snsxt Documentation

Release

Stephen Kelly

Nov 20, 2017

Contents

Contents:

CHAPTER 1

Introduction

This is the intro section

CHAPTER 2

snsxt package

2.1 Subpackages

2.1.1 snsxt.config package

2.1.1.1 Module contents

Module to load all of the individual config files Set up internal config dictionary for use throughout the program do any config re-mapping & aggregation here

2.1.2 snsxt.report package

2.1.2.1 Module contents

2.1.3 snsxt.sns_classes package

2.1.3.1 Subpackages

snsxt.sns_classes.config package

Module contents

Configurations module

2.1.3.2 Submodules

2.1.3.3 snsxt.sns_classes.classes module

General utility classes for the program

class snsxt.sns_classes.classes.**SnsAnalysisSample**(*id*, *analysis_config*, *sns_config*, *extra_handlers=None*)

Bases: util.classes.AnalysisItem

Container for metadata about a sample in the sns WES targeted exome sequencing run analysis output

from sns_classes import SnsWESAnalysisOutput import config d = '/ifs/data/molecpathlab/scripts/snsxt/snsxt/fixtures/sns_output/sns_analysis1' x = SnsWESAnalysisOutput(dir = d, id = 'sns_analysis1', sns_config = config.sns) samples = x.get_samples() sample = samples[0] sample.sns_config['analysis_output_index'].items() pattern = sample.id + '.dd.ra.rc.bam' sample.get_output_files(analysis_step = 'BAM-GATK-RA-RC', pattern = pattern)

_init_analysis_attrs(*analysis_config=None*)
Initialize the attributes passed from the parent analysis

_init_dirs(*analysis_config=None*)
Initialize the paths to dirs for the sample in the analysis

_init_files(*analysis_config=None*)
Initialize the paths to files in the analysis

get_output_files(*analysis_step*, *pattern*)
Get a file from the sample's analysis output

class snsxt.sns_classes.classes.**SnsWESAnalysisOutput**(*dir*, *id*, *sns_config*, *results_id=None*, *extra_handlers=None*)

Bases: util.classes.AnalysisItem

Container for metadata about a sns WES targeted exome sequencing run analysis

__init__(*dir*, *id*, *sns_config*, *results_id=None*, *extra_handlers=None*)
Initialize the object

dir = path to the analysis output directory id = ID for the analysis, typically the parent analysis output dir name, corresponding to a NextSeq run ID results_id = typically a time-stamped ID of the results for the analysis, and the subdir name for the anaysis output

e.g.: dir = "/ifs/data/molecpathlab/NGS580_WES/170623_NB501073_0015_AHY5Y3BGX2/results_2017-06-26_20-11-26" id = "170623_NB501073_0015_AHY5Y3BGX2" results_id = "results_2017-06-26_20-11-26"

sns_config = dictionary of configuration items for the run; requires 'analysis_output_index' dict, and 'email_recipients' extra_filehandlers = None or a list of handlers to add

from sns_classes import SnsWESAnalysisOutput import config d = '/ifs/data/molecpathlab/scripts/snsxt/snsxt/fixtures/sns_output/sns_analysis1' x = SnsWESAnalysisOutput(dir = d, id = 'sns_analysis1', sns_config = config.sns)

_init_attrs()
Initialize attributes for the analysis

_init_dirs()
Initialize the paths attributes for items associated with the sequencing run from list of dirnames and filename patterns for the output steps in the sns WES analysis output

_init_files()
Initialize the paths to files that might not have consistent naming including: the targets .bed file with the chromosome target regions

_init_static_files()
Initialize paths to files that should always exist in the same location for an analysis output directory

check_qsub_log_errors_present(*log_files=None*, *err_patterns=('ERROR:',)*)
Check the qsub logs for errors

expected_static_files()
Return a dict of files that are expected to exist in the analysis dir

get_analysis_config()
Return a dictionary of config values to pass to child Sample objects

get_qsub_logfiles(*logdir=None*)
Get the list of log files from the qsub dir

logdir = x.list_none(x.get_dirs('logs-qsub')) log_files = [item for item in find.find(logdir, search_type = 'file')]

get_samples(*samplesIDs=None*)
Get the samples for the analysis samplesIDs is a list of character string sample ID's

get_samplesIDs_from_samples_fastq_raw(*samples_fastq_raw_file=None*)
Get the samples in the run from the samples_fastq_raw file

get_summary_combined_contents(*summary_combined_wes_file=None*)
Get the contents of the 'summary-combined.wes.csv' file

summary_combined_contains_errors(*summary_combined_wes_rows=None*, *err_pattern='X'*)
Check the 'summary-combined.wes.csv' file for errors; any entry in the sheet that looks like 'X' summary_combined_wes_rows = list of dicts read in by CSV DictReader

validate()
Check if the analysis is valid for downstream usage

2.1.3.4 snsxt.sns_classes.test module

Run all the unit tests

2.1.3.5 snsxt.sns_classes.test_sns_classes module

unit tests for the find module

class snsxt.sns_classes.test_sns_classes.**TestAnalysisItem**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_AnalysisItem_files_entry1()
Test a NextSeq demo run that should be valid

test_AnalysisItem_files_entry_listnone()
Test a NextSeq demo run that should be valid

test_AnalysisItem_files_entry_missingkey()
Test a NextSeq demo run that should be valid

test_AnalysisItem_files_type()
Test a NextSeq demo run that should be valid

test_true()

class snsxt.sns_classes.test_sns_classes.**TestSnsWESAnalysisOutput**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_get_bam_dir_exists()

test_get_samples()

test_invalid_path()

test_no_settings()

test_qsub_errors()

test_summary_combined_errors()

test_valid_analysis_output()

2.1.3.6 Module contents

2.1.4 snsxt.sns_tasks package

2.1.4.1 Subpackages

snsxt.sns_tasks.scripts package

Module contents

2.1.4.2 Submodules

2.1.4.3 snsxt.sns_tasks.task_classes module

Updates global configs with task dir paths, then imports the base classes for snsxt analysis tasks used throughout the program

snsxt.sns_tasks.task_classes.**_setup_report**(*func*, **args*, ***kwargs*)
Decorator to set up the analysis task's report files

2.1.4.4 Module contents

Initialize the sns_tasks module and import task classes

2.1.5 snsxt.util package

2.1.5.1 Submodules

2.1.5.2 snsxt.util.classes module

General utility classes for the program

class snsxt.util.classes.**AnalysisItem**(*id*, *extra_handlers=None*)
Bases: *snsxt.util.classes.LoggedObject*

Base class for objects associated with a data analysis

add_file(*name*, *path*)
Add a file to the analysis object's 'files' dict name = dict key paths_list = list of file paths

add_files(*name*, *paths_list*)
Add a file to the analysis object's 'files' dict name = dict key paths_list = list of file paths

get_dirs(*name*)
Retrieve a file by name from the object's 'files' dict name = dict key i = index entry in file list

get_files(*name*)
Retrieve a file by name from the object's 'files' dict name = dict key i = index entry in file list

list_none(*l*)
return None for an empty list, or the first element of a list convenience function for dealing with object's file lists

set_dir(*name*, *path*)
Add a single dir to the analysis object's 'dirs' dict name = dict key path = dict value

set_dirs(*name*, *paths_list*)
Add dirs to the analysis object's 'dirs' dict name = dict key paths_list = list of file paths

set_file(*name*, *path*)
Add a single file to the analysis object's 'files' dict name = dict key path = dict value

set_files(*name*, *paths_list*)
Add a file to the analysis object's 'files' dict name = dict key paths_list = list of file paths

class snsxt.util.classes.**LoggedObject**(*id*, *extra_handlers=None*)
Bases: object

Base class for an object with its own custom logger

Requires an id to be passed extra_handlers should be a list of handlers to add to the logger

get_handler_paths(*logger, types=['FileHandler']*)
Get the paths to all handlers returns a dict of format {name: path}

log_handler_paths(*logger, types=['FileHandler']*)
Log the paths to all handlers

2.1.5.3 snsxt.util.find module

Functions for finding files and dirs

snsxt.util.find.**find**(*search_dir*, *inclusion_patterns=('*',)*, *exclusion_patterns=()*, *search_type='all'*, *num_limit=None*, *level_limit=None*, *match_mode='any'*)
Function to search for files and directories

Parameters

- **search_dir** (*str*) – path to the directory in which to search for files and subdirectories
- **inclusion_patterns** (*list or tuple*) – a list or tuple of patterns to match files/dirs against for inclusion in match output
- **exclusion_patterns** (*list or tuple*) – a list or tuple of patterns to match files/dirs against for exclusion from match output

- **num_limit** (*int*) – the number of matches to return; use *None* for no limit
- **level_limit** (*int*) – the number of directory levels to recurse; 0 is parent dir only
- **match_mode** – 'any' or 'all'; matches any of the provided inclusion_patterns, or all of them
- **search_type** – 'all', 'file', or 'dir'; type of items to find

Returns a list of matching file or directory paths

Return type list

snsxt.util.find.**find_files**(*search_dir*, *search_filename*)
deprecated function that returns the paths to all files matching the supplied filename in the search dir

snsxt.util.find.**find_gen**(*search_dir*, *inclusion_patterns=('*',)*, *exclusion_patterns=()*, *search_type='all'*, *level_limit=None*, *match_mode='any'*)
Generator function to return file matches. Used internally by *find*

Parameters

- **search_dir** (*str*) – path to the directory in which to search for files and subdirectories
- **inclusion_patterns** (*list or tuple*) – a list or tuple of patterns to match files/dirs against for inclusion in match output
- **exclusion_patterns** (*list or tuple*) – a list or tuple of patterns to match files/dirs against for exclusion from match output
- **level_limit** (*int*) – the number of directory levels to recurse; 0 is parent dir only
- **match_mode** – 'any' or 'all'; matches any of the provided inclusion_patterns, or all of them
- **search_type** – 'all', 'file', or 'dir'; type of items to find

snsxt.util.find.**multi_filter**(*names*, *patterns*, *match_mode='any'*)
Generator function which yields the names that match one or more of the patterns.

snsxt.util.find.**super_filter**(*names*, *inclusion_patterns=('*',)*, *exclusion_patterns=()*, *match_mode='any'*)
Enhanced version of *fnmatch.filter()* that accepts multiple inclusion and exclusion patterns.

Filter the input names by choosing only those that are matched by some pattern in *inclusion_patterns* _and_ not by any in *exclusion_patterns*.

Adapted from: https://codereview.stackexchange.com/questions/74713/filtering-with-multiple-inclusion-and-exclusion-patterns

snsxt.util.find.**walklevel**(*some_dir*, *level=1*)
deprecated function that recursively searches a directory for all items up to a given depth

---

Examples

Example usage:

```
file_list = []
for item in pf.walklevel(some_dir):
    if (item.endswith('my_file.txt') and os.path.isfile(item) ):
        file_list.append(item)
```

2.1.5.4 snsxt.util.git module

Functions for finding files and dirs

tested with python 2.7

snsxt.util.git.**parse_git**(*attribute*)
Check the current git repo for one of the following items attribute = "hash" attribute = "hash_short" attribute = "branch"

snsxt.util.git.**print_iter**(*iterable*)
basic printing of every item in an iterable object

snsxt.util.git.**validate_branch**(*allowed=('master', 'production')*)

2.1.5.5 snsxt.util.log module

Functions & items to set up the program loggers

snsxt.util.log.**add_handlers**(*logger*, *handlers*)
Add filehandlers to the logger

snsxt.util.log.**add_missing_console_handler**(*logger*, **args*, ***kwargs*)
Adds a *console* `StreamHandler` if a handler named "console" is not present already in the logger

---

Examples

Example usage:

```
>>> import log
>>> import logging
>>> import qsub
>>> log.has_console_handler(qsub.logger)
False
>>> log.add_missing_console_handler(qsub.logger)
>>> log.has_console_handler(qsub.logger)
True
```

---

snsxt.util.log.**build_console_handler**(*name='console'*, *level=10*, *log_format='[%(asctime)s] %(levelname)s (%(name)s:%(funcName)s:%(lineno)d) %(message)s'*, *datefmt='%Y-%m-%d %H:%M:%S'*)
Returns a basic "console" `StreamHandler`

snsxt.util.log.**build_logger**(*name*, *level=10*, *log_format='[%(asctime)s] %(levelname)s (%(name)s:%(funcName)s:%(lineno)d) %(message)s'*)
Create a basic logger instance Only add console handler by default

snsxt.util.log.**create_main_filehandler**(*log_file*, *name='main'*, *level=10*, *log_format='%(asctime)s:%(name)s:%(module)s:%(funcName)s:%(lineno)d*
Return the 'main' file handler using globally set variables

snsxt.util.log.**email_log_filehandler**(*log_file*, *name='emaillog'*, *level=20*, *log_format='[%(levelname)-8s] %(message)s'*, *datefmt='%Y-%m-%d %H:%M:%S'*)
Return a fileHandler for a log meant to be used as the body of an email

snsxt.util.log.**get_all_handlers**(*logger*, *types=('FileHandler',)*)
Get all logger handlers of the given types from the logger types = ['FileHandler', 'StreamHandler'] x = [h for h in get_all_handlers(logger)]

snsxt.util.log.**get_logger_handler**(*logger*, *handler_name*, *handler_type='FileHandler'*)
Get the filehander object from a logger

snsxt.util.log.**has_console_handler**(*logger*)
Searches a logger's handlers to determine if a `console` handler is present

Parameters **logger** (*logging.Logger*) – a `logging.Logger` object

snsxt.util.log.**log_all_handler_filepaths**(*logger*)
Adds Info log messages for all filepaths for all file handlers

snsxt.util.log.**log_exception**(*logger*, *errors*)
Create a log entry with the errors and traceback

snsxt.util.log.**log_setup**(*config_yaml*, *logger_name*)
Set up the logger for the script using a YAML config file config = path to YAML config file

snsxt.util.log.**logger_filepath**(*logger*, *handler_name*)
Get the path to the filehander log file

snsxt.util.log.**logpath**(*logfile='log.txt'*)
Return the path to the main log file; needed by the logging.yml use this for dynamic output log file paths & names

snsxt.util.log.**print_filehandler_filepaths_to_log**(*logger*)
Make a log entry with the paths to each filehanlder in the logger

snsxt.util.log.**remove_all_handlers**(*logger*, *types=('FileHandler', 'StreamHandler')*)
Remove all of the handlers from a logger object

snsxt.util.log.**remove_handlers**(*logger*, *handlers*)
Removes all the handlers from a logger

snsxt.util.log.**timestamp**()
Return a timestamp string

2.1.5.6 snsxt.util.mutt module

This script provides a flexible wrapper for mailing files from a remote server with mutt

USAGE: mutt.py -s "Subject line" -r "address1@gmail.com, address2@gmail.com" -rt "my.address@internets.com" -m "This is my email message" /path/to/attachment1.txt /path/to/attahment2.txt

example mutt command which will be created: # reply-to field; PUT YOUR EMAIL HERE export EMAIL="kellys04@nyumc.org" recipient_list="address1@gmail.com, address2@gmail.com" mutt -s "$SUBJECT_LINE" -a "$attachment_file" -a "$summary_file" -a "$zipfile" – "$recipient_list" <<E0F email message HERE E0F

snsxt.util.mutt.**get_file_contents**(*file*)
Return a string containing all lines in the file

snsxt.util.mutt.**get_reply_to_address**(*server*)
Get the email address to use for the 'reply to' field in the email needs to be supplied with a server name

snsxt.util.mutt.**make_attachement_string**(*attachment_files*)
Return a string to use to in the mutt command to include attachment files ex: -a "$attachment_file" -a "$summary_file" -a "$zipfile"

snsxt.util.mutt.**mutt_mail**(*recipient_list*, *reply_to=''*, *subject_line='[mutt.py]'*, *message='~ This message was sent by the mutt.py email script ~'*, *message_file=None*, *attachment_files=[]*, *return_only_mode=False*, *quiet=False*)
Main control function for the program Send the message with mutt

recipient_list = character string; Format is 'address1@gmail.com, address2@gmail.com'

snsxt.util.mutt.**run**()
Run the monitoring program arg parsing goes here, if program was run as a script

snsxt.util.mutt.**subprocess_cmd**(*command*)
Runs a terminal command with stdout piping enabled

2.1.5.7 snsxt.util.qsub module

A collection of functions and objects for submitting jobs to the NYUMC SGE compute cluster with *qsub* from within Python, and monitoring them until completion

This submodule can also be run as a stand-alone demo script

class snsxt.util.qsub.**Job**(*id*, *name=None*, *log_dir=None*, *debug=False*)
Bases: object

Main object class for tracking and validating a compute job that has been submitted to the HPC cluster with the *qsub* command

---

Notes

The default action upon initialization is to query *qstat* to determine whether the job is currently running. After a job has completed, built-in methods can be used to query *qacct -j* to determine if the job finished with a successful exit status. Both *qstat* and *qacct* are queried by making system calls to the the corresponding programs and parsing their stdout messages.

Many of the methods included with this object class have stand-alone functions of the same name, with the same usage & functionality.

Examples

Example usage:

```
x = qsub.Job('2379768')
x.running()
x.present()
```

__init__(*id*, *name=None*, *log_dir=None*, *debug=False*)

Parameters

- **id** (*int*) – numeric job ID, as returned by *qsub* at job submission
- **name** (*str*) – the name given to the compute job
- **log_dir** (*str*) – path to the directory used to hold log output by the compute job
- **debug** (*bool*) – intialize the job without immediately querying *qstat* to determine job status

Variables

- ***job_state_key*** (*dict*) – the module's *job_state_key* object
- **id** (*int*) – a numeric ID for the Job object
- **name** (*str*) – a name for the Job

- **log_dir** (*str*) – path to the directory used to hold log output by the compute job
- **log_paths** (*dict*) – dictionary containing the types and paths to the job's output logs
- **completions** (*str*) – character string used to describe the job and its completion states

_completions()
Makes a default 'completions' string attribute

Returns character string describing the object and its qsub log paths

Return type str

_debug_update(*qstat_stdout*)
Debug update mode with requires a qstat_stdout to be passed manually after object initialization

_update()
Update the object's status attributes based on *qstat* stdout messages

error()
Returns *True* or *False* whether or not the job is currently considered to be in an error state

Returns *True* if in error, otherwise *False*

Return type bool

filter_qacct(*qacct_dict=None*, *days_limit=7*, *username=None*)
Filters out 'bad' entries from the *qacct* output dictionary

Parameters

- **qacct_dict** (*dict*) – dictionary containing job records which represent *qacct* entries
- **days_limit** (*int or None*) – Maximum allowed age of a job. Defaults to 7 days, change this to *None* to disable date filtering
- **username** (*str*) – The username which *qacct* records must match, defaults to the current user's name

Returns a dictionary which will hopefully contain only one *qacct* record, hopefully matching the intended compute job

Return type dict

Notes

Filtering is required to remove historic job records from the *qacct* output; only one record can remain in order for the job's completeion status to be determined. This function will try to identify entries which are extraneous and do not represent the intended compute job. The default filtering criteria will first try filter out records that contain usernames which do not match that of the current user. Next, records with a timestamp older than the provided *days_limit* will also be filtered out, in case the current user has multiple job entries for the given *job_id*. Note that the timestamp format used in the *qacct* output is inconsistent, so this type of filtering may be prone to errors.

---

get_is_error(*state*, *job_state_key*)
Checks if the job is considered to in an error state

Returns

Return type bool

get_is_present(*id*, *entry=None*, *qstat_stdout=None*)
Finds out if a job is present in qsub

Returns

Return type bool

get_is_running(*state*, *job_state_key*)
Checks if the job is considered to be running

Returns

Return type bool

get_job(*id*, *qstat_stdout=None*)
Retrieves the job's *qstat* entry

Returns

Return type str

get_log_file(*_type='stdout'*)
Returns the expected path to the job's log file

Parameters **_type** (*str*) – either 'stdout' or 'stderr', representing the type of log path to generate

---

Notes

A stdout log file basename for a compute job with an ID of *4088513* and a name of *python* would look like this: *python.o4088513* The corresponding stderr log name would look like: *python.e4088513*

---

get_qacct(*job_id=None*)
Gets the *qacct* entry for a completed qsub job, used to determine if the job completed successfully

---

Notes

This operation is extremely slow, takes about 10 - 30+ seconds to complete

---

Returns The character string representation of the stdout from the *qacct -j* command for the job

Return type str

get_qacct_job_failed_status(*failed_entry*)
Special parsing for the 'failed' entry in qacct output because its not a plain digit value its got some weird text description stuck in there too

Returns the first int value found after splitting text on the first whitespace found

Return type int

Examples

Example of weird 'failed' entry that needs to be parsed:

```
{'failed': '100 : assumedly after job'}
```

In this case, the value 100 would be returned

---

get_state(*status*, *job_state_key*)
Gets the interpretation of the job's status from the *job_state_key*, e.g. "Running", etc.

Returns

Return type str

get_status(*id*, *entry=None*, *qstat_stdout=None*)
Gets the status of the qsub job, e.g. "Eqw", "r", etc.

Returns

Return type str

present()
Returns *True* or *False* whether or not the job is currently in the *qstat* queue

Returns *True* if present, otherwise *False*

Return type bool

qacct2dict(*proc_stdout=None*, *entry_delim=None*)
Converts text output from *qacct* into a dictionary for parsing

Parameters **entry_delim** (*str*) – character string delimiter to split entries in the *qacct* output, defaults to '=='

Returns a dictionary of individual records containing metadata about the completion status of jobs with the matching *job_id*

Return type dict

Notes

qacct returns multiple entries per *job_id*, because the *job_id* wrap around. So multiple historic jobs with the same *job_id* number will also be returned, delimited by a long string of ===

---

running()
Returns *True* or *False* whether or not the job is currently considered to be running

Returns *True* if running, otherwise *False*

Return type bool

update_completion_validations(*validation_dict*)
Updates the *completion_validations* dict of validation stats with a pretty printed view of the *validations* dictionary, along with the Job's text string representation

update_log_files(*_type='stdout'*)
Updates the paths to the log files in the *log_paths* attribute

validate_completion(*job_id=None*, **args*, ***kwargs*)
Checks if the qsub job completed successfully. Multiple validation criteria are evaluated one at a time, and the results of each are added to a *completion_validations* dictionary attribute along with a verbose description of the criteria. After all the criteria have been evaluated, returns a boolean *True* or *False* to determine if all criteria passed validation. This determines if a compute job is considered to have completed successfully or not.

Returns *True* or *False*, whether or not all job completion validation criteria passed

Return type bool

snsxt.util.qsub.**demo_multi_qsub**(*job_num=3*)
Demo of the qsub code functions. Submits multiple jobs and monitors them to completion.

snsxt.util.qsub.**demo_qsub**()
Demo the qsub code functions

Examples

Example usages:

```
import qsub; job = qsub.submit(log_dir = "logs", print_verbose = True); qsub.
↪monitor_jobs([job], print_verbose = True); job.validate_completion(); print(job.
↪completions)

import qsub; job = qsub.submit(log_dir = "logs", print_verbose = True, monitor =␣
↪True); job.validate_completion()

import qsub; job = qsub.submit(log_dir = "logs", print_verbose = True, monitor =␣
↪True, validate = True)
```

snsxt.util.qsub.**filter_qacct**(*qacct_dict*, *days_limit=7*)
Filters out 'bad' entries from the dict

snsxt.util.qsub.**find_all_job_id_names**(*text*)
Searchs a multi-line character string for all *qsub* job submission messages, where *text* represents the stdout from a series of shell commands where are assumed to have submitted a number of *qsub* jobs (e.g. by an external program)

Parameters **text** (*str*) – a single character string, e.g. representing line(s) of text assumed to be stdout from a shell command that submitted *qsub* jobs

Notes

This function works by parsing the provided text for lines that look like this:

```
Your job 3947957 ("sns.wes.SeraCare-1to1-Positive") has been submitted
```

Examples

Example usage:

```
>>> text = '\n\n process sample SeraCare-1to1-Positive\n\n CMD: qsub -q all.q -
↪cwd -b y -j y -N sns.wes.SeraCare-1to1-Positive -M kellys04@nyumc.org -m a -
↪hard -l mem_free=64G -pe threaded 8-16 bash /ifs/data/molecpathlab/scripts/
↪snsxt/sns_output/test/sns/routes/wes.sh /ifs/data/molecpathlab/scripts/snsxt/
↪sns_output/test SeraCare-1to1-Positive\nYour job 3947957 ("sns.wes.SeraCare-
↪1to1-Positive") has been submitted\n\n'
>>> [(job_id, job_name) for job_id, job_name in find_all_job_id_names(text)]
[('3947957', 'sns.wes.SeraCare-1to1-Positive')]
```

snsxt.util.qsub.**get_job_ID_name**(*proc_stdout*)
Parses stdout text to find lines that match the output message from a *qsub* job submission

Returns *(<job id number>, <job name>)*

Return type tuple

Examples

Example usage:

```
proc_stdout = submit_job(return_stdout = True) # 'Your job 1245023 ("python") has␣
↪been submitted'
job_id, job_name = get_job_ID_name(proc_stdout)
```

snsxt.util.qsub.**get_qacct**(*job_id*)
Gets the qacct entry for a completed qsub job

snsxt.util.qsub.**get_qacct_job_failed_status**(*failed_entry*)
Special parsing for the 'failed' entry in qacct output because its not a plain digit value its got some weird text description stuck in there too sometimes

Examples

Example text that needs parsing:

```
{'failed': '100 : assumedly after job'}
```

snsxt.util.qsub.**job_state_key = defaultdict(<function <lambda>>, {'r': 'Running', 'dr': None, 'qw': 'Waiting', 'Eq**
dictionary containing possible qsub job states; default state is None

format *key: value*, where *key* is the character string representation of the job state provided by *qstat* output, and *value* is a description of the state.

Eqw: *Error*; the job is in an error status and never started running

r: *Running*; the job is currently running

qw: *Waiting*; the job is currently in the scheduler queue waiting to run

t: *None*; ???

dr: *None*; the job has been submitted for deletion and will be deleted

snsxt.util.qsub.**kill_job_ids**(*job_ids*)
Kills qsub jobs by issuing the `qdel` command

Parameters **job_ids** (*list*) – a list of job ID numbers

Examples

Example usage:

```
import qsub
job_ids = ['4104004', '4104006', '4104009']
qsub.kill_job_ids(job_ids = job_ids)
```

snsxt.util.qsub.**kill_jobs**(*jobs*)
Kills qsub jobs by issuing the `qdel` command

Parameters **jobs** (*list*) – a list of `Job` objects

snsxt.util.qsub.**monitor_jobs**(*jobs=None*, *kill_err=True*, *print_verbose=False*, ***kwargs*)
Monitors a list of qsub *Job* objects for completion. Job monitoring is accomplished by calling each job's

present() and *error()* methods, then waiting for several seconds. Jobs that are no longer present in *qstat* or have an error state will be removed from the monitoring queue. The function will repeatedly check each job and then wait, removing absent or errored jobs, until no jobs remain in the monitoring queue. Optionally, jobs that had an error status will be killed with the *qdel* command, or else they will remain in *qstat* indefinitely.

This function allows your program to wait for jobs to finish running before continuing.

Parameters

- **jobs** (*list*) – a list of *Job* objects
- **kill_err** (*bool*) – *True* or *False*, whether or not jobs left in error state should be automatically killed. Its recommened to leave this *True*
- **print_verbose** (*bool*) – whether or not descriptions of the steps being taken should be printed to the console with Python's *print* function

Returns a tuple of lists containing *Job* objects, in the format: *(completed_jobs, err_jobs)*

Return type tuple

Notes

This function will only check whether a job is present/absent in the *qstat* queue, or in an error state in the *qstat* queue; it does not actually check if a job is in a 'Running' state.

If a job is present and not in error state, it is assumed to either be 'qw' (waiting to run), or 'r' (running). In both cases, it is assumed that the job will eventually finish and leave the *qstat* queue, and subsequently be removed from this function's monitoring queue.

Jobs in 'Eqw' error state are stuck and will not leave on their own so must be removed automatically by this function, or killed manually by the end user.

The `jobs` is mutable and passed by reference; this means that upon completion of this function, the original `jobs` list will be depleted:

```
>>> import qsub
>>> jobs = []
>>> len(jobs)
0
>>> for i in range(5):
...     job = qsub.submit('sleep 20')
...     jobs.append(job)
...
>>> len(jobs)
5
>>> qsub.monitor_jobs(jobs = jobs)
([Job(id = 4098911, name = python, log_dir = None), Job(id = 4098913, name =␣
↪python, log_dir = None), Job(id = 4098915, name = python, log_dir = None),␣
↪Job(id = 4098912, name = python, log_dir = None), Job(id = 4098914, name =␣
↪python, log_dir = None)], [])
>>> len(jobs)
0
```

Examples

Example usage:

```
job = submit(print_verbose = True)
completed_jobs, err_jobs = monitor_jobs([job], print_verbose = True)
[job.validate_completion() for job in completed_jobs]
```

snsxt.util.qsub.**qacct2dict**(*proc_stdout*)

Converts text output from qacct into a dictionary for parsing

snsxt.util.qsub.**submit**(*verbose=False*, *log_dir=None*, *monitor=False*, *validate=False*, **args*, ***kwargs*)

Submits a shell command to be run as a *qsub* compute job. Returns a *Job* object. Passes args and kwargs to *submit_job*. Compute jobs are created by assembling a *qsub* shell command using a bash heredoc wrapped around the provided shell command to be executed. The numeric job ID and job name echoed by *qsub* on stdout will be captured and used to generate a 'Job' object.

Parameters

- **verbose** (*bool*) – *True* or *False*, whether or not the generated *qsub* command should be printed in log output
- **log_dir** (*str*) – the directory to use for qsub job log output files, defaults to the current working directory
- **monitor** (*bool*) – whether the job should be immediately monitored until completion
- **validate** (*bool*) – whether or not the job should immediately be validated upon completion
- ***args** (*list*) – list of arguments to pass on to *submit_job*
- ****kwargs** (*dict*) – dictionary of args to pass on to *submit_job*

Returns a *Job* object, representing a *qsub* compute job that has been submitted to the HPC cluster

Return type *Job*

Examples

Example usage:

```
job = submit(command = 'echo foo')
job = submit(command = 'echo foo', log_dir = "logs", print_verbose = True,␣
↪monitor = True, validate = True)
```

snsxt.util.qsub.**submit_job**(*command='echo foo'*, *params='-j y'*, *name='python'*, *stdout_log_dir=None*, *stderr_log_dir=None*, *return_stdout=False*, *verbose=False*, *pre_commands='set -x'*, *post_commands='set +x'*, *sleeps=0.5*, *print_verbose=False*, ***kwargs*)

Internal function for submitting compute jobs to the HPC cluster running SGE by using the *qsub* shell command. Call this function with *submit* instead; args and kwargs will be evaluated here. Creates a *qsub* shell command to be run in a subprocess, submitting the cluster job with a bash heredoc wrapper. Basic format for job submission to the SGE cluster with qsub using a bash heredoc format

Parameters

- **command** (*str*) – shell commands to be run inside the compute job
- **params** (*str*) – extra params to be passed to *qsub*
- **name** (*str*) – the name of the qsub compute job

- **stdout_log_dir** (*str*) – the path to the directory to use for *qsub* log output; if *None*, defaults to the current working directory
- **stderr_log_dir** (*str*) – the path to the directory to use for *qsub* log output; if *None*, defaults to the current working directory
- **return_stdout** (*bool*) – whether or not the function should *return* the stdout of the *qsub* submission subprocess call, its recommened to always leave this set to *True*, otherwise stdout will be printed to program the log output
- **verbose** (*bool*) – whether or not the generated *qsub* command should be printed in program log output
- **pre_commands** (*str*) – commands to run before the *command* inside the qsub job; defaults to 'set -x' in order to provide verbose qsub log output, you can also put environment modulation code here.
- **post_commands** (*str*) – commands to run after the *command* inside the qsub job; defaults to 'set +x'
- **sleeps** (*int*) – number of seconds to *sleep* after submitting a *qsub* job; it is recommened to leave this set to a value >0 in order to avoid overwhelming the job scheduler with requests
- **print_verbose** (*bool*) – print the generated *qsub* command to the console with the Python *print* function (as opposed to logger output)

Returns returns the stdout of the evaluated *qsub* shell command, assuming *return_stdout = True* was passed. Otherwise, returns nothing.

Return type str

Notes

stdout_log_dir and *stderr_log_dir* should have trailing slashes in their paths, and are set to the same path by default using the *log_dir* arg in *submit*

Malformed or nonexistant *stdout_log_dir* and *stderr_log_dir* paths are a common source for compute job failure.

Call this function with *submit* instead.

This function generates a *qsub* shell command in a format such as this:

```
qsub -j y -N "python" -o :"/ifs/data/molecpathlab/scripts/snsxt/snsxt/util/" -e :
↪"/ifs/data/molecpathlab/scripts/snsxt/snsxt/util/" <<E0F
set -x

    cat /etc/hosts
    sleep 10

set +x
E0F
```

The generated shell command will be evaluated by Python *subprocess*, and its stdout messages returned.

---

snsxt.util.qsub.**subprocess_cmd**(*command*, *return_stdout=False*)

Runs a terminal command with stdout piping enabled

Notes

universal_newlines=True required for Python 2 3 compatibility with stdout parsing

snsxt.util.qsub.**validate_job_completion**(*job_id*)
Checks if a qsub job completed successfully

2.1.5.8 snsxt.util.sh module

http://amoffat.github.io/sh/

2.1.5.9 snsxt.util.template module

Template Python script

class snsxt.util.template.**Container**
Bases: object

basic container for information

snsxt.util.template.**main**()
Main control function for the program

snsxt.util.template.**run**()
Run the monitoring program arg parsing goes here, if program was run as a script

2.1.5.10 snsxt.util.test module

Run all the unit tests

2.1.5.11 snsxt.util.test_find module

unit tests for the find module

class snsxt.util.test_find.**TestSuperFilter**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_error()

test_fail()

test_super_filter_all_Eqw()

test_super_filter_all_Eqw_fail()

test_true()

2.1.5.12 snsxt.util.test_qsub module

unit tests for the find module

class snsxt.util.test_qsub.**TestJob**(*methodName='runTest'*)
Bases: unittest.case.TestCase

setUp()

tearDown()

test_debug_init_Job()
Make sure that the 'debug' init setting prevents attributes from being set

test_error()

test_fail()

test_find_all_job_id_names1()
Test that job IDs and names can be parsed from a blob of text

test_get_job1()
Test that a job can be retrieved from qstat_stdout

test_job_Eqw()
Make sure an Eqw job can be identified

test_job_Eqw_not_running()
Make sure an Eqw job is labeled as not running

test_running_job1()
Find running job id = '2495634' self.qstat_stdout_r_Eqw_file

qstat_stdout_r_Eqw_file = "fixtures/qstat_stdout_r_Eqw.txt" with open(qstat_stdout_r_Eqw_file, "rb") as f: qstat_stdout_r_Eqw_str = f.read() from qsub import Job x = Job(id = '2495634', debug = True)

test_true()

test_validate_qacct_killed1()
Test that a job that was killed due to errors does not pass validation

test_validate_qacct_normal1()
Test that a job can be validated from qacct stdout

test_validate_qacct_normal1_too_old()
Test that a job can be validated from qacct stdout

test_validate_qacct_normal_wrongusername()
Test that a job can be validated from qacct stdout

2.1.5.13 snsxt.util.test_tools module

unit tests for the find module

class snsxt.util.test_tools.**TestDirHop**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_cwd_change()

test_cwd_change_fail()

test_true()

class snsxt.util.test_tools.**TestItemExists**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_item_should_exist_any()

test_item_should_exist_dir()

test_item_should_exist_file()

test_item_should_not_exist_file()

test_item_wrong_type()

class snsxt.util.test_tools.**TestNumLines**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_num_lines1()

test_skip()

class snsxt.util.test_tools.**TestSubprocessCmd**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_cmd_echo_stdout()

test_cmd_echo_success()

test_cmd_fail()

class snsxt.util.test_tools.**TestUpdateJSON**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_update_json1()

test_update_missingfile()

class snsxt.util.test_tools.**TestWriteTabularOverlap**(*methodName='runTest'*)
Bases: unittest.case.TestCase

write_tabular_overlap

test_full_overlap()

test_partial_overlap()

test_true()

2.1.5.14 snsxt.util.tools module

General utility functions and classes for the program

class snsxt.util.tools.**Container**
Bases: object

basic container for information

class snsxt.util.tools.**DirHop**(*directory*)
Bases: object

A class for executing commands in the context of a different working directory adapted from: https://mklammler.wordpress.com/2011/08/14/safe-directory-hopping-with-python/

with DirHop('/some/dir') as d: do_something()

class snsxt.util.tools.**SubprocessCmd**(*command*)
Bases: object

A command to be run in subprocess

run_cmd = SubprocessCmd(command = 'echo foo').run()

run(*command=None*)
Run the command, capture the process object

universal_newlines=True required for Python 2 3 compatibility with stdout parsing

snsxt.util.tools.**backup_file**(*input_file*, *return_path=False*, *sys_print=False*, *use_logger=None*)
backup a file by moving it to a folder called 'old' and appending a timestamp use_logger is a logger object to log to

snsxt.util.tools.**compare**(*x*, *y*)

snsxt.util.tools.**copy_and_overwrite**(*from_path*, *to_path*)
copy a directory tree to a new locaiton and overwrite if it already exits

snsxt.util.tools.**item_exists**(*item*, *item_type='any'*, *n=False*)
Check that an item exists item_type is 'any', 'file', 'dir' n is True or False and negates 'exists'

snsxt.util.tools.**json_dumps**(*object*)

snsxt.util.tools.**load_json**(*input_file*)

snsxt.util.tools.**mkdirs**(*path*, *return_path=False*)
Make a directory, and all parent dir's in the path

snsxt.util.tools.**my_debugger**(*vars*)
starts interactive Python terminal at location in script very handy for debugging call this function with my_debugger(globals().copy()) anywhere in the body of the script, or my_debugger(locals().copy()) within a script function

snsxt.util.tools.**num_lines**(*input_file*, *skip=0*)
Count the number of lines in a file TODO: add tests for this one

snsxt.util.tools.**print_dict**(*mydict*)
pretty printing for dict entries

snsxt.util.tools.**print_json**(*object*)

snsxt.util.tools.**reply_to_address**(*servername*, *username=None*)
Get the email address to use for the 'reply to' field in emails

snsxt.util.tools.**subprocess_cmd**(*command*, *return_stdout=False*)

snsxt.util.tools.**timestamp**()
Return a timestamp string

snsxt.util.tools.**update_json**(*data*, *input_file*)
Add new data to an existing JSON file, or create the file if it doesnt exist

snsxt.util.tools.**write_dicts_to_csv**(*dict_list*, *output_file*)
write a list of dicts to a CSV file

snsxt.util.tools.**write_json**(*object*, *output_file*)

snsxt.util.tools.**write_tabular_overlap**(*file1*, *ref_file*, *output_file*, *delim='\t'*, *inverse=False*)
Find matching entries between two tabular files Write out all the entries in 'file1' that are found in the 'ref_file' save entries to the output_file both 'file1' and 'ref_file' must have headers in common inverse = True write out entries in file1 that are not in ref_file

2.1.5.15 Module contents

2.2 Submodules

2.3 snsxt.cleanup module

Functions for cleaning up after an analysis is finished

snsxt.cleanup.**analysis_complete**(*analysis*)
Actions to take after an analysis is done

Parameters **analysis** (SnsWESAnalysisOutput) – object representing output from an *sns wes* analysis pipeline output on which to run downstream analysis tasks

snsxt.cleanup.**save_configs**(*analysis_dir*)
Saves the global configs object to a YAML file in the analysis dir

Parameters **analysis_dir** (*str*) – path to a directory to hold the analysis output

Notes

Some config items are added or modified during program run time, so final configs may not exactly match starting configs set in external config YAML files

2.4 snsxt.job_management module

Functions for custom management of compute cluster qsub jobs

snsxt.job_management.**background_jobs = []**
If an analysis task generated qsub jobs, but did not wait for them to finish, they will be captured in this list and will be monitored to completion when *run_tasks* finishes running all tasks. This way, the program will not exit until all jobs created have finished.

snsxt.job_management.**kill_background_jobs**()
Kills all jobs in the background_jobs

snsxt.job_management.**monitor_validate_background_jobs**()
Monitors the global background_jobs until completion, then validates their completion status.

snsxt.job_management.**monitor_validate_jobs**(*jobs*)
Monitors a list of qsub jobs until completion, then validates their completion status.

Parameters **jobs** (*list*) – a list of of qsub.Job objects

2.5 snsxt.mail module

Sends email output of the pipeline results

snsxt.mail.**check_default_address**(*address*, *server*, *default_key='__self__'*)
Checks if the provided address matches the default_key, and if so, returns a default email address made from the username of the user running the program + the server.

Parameters

- **address** (*str*) – email address(es) in the format 'email1@server.com,email2@server.com'
- **server** (*str*) – email server to use for a default email address
- **default_key** (*str*) – value to use for recognizing when a default address should be returned

Returns either the original address string, or an email address composed of the user's system name + server

Return type str

snsxt.mail.**email_error_output**(*message_file*, **args*, ***kwargs*)
Sends an email in the event that errors occurred during the analysis.

Parameters **message_file** (*str*) – path to a file to use as the body of the email, typically the program's log file

Keyword Arguments

- **subject_line** (*str*) – the subject line that should be used for the email
- **recipient_list** (*str*) – the recipients for the email, in the format ‘‘recipient_list = “user1@server.com,user2@server.com” ‘‘

snsxt.mail.**email_files = []**
This list should contain file paths output by analysis tasks for inclusion as email attachments at the end of a successful analysis pipeline. It should be accessed by other parts of the program external to this module

---

Examples

Example usage:

```
task_output_file = 'foo.txt'
mail.email_files.append(task_output_file)
```

---

snsxt.mail.**email_output**(*message_file*, **args*, ***kwargs*)
Sends an email upon the successful completion of the analysis pipeline. If any `email_files` were set by the program while running, they will be validated and included as email attachments.

Parameters

- **message_file** (*str*) – path to a file to use as the body of the email, typically the program's log file
- **args** (*list*) – a list containing extra args to pass to `email_output()`
- **kwargs** (*dict*) – a dictionary containing extra args to pass to `email_output()`

Keyword Arguments

- **recipient_list** (*str*) – the recipients for the email, in the format ‘‘recipient_list = “user1@server.com,user2@server.com” ‘‘
- **reply_to** (*str*) – email address to use in the ‘Reply To’ field of the email
- **subject_line** (*str*) – the subject line that should be used for the email

snsxt.mail.**sns_start_email**(*analysis_dir*, ***kwargs*)
Emails the user when the sns pipeline starts

Parameters

- **analysis_dir** (*str*) – path to a directory to hold the analysis output
- **kwargs** (*dict*) – dictionary containing extra args to pass to *run_tasks*

snsxt.mail.**validate_email_files**()
Makes sure all the items in the `email_files` list exist and are considered valid for inclusion in email output

---

Notes

Since the email output is sent by an external program such as `mutt`, it is important that file attachments be valid before attempting to include them, since it will be more difficult to ensure that the email is sent successfully.

2.6 snsxt.run module

Runs a series of analysis tasks

Originally designed as an extension to the sns pipeline output, with the flexibility of added ad hoc extra analysis tasks for downstream processing

`snsxt.run.`**`configs`** **= {'analysis_id_file': 'analysis_id.txt', 'tasks_config_dir': 'config', 'report_compile_script': '/home/doc**
The main configurations dictionary to use for settings throughout the program. The *sns_repo_dir* value is modified at program run time, by preprending the *snsxt_dir* path (path to this script's directory). Other dict keys are set at program run time as well, including *snsxt_parent_dir*, *snsxt_dir*, and *extra_handlers*

`snsxt.run.`**`default_probes`** **= '/home/docs/checkouts/readthedocs.org/user_builds/snsxt/checkouts/latest/probes.bed'**
A .bed formatted file to use by default for CNV analysis. Must have only 3 tab-delimited columns.

`snsxt.run.`**`default_targets`** **= '/home/docs/checkouts/readthedocs.org/user_builds/snsxt/checkouts/latest/targets.bed'**
A .bed formatted file to use by default as the target regions for variant calling

`snsxt.run.`**`default_task_list`** **= '/home/docs/checkouts/readthedocs.org/user_builds/snsxt/checkouts/latest/task_lists/de**
The YAML formatted task list containing analysis tasks to be run by default

`snsxt.run.`**`email_logpath`**`()`
Returns the path to the email log file; needed by the logging.yml config file

This generates dynamic output log file paths & names

Returns a Python logging FileHandler object configured with a log file path set dynamically at program run time

Return type logging.FileHandler

`snsxt.run.`**`extra_handlers`** **= [<logging.FileHandler object>, <logging.FileHandler object>]**
Python logging Filehandlers to be passed throughout the program, in order to keep all submodules logging to the same file(s) set by *logpath()* and *email_logpath()*

`snsxt.run.`**`get_task_list`**`(`*task_list_file*`)`
Reads the task_list from a YAML formatted file

Parameters **`task_list_file`** (*str*) – the path to a YAML formatted file from which to read analysis tasks

Returns a dictionary containing the contents of the YAML *task_list_file*

Return type dict

`snsxt.run.`**`logpath`**`()`
Returns the path to the main log file; needed by the logging.yml config file

This generates dynamic output log file paths & names

Returns a Python logging FileHandler object configured with a log file path set dynamically at program run time

Return type logging.FileHandler

`snsxt.run.`**`main`**`(`***kwargs*`)`
Main control function for the program

Parameters **kwargs** (*dict*) – dictionary containing args to run the program, expected to be passed from *parse()* and passed on to *run_sns_tasks()* and *run_sns_tasks()*

Keyword Arguments

- **analysis_id** (*str*) – an identifier for the analysis (e.g. the NextSeq run ID)
- **results_id** (*str*) – a sub-identifier for the analysis (e.g. a timestamp)
- **task_list_file** (*str*) – the path to a YAML formatted file containing analysis tasks to be run
- **debug_mode** (*bool*) – prevents the program from halting if errors are found in qsub log output files; defaults to *False*. *True* = do not stop for qsub log errors, *False* = stop if errors are found
- **fastq_dirs** (*list*) – a list of paths to directories to use as input data locations for a new *sns* analysis. These directories should contain .fastq.gz files within two levels from the top level of the dir (e.g. at most 2 subdirs deep). The .fastq.gz files contained in these directories should keep the exact filenames output by the NextSeq; sample parsing will take place automatically.
- **targets_bed** (*str*) – path to a .bed formatted file to use as the target regions for variant calling
- **probes_bed** (*str*) – path to a .bed formatted file to use as the probes for CNV analysis
- **pairs_sheet** (*str*) – path to a .csv samplesheet to use for matching tumor and normal samples in the paired variant calling analysis steps. See GitHub for example.

snsxt.run.**parse**()

Runs the program based on CLI arguments. arg parsing happens here, if program was run as a script

Returns a dictionary of keyword arguments to pass to *main()*

Return type dict

Examples

Example script usage:

```
snsxt$ snsxt/run.py -d mini_analysis-controls/ -f mini_analysis-controls/fastq/ -
↪a mini_analysis -r results1 -t task_lists/dev.yml --pairs_sheet mini_analysis-
↪controls/samples.pairs.csv_usethis
```

---

snsxt.run.**startup**()

Configures global attributes of other modules, and performs other actions, when the program starts up

2.7 snsxt.setup_report module

Sets up and compiles the parent analysis report for the pipeline output

snsxt.setup_report.**compile_RMD_report**(*input_file*)

Compiles a .Rmd format report using the R script set in the configs.

Returns the tools.SubprocessCmd object for the shell command that was run to execute the report compilation script

Return type *SubprocessCmd*

snsxt.setup_report.**get_main_report_file**()
Gets the path to the main parent report .Rmd file which should be used to compile the analysis report.

Returns the path to the parent .Rmd file to use in compiling the report

Return type str

snsxt.setup_report.**get_report_files**()
Gets the supporting files for the parent analysis report based on the configs. These include files with helper functions, sub-reports, etc.

Returns a list of paths to files that should be used to set up the parent analysis report

Return type list

snsxt.setup_report.**setup_report**(*output_dir*, *analysis_id=None*, *results_id=None*)
setup the main analysis report in the analysis directory by copying over every associated file for the report to the output dir

2.8 snsxt.test module

Runs all the unit tests found throughout the program

2.9 snsxt.validation module

Functions for validating aspects of the pipeline

snsxt.validation.**background_output_files = []**
By default, a task will validated its expected output files upon task completion. However, tasks that submit qsub jobs and do not wait for them to complete will not be able to validate their expected output files. Instead, the paths to those expected files will be collected in this list, and they will be evaluated once all qsub jobs have been monitored to completion and validated.

snsxt.validation.**validate_background_output_files**()
Validates the global `background_output_files` list contents.

snsxt.validation.**validate_items**(*items*)
Runs validations on a list of items

Parameters **items** (*list*) – a list of file or dir paths to be validated

2.10 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

B

C

D

E

F

S

T

U

V

W