VUSTP-x-VUSTP3-x-VUSTP3a-x-T1-x-FSL_First,FSL_First,JOB_FAILED,
      Job Pending
assessor,VUSTP,VUSTP3,MR,VUSTP3a,
      VUSTP-x-VUSTP3-x-VUSTP3a-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP4,MR,VUSTP4a,
```

(continues on next page)

(continued from previous page)

```
      VUSTP-x-VUSTP4-x-VUSTP4a-x-MPRAGE-x-VBMQA,VBMQA,JOB_FAILED,
      Job Pending
assessor,VUSTP,VUSTP4,MR,VUSTP4a,
      VUSTP-x-VUSTP4-x-VUSTP4a-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP5,MR,VUSTP5a,
      VUSTP-x-VUSTP5-x-VUSTP5a-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP6,MR,VUSTP6a,
      VUSTP-x-VUSTP6-x-VUSTP6a-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP7,MR,VUSTP7a,
      VUSTP-x-VUSTP7-x-VUSTP7a-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP8,MR,VUSTP8a,
      VUSTP-x-VUSTP8-x-VUSTP8a-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP8,MR,VUSTP8b,
      VUSTP-x-VUSTP8-x-VUSTP8b-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP9,MR,VUSTP9a,
      VUSTP-x-VUSTP9-x-VUSTP9a-x-LST_v1,LST_v1,JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP9,MR,VUSTP9a,
      VUSTP-x-VUSTP9-x-VUSTP9a-x-LST_vDEV0,LST_vDEV0,JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP9,MR,VUSTP9a,
      VUSTP-x-VUSTP9-x-VUSTP9a-x-MPRAGE-x-VBMQA,VBMQA,JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP9,MR,VUSTP9a,
      VUSTP-x-VUSTP9-x-VUSTP9a-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP9,MR,VUSTP9b,
      VUSTP-x-VUSTP9-x-VUSTP9b-x-LST_v1,LST_v1,JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP9,MR,VUSTP9b,
      VUSTP-x-VUSTP9-x-VUSTP9b-x-LST_vDEV0,LST_vDEV0,JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP9,MR,VUSTP9b,
      VUSTP-x-VUSTP9-x-VUSTP9b-x-MPRAGE-x-VBMQA,VBMQA,JOB_FAILED,Job Pending
assessor,VUSTP,VUSTP9,MR,VUSTP9b,
      VUSTP-x-VUSTP9-x-VUSTP9b-x-nonrigid_reg_to_ATLAS,nonrigid_reg_to_ATLAS,
      JOB_FAILED,Job Pending
===================================================================
```

You can then check the different errors for each assessor and restart the assessors using XnatSwitchProcessStatus as we saw earlier. You can also modify the header of the output to have more information (see available header name with -printformat). For example, to see the walltime and memory used as well as the starting date for the jobs that are COMPLETE for the session VUSTP1a:

- Xnatcheck -p VUSTP –filters procstatus=COMPLETE session_label=VUSTP1a –format assessor_label,proctype,procstatus,walltimeused,memused,jobstartdate

The output now for the csv is:

```
object_type,assessor_label,proctype,procstatus,walltimeused,memused,jobstartdate
assessor,VUSTP-x-VUSTP1-x-VUSTP1a-x-1001-x-dtiQA_v2,dtiQA_v2,COMPLETE,
      17:02:43,3127140,2015-02-04
```

(continues on next page)

(continued from previous page)

```
assessor,VUSTP-x-VUSTP1-x-VUSTP1a-x-1001-x-dtiQA_v3,dtiQA_v3,COMPLETE,
      16:43:45,3135972,2015-02-04
assessor,VUSTP-x-VUSTP1-x-VUSTP1a-x-301-x-FSL_First,FSL_First,COMPLETE,
      00:22:17,1613624,2015-02-04
assessor,VUSTP-x-VUSTP1-x-VUSTP1a-x-301-x-Multi_Atlas,Multi_Atlas,COMPLETE,
      1-10:40:20,5585220,2015-02-04
assessor,VUSTP-x-VUSTP1-x-VUSTP1a-x-301-x-VBMQA,VBMQA,COMPLETE,
      00:20:13,1380344,2015-02-19
assessor,VUSTP-x-VUSTP1-x-VUSTP1a-x-FS,FreeSurfer,COMPLETE, , ,2014-09-22
assessor,VUSTP-x-VUSTP1-x-VUSTP1a-x-White_Matter_Stamper,White_Matter_Stamper,
      COMPLETE,01:57:14,2254504,2015-02-16
assessor,VUSTP-x-VUSTP1-x-VUSTP1a-x-dtiQA_Multi,dtiQA_Multi,COMPLETE,
      16:35:51,3109260,2015-02-04
assessor,VUSTP-x-VUSTP1-x-VUSTP1a-x-intra_sess_reg,intra_sess_reg,COMPLETE,
      00:03:34,318328,2015-02-04
```

Run dax_update Manually on a Project (Advanced Users)

You can run manually dax_update on a project if you want to update directly a session and not wait for the next time it will run. To do so, you will need to use this command line:

- dax_update ProjectSettings.yaml –project PID –sessions S_ID1,S_ID2

Run dax_launch Manually on a Project (Advanced Users)

You can run manually a dax_launch on a project if you want to submit jobs (assessors with the status NEED_TO_RUN) to the cluster and not wait for the next time it automatically runs. To do so, you will need to use this command line:

- dax_launch ProjectSettings.py –project PID –sessions S_ID1,S_ID2

Common and Spurious Errors You May Encounter

PyXNAT is still a work in progress. As such, you may encounter errors that make little to no sense. A common one that you may get is this:

DatabaseError:

2.13.2 Unable to Read Experiments for Project: XXXXXXXX

You can get technical details here. Please continue your visit at our home page. Where XXXXXXX will be your XNAT Project ID (like VUSTP). Chances are likely that users don't have access to your project. It's a quick fix.

2.13.3 Restarting a Job

Jobs can be restarted using XnatSwitchProcessStatus:

- XnatSwitchProcessStatus -s NEED_INPUTS -d –select

Note that you can also switch the process status to NEED_INPUTS in the GUI but the associated data is NOT deleted. Thus, the preferred way is to use XnatSwitchProcessStatus.

2.13.4 Project Settings Files

The dax_project_settings need to specify an attribute change in the processor variables from the project_settings file. Consider the yaml script from the snapshot. To change scan types in a project settings file, we do:

```
- name: multi_atlas_v3_0_0_VUIIS_ABCD
  filepath: Multi_Atlas_v3.0.0_processor.yaml
  arguments:
    inputs.xnat.scans.scan_t1.types: "ABCD_T1W3D"
```

To change the attributes from the "resources" section from the processor, the arguments would be passed thus:

- inputs.xnat.scans.resource.t1_file_fmatch:"*.nii.gz"

and not as

- inputs.xnat.scans.resource.NIFTI.fmatch

2.13.5 Adding Directories Caused by OSError (only relevant to LDAX)

[Errno 2] No such file or directory from CRITICAL messages in past 24 hours email

Usually check /scratch/$USER/Modules_tmp, which is based on the project name, not the file name. For instance, this ginko file may have something like the following:

- OSError: [Errno 2] No such file or directory: '/scratch/vuiisccidev/Modules_tmp/MSSeg2016/MSSeg2016_preview_nifti_ginko_s
- The MSSeg2016 and MSSeg2016/MSSeg2016_preview_nifti_ginko_settings directories would need to be created

2.13.6 Settings Directory is Missing from tmp Folder (only relevant to LDAX)

We need to check REDCap. Settings files should not be in the /tmp/ folder. Normally, they would be somewhere like:

```
'/scratch/vuiisccidev/Modules_tmp/MSSeg2016/MSSeg2016_preview_nifti_ginko_settings'
```

2.13.7 Verifying the Spider is Waiting to get Uploaded to XNAT

- The upload queue is different from the ACCRE queue
- The ACCRE cluster is not involved in the upload process
- Upload happens from the following directory:

```
/scratch/$USER/Spider_upload_dir
```

2.14 BIDSMapping: Walkthrough Tutorial

2.14.1 Introduction

This is a tutorial for using BIDSMapping tool, a DAX command line tool (https://github.com/VUIIS/dax). The BIDSMapping tool allows the user to create, update or replace rules/mapping at the project level on XNAT. For using BIDSMapping tool you require

- the lastest verion of DAX installed. Please check https://dax.readthedocs.io/en/latest/installing_dax_in_a_virtual_environment.html to install DAX in a virtual environment.
- A project on XNAT with imaging data.
- A dcm2niix module turned on for the project. Preferred if the dcm2niix_bids module is turned on for the project. The dcm2niix_bids will add the required json sidecar. However, the BIDSMapping tool is capable of adding the json sidecar when missing.

2.14.2 Table of Contents

Step 1 Mapping Datatype and Scans

You need to create a mapping for BIDS datatype and scans on XNAT. First, create the CSV file of the mapping that you would like to upload to XNAT.

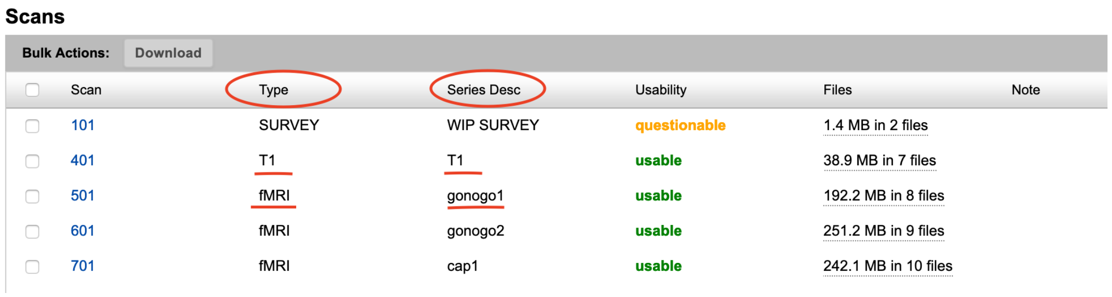
Open a CSV file

```
(dax) $ vim (or nano or any editor you like) datatype.csv
```

Type the series_description and datatype you want to map

```
series_description,datatype
T1,anat
gonogo1,func
gonogo2,func
cap1,func
cap2,func
mid1,func
mid2,func
mid3,func
```

Please note, instead of scan_type in column 1 header series_description can also be used. Make sure the scan_type or series_description is from the scan on XNAT. Image below shows where the information can be found on XNAT


Scans

Bulk Actions: Download

	Scan	Type	Series Desc	Usability	Files	Note
☐	101	SURVEY	WIP SURVEY	questionable	1.4 MB in 2 files	
☐	401	T1	T1	usable	38.9 MB in 7 files	
☐	501	fMRI	gonogo1	usable	192.2 MB in 8 files	
☐	601	fMRI	gonogo2	usable	251.2 MB in 9 files	
☐	701	fMRI	cap1	usable	242.1 MB in 10 files	

Datatype column correspond to the BIDS datatype folder (https://bids.neuroimaging.io/) for the scan to be in. BIDS datatype folder is either - anat (structural imaging such as T1,T2,etc.), - func (task based and resting state functional MRI), - fmap (field inhomogeneity mapping data such as fieldmaps) or - dwi (diffusion weighted imaging). For more information checkout page 4 and 8 in https://www.biorxiv.org/content/biorxiv/suppl/2016/05/12/034561.DC4/034561-1.pdf

Step 2 Upload Datatype Mapping to XNAT

This step allows the user to upload datatype mapping rules to XNAT. These mapping rules are then later used by XnatToBids function to organise the scan from XNAT in the respective BIDS datatype folder. Upload the CSV file (from Step 1) with the mapping rules to XNAT project level using BIDSMapping –create. If scan_type is used as column 1 header in Step 1, use –xnatinfo scan_type option.

```
(dax) $ BIDSMapping --project ZALD_TTS --create datatype.csv --type datatype --xnatinfo␣
↪series_description
```

```
################################################################
#                      BIDSMAPPING                             #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
```

(continues on next page)

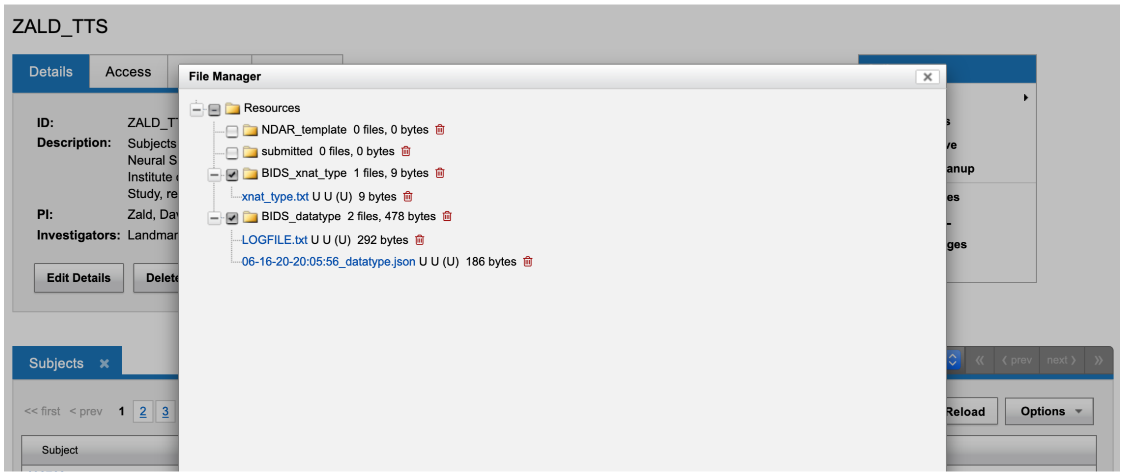
(continued from previous page)

```
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Upload rules/mapping to Project level on XNAT.           #
# Parameters:                                                  #
#     Project ID           -> ZALD_TTS                         #
#     XNAT mapping type    -> series_description               #
#     BIDS mapping type    -> datatype                         #
#     Create mapping with  -> datatype.csv                     #
################################################################

INFO: connection to xnat <http://129.59.135.143:8080/xnat>:
The info used from XNAT is series_description
CSV mapping format is good
date 16-06-20-20:05:56
CREATED: New mapping file 06-16-20-20:05:56_datatype.json is uploaded
```

Step 3 Check Project Level File Manager

Check Manage Files on XNAT project level. There will be two Resources created; one for XNAT type and the other for datatype mapping. XNAT type will have text file with either scan_type or series_description in it. Datatype mapping will have a .json file of the mapping and a LOGFILE.txt with the logging of rules added and deleted.



Resources
NDAR_template 0 files, 0 bytes
submitted 0 files, 0 bytes
BIDS_xnat_type 1 files, 9 bytes
xnat_type.txt U U (U) 9 bytes
BIDS_datatype 2 files, 478 bytes
LOGFILE.txt U U (U) 292 bytes
06-16-20-20:05:56_datatype.json U U (U) 186 bytes

Steps 4 through 8 are ONLY FOR FUNCTIONAL SCANS

Step 4 Mapping Tasktype and Scans

For functional scans, tasktype mapping is necessary. These mapping rules are to map the scan in XNAT to the task. The task refers to the task performed by the subject during the MRI acquisition (For example: rest for resting state). The task could be any activity. The task is required for BIDS filenaming. For more information check out page 11 in https://www.biorxiv.org/content/biorxiv/suppl/2016/05/12/034561.DC4/034561-1.pdf

Similar to Step 1, create tasktype CSV mapping.

```
(dax) $ vim (or nano or any editor you like) tasktype.csv
```

```
series_description,tasktype
gonogo1,gonogo
gonogo2,gonogo
cap1,cap1
cap2,cap2
mid1,mid1
mid2,mid2
mid3,mid3
```

Step 5 Upload Tasktype Mapping to XNAT

This step allows the user to upload tasktype mapping rules to XNAT. The XnatToBids in DAX uses this tasktype mapping to name the funcational scans in the BIDS folder. If there is no tasktype mapping the BIDS conversion will fail for functional scans.

Similar to Step 2, upload the Step 4 CSV mapping to XNAT using BIDMapping tool.

```
(dax) $ BIDSMapping --project ZALD_TTS --create tasktype.csv --type tasktype --xnatinfo␣
↪series_description
```

```
################################################################
#                     BIDSMAPPING                              #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Upload rules/mapping to Project level on XNAT.           #
# Parameters:                                                  #
#     Project ID           -> ZALD_TTS                         #
#     XNAT mapping type    -> series_description               #
#     BIDS mapping type    -> tasktype                         #
#     Create mapping with  -> tasktype.csv                     #
################################################################

INFO: connection to xnat <http://129.59.135.143:8080/xnat>:
The info used from XNAT is series_description
CSV mapping format is good
date 16-06-20-20:12:12
CREATED: New mapping file 06-16-20-20:12:12_tasktype.json is uploaded
```

Step 6 Upload Repetition Time Mapping to XNAT

For functional scan, repetition time (TR) CSV mapping is necessary. This is because there could be some error in the TR found in the NIFTI header or in the JSON sidecar. In order to get the correct TR, we require the user to upload TR and XNAT scan mapping.

```
(dax) $ vim (or nano or any editor you like) repetition_time.csv
```

```
series_description,repetition_time_sec
gonogo1,0.862
gonogo2,0.862
```

Step 7 Upload Repetition Time Mapping to XNAT

This step allows the user to upload TR mapping rules to XNAT. TR value is vital during processing. If there is no repetition time mapping the BIDS conversion will fail for functional scans.

Upload the above Step 6 mapping to XNAT using the BIDSMapping tool

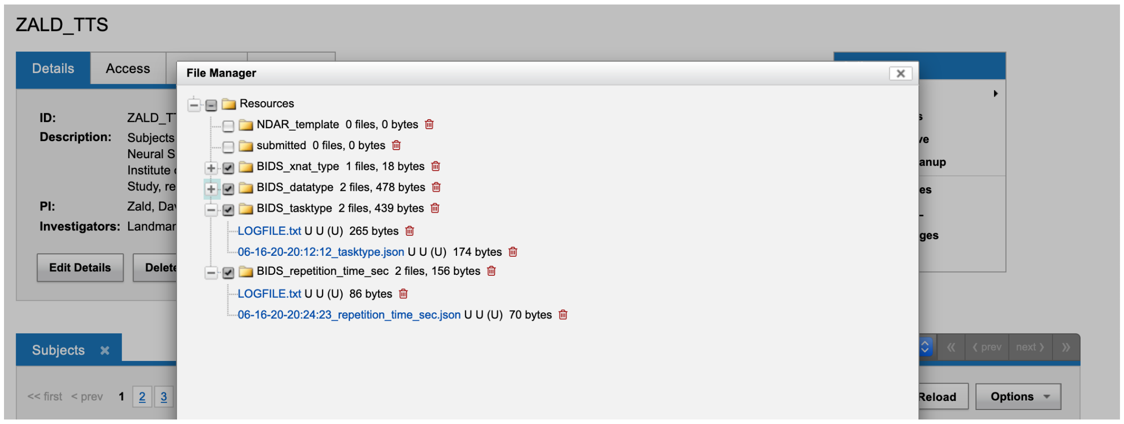
```
(dax) $ BIDSMapping --project ZALD_TTS --create repetition_time.csv --type repetition_
↪time_sec --xnatinfo series_description
```

```
################################################################
#                      BIDSMAPPING                             #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Upload rules/mapping to Project level on XNAT.           #
# Parameters:                                                  #
#     Project ID           -> ZALD_TTS                         #
#     XNAT mapping type    -> series_description               #
#     BIDS mapping type    -> repetition_time_sec              #
#     Create mapping with  -> repetition_time.csv              #
################################################################

INFO: connection to xnat <http://129.59.135.143:8080/xnat>:
The info used from XNAT is series_description
CSV mapping format is good
date 16-06-20-20:15:50
CREATED: New mapping file 06-16-20-20:15:50_repetition_time_sec.json is uploaded
```

Step 8 Check Project Level File Manager

Check Manage Files on XNAT project level. There should be two more BIDS Resources created. One for TR mapping and another for tasktype mapping.



File Manager

```
Resources
  NDAR_template  0 files, 0 bytes
  submitted  0 files, 0 bytes
  BIDS_xnat_type  1 files, 18 bytes
  BIDS_datatype  2 files, 478 bytes
  BIDS_tasktype  2 files, 439 bytes
    LOGFILE.txt U U (U)  265 bytes
    06-16-20-20:12:12_tasktype.json U U (U)  174 bytes
  BIDS_repetition_time_sec  2 files, 156 bytes
    LOGFILE.txt U U (U)  86 bytes
    06-16-20-20:24:23_repetition_time_sec.json U U (U)  70 bytes
```

Step 9 Mapping Perfusion Imaging Type

For perfusion imaging, you need to create a mapping for BIDS perfusion type on XNAT. First, create the CSV file of the mapping that you would like to upload to XNAT.

Open a CSV file

```
(dax) $ vim (or nano or any editor you like) asltype.csv
```

Type the series_description and asltype you want to map

```
series_description,asltype
ASL,asl
pCASL,asl
ASL_m0,m0scan
pCASL_M0,m0scan
```

ASLtype column correspond to the required BIDS naming structure for perfusion imaging type (https://bids.neuroimaging.io/). BIDS datatype folder is either - asl (Perfusion imaging scan such as ASL,CASL,pCASL,pASL,etc.), - m0scan (Reference scan for blood flow calculation. If included in asl image, do not map.),

For more information check out https://docs.google.com/document/d/15tnn5F10KpgHypaQJNNGiNKsni9035GtDqJzWqkkP6c

Step 10 Upload Perfusion Type to XNAT

This step allows the user to upload asltype mapping rules to XNAT. If there is no asltype mapping the BIDS conversion will fail for perfusion scans.

Upload the above Step 9 mapping to XNAT using the BIDSMapping tool

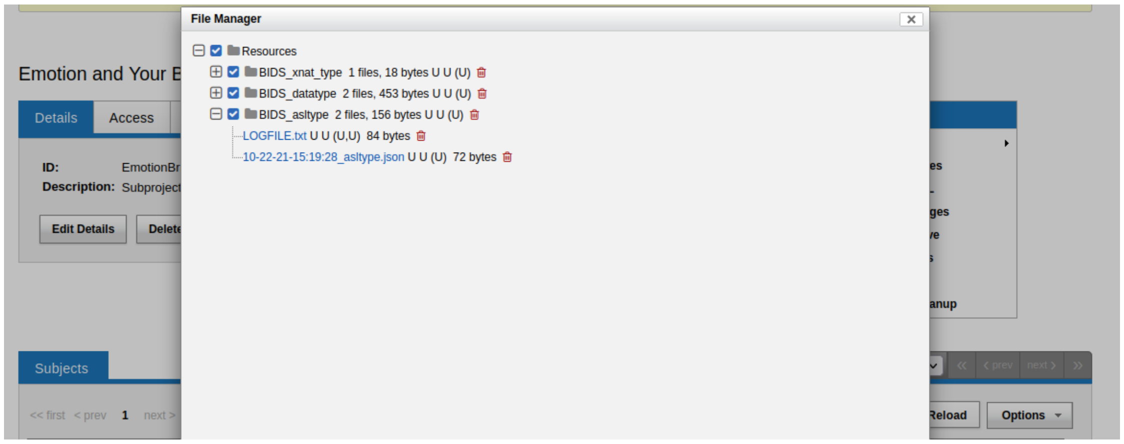
```
(dax) $ BIDSMapping --project ZALD_TTS --create asltype.csv --type asltype --xnatinfo␣
↪series_description
```

```
################################################################
#                     BIDSMAPPING                              #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Upload rules/mapping to Project level on XNAT.           #
# Parameters:                                                  #
#     Project ID           -> EmotionBrain                     #
#     XNAT mapping type    -> series_description               #
#     BIDS mapping type    -> asltype                          #
#     Create mapping with  -> asltype.csv                      #
################################################################

INFO: connection to xnat <http://129.59.135.143:8080/xnat>:
The info used from XNAT is series_description
CSV mapping format is good
date 16-06-20-20:15:50
CREATED: New mapping file 06-16-20-20:15:50_asltype.json is uploaded
```

Step 11 Check Project Level File Manager

Check Manage Files on XNAT project level. There should be one more BIDS Resource created for asltype mapping.



File Manager

Resources
- BIDS_xnat_type 1 files, 18 bytes U U (U)
- BIDS_datatype 2 files, 453 bytes U U (U)
- BIDS_asltype 2 files, 156 bytes U U (U)
  - LOGFILE.txt U U (U,U) 84 bytes
  - 10-22-21-15:19:28_asltype.json U U (U) 72 bytes

2.14.3 Additional Useful BIDSMapping Tool Options

There are additional options such as –replace and –update

- The user can use –replace option to remove existing rules and add new rules. This is useful when the user made a mistake in creating the rules and the rules need to be deleted and replaced by new ones. Please note, the steps 9-11 can be followed for using the option –replace in the BIDSMapping tool.
- The user can use –update option to add new mapping rules to the existing mapping at the project level. This is useful when the user added new scans with new scan types to a project and would like to add mapping rules for these scan types. Please note, the steps 12-14 can be followed for using the option –update in the BIDSMapping tool.

Step 12 Correct Old Mapping

To replace a mapping at project level, create the new CSV mapping. Here, we are replacing repetition_time mapping.

```
(dax) $ vim (or nano or any editor you like) correct_repetition_time.csv
```

```
series_description,repetition_time_sec
gonogo1,2
gonogo2,2
```

Step 13 Replace Existing Mapping

Use option –replace in the BIDSMapping tool. –replace removes the old mapping rules and adds new ones.

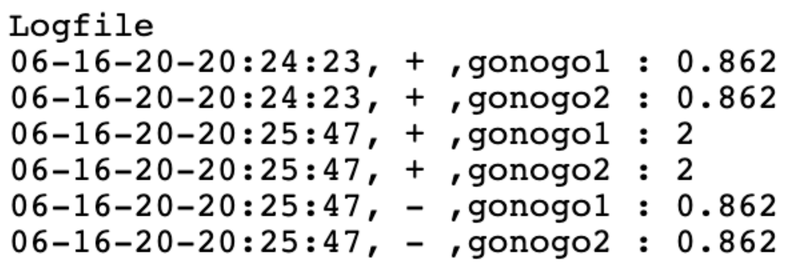
```
(dax) $ BIDSMapping --project ZALD_TTS --replace correct_repetition_time.csv --type␣
↪repetition_time_sec --xnatinfo series_description
```

```
################################################################
#                      BIDSMAPPING                             #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Upload rules/mapping to Project level on XNAT.           #
# Parameters:                                                  #
#     Project ID           -> ZALD_TTS                         #
#     XNAT mapping type    -> series_description               #
#     BIDS mapping type    -> repetition_time_sec              #
#     Create mapping with  -> correct_repetition_time.csv      #
################################################################

INFO: connection to xnat <http://129.59.135.143:8080/xnat>:
The info used from XNAT is series_description
CSV mapping format is good
UPDATED: uploaded mapping file 06-16-20-20:25:47_repetition_time_sec.json
```

Step 14 Check Corrected LOGFILE

Check the LOGFILE.txt or json mapping at the XNAT project level under the repetition time Resources.



```
Logfile
06-16-20-20:24:23, + ,gonogo1 : 0.862
06-16-20-20:24:23, + ,gonogo2 : 0.862
06-16-20-20:25:47, + ,gonogo1 : 2
06-16-20-20:25:47, + ,gonogo2 : 2
06-16-20-20:25:47, - ,gonogo1 : 0.862
06-16-20-20:25:47, - ,gonogo2 : 0.862
```

Step 15 Add New Mapping

To update a mapping at project level, create the new CSV mapping. Here, we are updating repetition_time mapping.

```
(dax) $ vim (or nano or any editor you like) add_new_repetition_time.csv
```

```
series_description,repetition_time_sec
cap1,2
cap2,2
mid1,2
mid2,2
mid3,2
```

Step 16 Update Existing Mapping

Use option –update in the BIDSMapping tool. –update add the new mapping rules to the existing mapping rules.

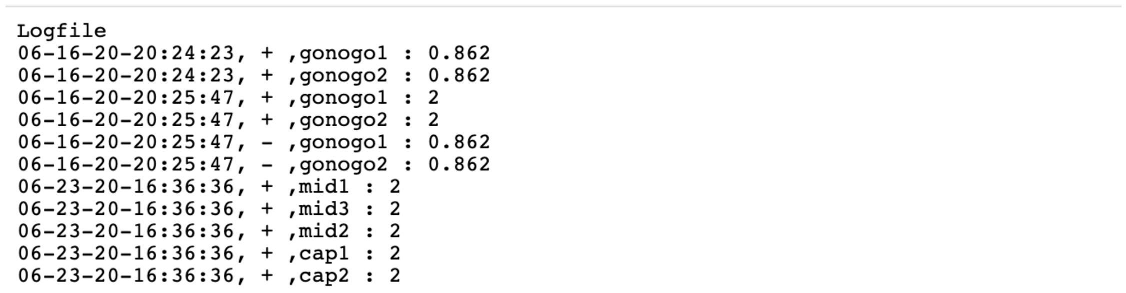
```
(dax) $ BIDSMapping --project ZALD_TTS --update add_new_repetition_time.csv --type␣
↪repetition_time_sec --xnatinfo series_description
```

```
################################################################
#                     BIDSMAPPING                              #
#                                                              #
# Developed by the MASI Lab Vanderbilt University, TN, USA.    #
# If issues, please start a thread here:                       #
# https://groups.google.com/forum/#!forum/vuiis-cci            #
# Usage:                                                       #
#     Upload rules/mapping to Project level on XNAT.           #
# Parameters:                                                  #
#     Project ID            -> ZALD_TTS                        #
#     XNAT mapping type     -> series_description              #
#     BIDS mapping type     -> repetition_time_sec             #
#     Create mapping with  -> add_new_repetition_time.csv      #
################################################################

INFO: connection to xnat <http://129.59.135.143:8080/xnat>:
The info used from XNAT is series_description
CSV mapping format is good
UPDATED: uploaded mapping file 06-23-20-16:36:36_repetition_time_sec.json
```

Step 17 Check Updated LOGFILE

Check the LOGFILE.txt or json mapping at the XNAT project level under the repetition time Resources.



```
Logfile
06-16-20-20:24:23, + ,gonogo1 : 0.862
06-16-20-20:24:23, + ,gonogo2 : 0.862
06-16-20-20:25:47, + ,gonogo1 : 2
06-16-20-20:25:47, + ,gonogo2 : 2
06-16-20-20:25:47, - ,gonogo1 : 0.862
06-16-20-20:25:47, - ,gonogo2 : 0.862
06-23-20-16:36:36, + ,mid1 : 2
06-23-20-16:36:36, + ,mid3 : 2
06-23-20-16:36:36, + ,mid2 : 2
06-23-20-16:36:36, + ,cap1 : 2
06-23-20-16:36:36, + ,cap2 : 2
```

PYTHON MODULE INDEX

INDEX