European Organization for Nuclear Research
CERN BE/ABP

CERN/**/**
Updated October 2019

PySixDesk

Version 1.0

The running Environment for SixTrack

User's Reference Manual

X. Lu (CSNS, CERN), A. Mereghetti (CERN), L. Coyle (EPFL, CERN)

Abstract

The aim of PySixDesk is to manage and control massive sixtrack simulations starting from a MADX input file.

Geneva, Switzerland
December 18, 2019

Acknowledgement

We would like to thank our colleagues at CERN to help us to find nasty bugs and for a thorough check of the program.

– *...*

Contents

Chapter 1

Introduction

pySixDesk is a novel platform for managing massive simulations using sixtrack which is a single particle tracking code widely used at CERN. It allows to set up and manage job submission, gather results and analyse them. pySixDesk comes as a python library, hence, it can be imported into a python terminal or into custom-made python scripts.

The pySixDesk package could be obtained from github repository:
https://github.com/SixTrack/pysixdesk

The requirements:

The native environment of pySixDesk is CERN's lxplus login service;The guidelines in the following will assume that this is the case.

1. AFS and openAFS for disk storage;
2. kerberos for login and user identification;
3. htcondor, as batch service native at CERN;
4. BOINC, as additional batch service for long-term, large simulation campaigns;
5. sqlite, for the database monitoring the progress of jobs and storing data;
6. mysql, the central database service used to store data;
7. python3, as main language.

Shell set-up

It is recommended to use pySixDesk from the python shell. Please remember to use python3. On lxplus, python3 is available as python3 command, since the default python command uses version 2.7.5.

In order to use the library, it is essential to declare in your live python environment the path where the pysixdesk package can be found. This can be accomplished adding the path to the pysixdesk package to the PYTHONPATH environment variable (in the following, $pysixdesk_path is the full path to pysixdesk),eg:

```
export PYTHON=$PYTHONPATH:$pysixdesk_path/
```

or to add it to the sys.path list, e.g:

```
import sys
sys.path.append(<path_to_pysixdesk>/)
```

1.1 Workflow of a study

In pysixdesk, the program will start from a configuration file named 'config.py'. It contains all the necessary parameters needed by the jobs, such as database type, name of template file, pathes of madx and sixtrack executable, boinc spool directory, scan parameters for sixtrack and so on.

Due to the input files of an acutal sixtrack job are generated by madx, so the progress is divided into two parts. The first part is called preprocess job which will execute madx to generate input files for sixtrack, and other steps such as one-turn job for DA study, merge aperture node into fort.2 for collimation study. The second part is the sixtrack job which will execute the actual sixtrack job.

There are two scripts 'preprocess.py' and 'sixtrack.py' in the program corresponding to these two parts respectively.

preprocess jobs

The workflow of preprocess job is shown in Figure 1.1, all the studies will start from a single *.mask file. The mask file has some placeholders to generate the actual madx input files by replacing the placeholders with the given values.

```
.....
! A Laundau octupole current 20A inj, -570A col
I_MO=%OCT;

!General switch to select collision (0/1)
ON_COLLISION:=0;
!General switch to install bb lens (0/1)
ON_BB_SWITCH:=0;
......
```

For the upper example, the user should set 'self.madx_params['OCT'] = 0' in the config file to generate an acutal madx input file.

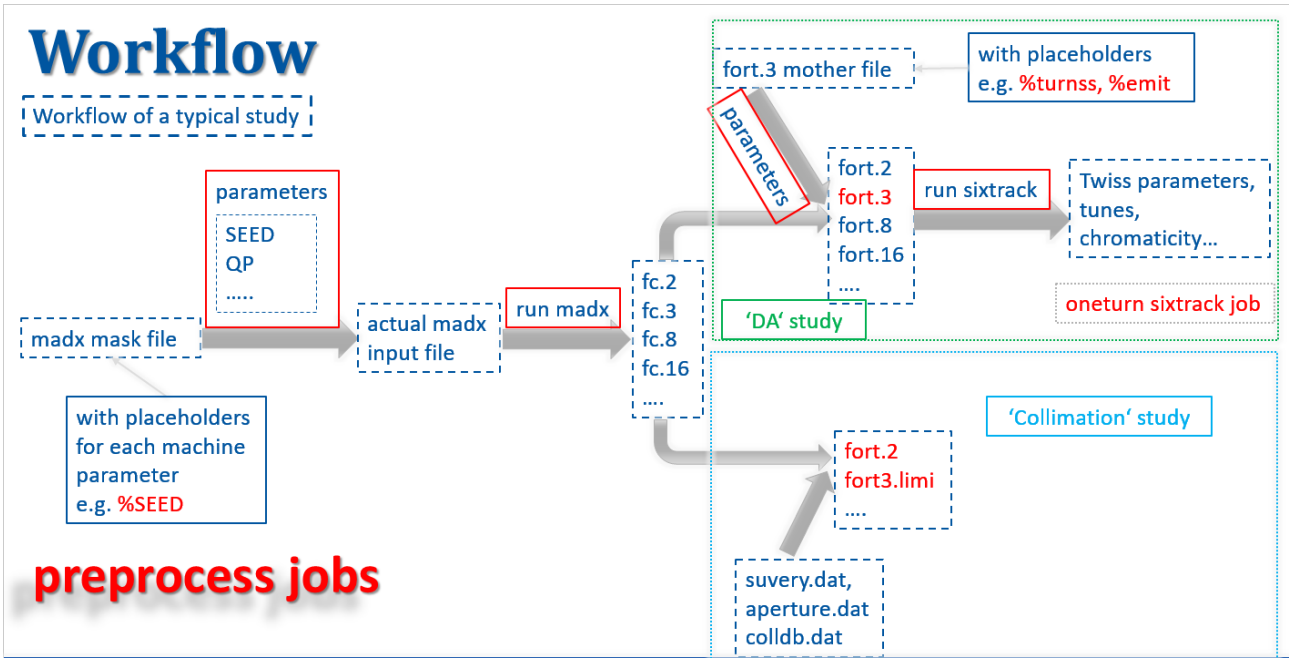

Figure 1.1: The workflow of preprocess job

The program will create the actual madx input file and execute madx to generate files for sixtrack jobs. In general, they will be 'fc.2', 'fc.3', 'fc.8', 'fc.16'. If the user is working on dynamic aperture (DA) study, the one-turn sixtrack job is needed to get the twiss-parameters. If the user is working on collimation study, an additional step to merge aperture model onto fort.2 is needed.

Note that the oneturn step will disappear once the new DIST block in sixtrack is done(which is ongoing), and also the additional step for merging aperture model will disappear once aperture markers and aperture offsets can be consistently generated by MADX.

sixtrack job

After the preprocess jobs are finished, the user can begin to prepare and submit sixtrack jobs. The workflow of sixtrack job is shown in Figure 1.2, the job will start from a template file 'fort.3' with some placeholders. The program will replace the placeholders with the corresponding values defined in config.py to generate the actual input file 'fort.3', the execute sixtrack to generate the required results.

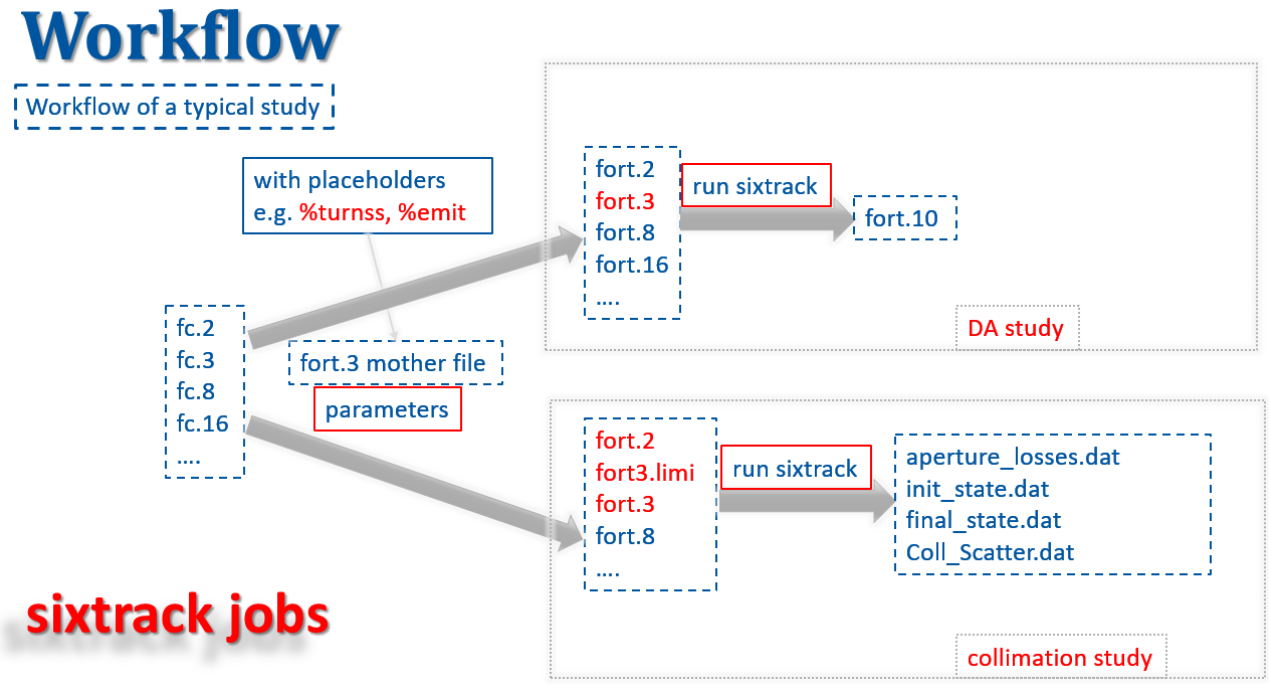

Figure 1.2: The workflow of sixtrack job

The detailed command to manage these steps will introduce in the next chapter.

parameter scan

For both madx job and sixtrack job, the user can scan the paratemeter conveniently in pysixdesk. Define the paratemeters for scan as list, and by default the program will generate all the studies over a Cartesian grid. For example, if the user is working on DA study and going to do the phase space scan:

```
...
self.sixtrack_params['amp'] = [(8,10), (10, 12)]
self.sixtrack_params['kang'] = [1,2]
...
```

And the program will insert 4 new lines in the table sixtrack_wu with:

```
(...(8,10), 1...), (...(8,10), 2...), (...(10,12), 1...), (...(10,12), 2...)
```

The user can also use a custom algorithm to do the scan except for Cartesian grid, just override the 'custom_product_preprocess' and 'custom_product_sixtrack' method with a new algorithm in the config.py file.

1.2 Database

In pySixDesk, sqlite3 and mysql are employed as the main databases to store the data. sqlite3 is the local file-based db, and mysql is the server-based db. For sqlite3 database, it will sit in your study path with an unified name **data.db**, and for mysql database, we use the DB-On-Demand server which is managed by CERN IT. To connect to mysql, the username, password, host, port are needed at least. For security reason, the **config.py** doesn't hold these information, the user should prepare a config file **.my.cnf** under the home directory **($HOME)** which contains the information, it looks like:

```
[client]
user = test1
password = test
host = 127.0.0.1
port = 3306
```

Tables

There are several tables for a study in the database.

```
boinc_vars # for boinc variable
env # for some general valriables
templates # template files, such as *.mask, fort.3
preprocess_wu # parameters for preprocess job
preprocess_task # records for all the preprocess tasks (each submission)
oneturn_sixtrack_wu # parameters for one-turn parameters
sixtrack_wu # parameters for sixtrack job
sixtrack_task # records for all the sixtrack task (each submission)

oneturn_sixtrack_results # results of one turn sixtrack jobs
six_results # results of actual sixtrack jobs
#some general info of the tracking
init_state
final_state
aperture_losses
#results of collimation study
collimation_losses
```

The detailed description of the tables can be found in https://github.com/SixTrack/pysixdesk/blob/master/doc/Table.md

1.3 Special features

There are also some special features in pysixdesk:

1. submit from local computer. Due to HTCondor have the spooling ability, so the user can submit jobs from a local computer. To use this feature, the user should setup kerberos and HTCondor on a local computer at first. Here is a sample instruction: https://twiki.cern.ch/twiki/bin/view/ABPComputing/LxbatchHTCondor.

   Note that the user should submit jobs with '-spool' option from a local computer.

2. group the jobs. pysixdesk could group the jobs by the given parameter. e.g. group by amplitude, then each group will have all scanned values of amplitude but same values of other parameters. And a group will be submitted to one HTCondor node.

3. prolong tracking. pysixdesk could prolong the tracking with checkpoint/restart feature. There are three flag in config.py to switch this feature, 'self.checkpoint_restart = False', 'self.first_turn=1', 'self.last_turn=100'. If checkpoint_restart is set to True, the program will find the checkpoint files (if exist) and prolong the complete job.

Note that the user should make sure the sixtrack executable in use is compiled with the CR option.

1.4 Tests

Below is a scalability test result for different databases. The test condition: 300 preprocess jobs, 30000 sixtrackjobs. All operations were done on CERN lxplus and all jobs were submitted to HTCondor.

Table 1.1: Scalability tests for these two databases.

Action/DB	**Sqlite3**	**Mysql**
`setup DB`	5s	11s
`prepare preprocess`	1s	1s
`submit`	6s	27s
`collection`	120s	-
`prepare sixtrack`	240s	128s
`submit`	435s	294s
`collection`	3h	-

Where '-' represent no need. Note that if the user select mysql and submit jobs to BOINC, the collection process is still needed. For the moment, the volunteer computers can't access to the mysql server directly.

Chapter 2

Detailed Guide

2.1 Step-by-Step Guide

This section will introduce the detailed steps and logics to setup a workspace and an actual study.

2.1.1 set up the worksapce

After importing all the necessary modules from pysixdesk, a workspace could be setup by the following command:

```
myWS = WorkSpace('./myWS')
```

where **'./myWS'** is the workspace name, this step will create a new folder with the following tree structure:

```
myWS/
├─ studies/
└─ templates/
   ├─ hl10.mask
   ├─ fort.3
   ├─ config.py
   ├─ htcondor_run.sub
   └─ CollDB.data
```

The studies folder contains different studies. A new study folder will be added when the user initialize a study.

All the files under templates folder were copied from the templates folder in pysixdesk program by default. The user can use different templates as needed by assigning the argument 'templates' to a custom path:

```
myWS = WorkSpace('./myWS', templates = <your path>)
```

The program will copy templates from <your path>, so in <your path> there should have 'config.py', 'mask file', 'fort.3' and 'htcondor_run.sub' at least.

2.1.2 set up a study

After setting up the workspace, a study could be initialized by the following command:

```
myWS.init_study('myStudy')
```

This action will create a new study folder under 'studies' folder with the following tree structure:

```
studies/
└─ myStudy/
   ├─ config.py
   ├─ hl10.mask
   ├─ fort.3
   ├─ htcondor_run.sub
   ├─ CollDB.data
   ├─ preprocess_input/
   ├─ preprocess_output/
   ├─ sixtrack_input/
   └─ sixtrack_output/
```

Note that the template files under myStudy folder were copied from the templates folder in workspace (myWS). And similarly the user can set up a study with custom template files:

```
myWS.init_study('myStudy', templates=<your path>)
```

The preprocess_input (sixtrack_input) folder holds all the input information for preprocess (sixtrack) jobs, and preprocess_output (sixtrack_output) holds the results of preprocess (sixtrack) jobs.

2.1.3 Config parameters

After initilizing a study, the user should edit the 'config.py' to prepare the parameters. The editable parameters are shown as below:

```
self.paths['madx_exe']  # the path of madx executable
self.paths['sixtrack_exe'] # the path of sixtrack executable
self.paths['boinc_spool'] # the path of boinc spool
self.oneturn = True # switch for one turn sixtrack job
self.collimation = False # switch for preprocess job for collimation study (merge
    aperture marker into fort.2, generate fort3.limi file)

self.checkpoint_restart = False # flag for CR feature
self.first_turn = 1 # the first turn when sixtrack job continue with checkpoint file
self.last_turn = 100 # the last turn of the sixtrack tracking

self.db_info['db_type'] = 'sql' # database type 'sql' or 'mysql'
self.max_submitjob = 15000 # the maximum number of jobs to submit per cluster id
```

```
#The keys are the names of madx output files, the values are the names of the input
    files needed by sixtrack. Due to they have the different naming covention, so
    there is an additional step to change the names of the files based on key-value
    map.
self.madx_output = {
     'fc.2': 'fort.2',
     'fc.3': 'fort.3.mad',
     'fc.3.aux': 'fort.3.aux',
     'fc.8': 'fort.8',
     'fc.16': 'fort.16',
     'fc.34': 'fort.34'}
self.madx_input['mask_file'] = 'hl10.mask'
self.madx_params = {} # The parameters for madx
```

```
#e.g.
self.madx_params['SEED'] = [1,2,3]
self.madx_params['IOCT'] = [100, 200]
self.madx_params['QP'] = [1,2,3]
.....
#Every placeholder in mask file should be found in self.madx_params dict.
```

```
self.oneturn_sixtrack_input['fort_file'] = 'fort.3'
self.oneturn_sixtrack_params = {}
```

```
#preprocess_output will add 'oneturnresult' automatically.
self.preprocess_output = dict{self.madx_output}
```

```
self.sixtrack_input['fort_file'] = 'fort.3'
self.sixtrack_params = {} # parameters for sixtrack
#e.g.
self.sixtrack_params['amp'] = [(8,10), (10,12)]
self.sixtrack_params['kang'] = [1,2,3,4]
self.sixtrack_params['turnss'] = 100
self.sixtrack_params['nss'] = 1
self.sixtrack_params['e0'] = 7000000
......
# Every placeholder in template 'fort.3' should be found in self.sixtrack_params dict
    .

self.sixtrack_input['input'] = dict(self.preprocess_output)
#The outputs of preprocess job will be the inputs for sixtrack job
self.sixtrack_input['additional_input'] = [] # additional input files needed by
    sixtrack jobs if any
```

For preprocess job for collimation study:

```
self.collimation_input = {'aperture':'allapert.b1', 'survey':'
    SurveyWithCrossing_XP_lowb.dat'}
```

```
# some other general parameters
self.env['emit'] = 3.75
self.env['gamma'] = 7460.5
self.env['kmax'] = 5
```

Settings of boinc:

```
self.paths['boinc_spool'] = '/afs/cern.ch/work/b/boinc/boinc/'
self.boinc_vars['workunitName'] = 'pysixdesk'
self.boinc_vars['fpopsEstimate'] = 30 * 2 * 10e5 / 2 * 10e6 * 6
self.boinc_vars['fpopsBound'] = self.boinc_vars['fpopsEstimate'] * 1000
self.boinc_vars['memBound'] = 100000000
self.boinc_vars['diskBound'] = 200000000
self.boinc_vars['delayBound'] = 2400000
self.boinc_vars['redundancy'] = 2
self.boinc_vars['copies'] = 2
self.boinc_vars['errors'] = 5
self.boinc_vars['numIssues'] = 5
self.boinc_vars['resultsWithoutConcensus'] = 3
self.boinc_vars['appName'] = 'sixtrack'
self.boinc_vars['appVer'] = 50205
```

```
# Select the cluster to submit jobs, HTCondor in default
self.cluster_class = submission.HTCondor
```

And a custom cluster could be used to submit jobs. In order to use it, please make sure that the file containing the custom cluster class which extends from submission.Cluster in the path PYTHONPATH. Then just assign the cluster_class attribute in config.py to the desired class:

```
import cluster
...
def __init__(self, name='study', location=os.getcwd()):
    super(MyStudy, self).__init__(name, location)
    self.cluster_class = cluster.custom
```

2.1.4 Load study

After the user prepare all the necessary , a study object could be loaded with the command:

```
mystudy = myWS.load_study('myStudy')
```

By defaults, the program will find the configuration file 'config.py' under study path to create the study object with the default class name 'MyStudy'. But, it is also possible to create the study object with a custom configuration file and a different class name:

```
mystudy = myWS.load_study('myStudy', module_path=<config file path>, class_name=
    custom_name)
```

During the process of creating study object, a database will be set up and the necessary directories and files will aslo be created. The concerned steps could be found in the class method 'customize'.

2.1.5 Update database

In section 2.1.4, a study object was obtained from the load_study method. Now the user can update the DB with the given parameters:

```
mystudy.update_db()
```

For the db table 'templates', 'boinc_vars' and 'env', the program will update the new values (overide old values) from the configuration file whenever the user call the update_db method. And for the table 'preprocess_wu', 'sixtrack_wu', the program will check the parameter changes at first to avoid duplicate records and create new lines to insert the new values.

2.1.6 Prepare input files

Before submitting jobs to batch-system, the user should prepare the necessary input files at first, such as 'htcondor_run.sub', 'job_id.list', 'sub.db'(for sqlite3). This can be done issuing the following command:

```
mystudy.prepare_preprocess_input()
```

This action will query all the incomplete jobs from db and write the corresponding task_ids into job_id.list. If the user use sqlite3 as the db, a small local db 'sub.db' will also be formed to store the necessary information needed by the jobs, this db file will be submitted to HTCondor together with the jobs.

If by any chance the jobs are removed or disappear, e.g. the user realizes wrong paratemeters are set after submission and remove all the submitted jobs via command 'condor_rm'. The user can resubmit the jobs by the following command:

```
mystudy.prepare_preprocess_input(resubmit=True)
```

This action will submit again all jobs.

For sixtrack jobs, the command is:

```
mystudy.prepare_sixtrack_input()
or
mystudy.prepare_sixtrack_input(resubmit=True)
```

Sixtrack jobs can be submitted to Boinc with the following command:

```
mystudy.prepare_sixtrack_input(boinc=True)
```

Please note that the jobs will be anyway submitted to HTCondor at first, and executed with a few turns to check correctness of the configuration and survival of particles within the very first turns automatically. If the job pass the test, it will be submitted to Boinc.

Another special feature of sixtrack job is that the jobs could be grouped based on a specified parameter:

```
mystudy.prepare_sixtrack_input(groupby='amp')
```

This action will group the jobs wrt amplitude, and submit the groups to HTCondor which means one condor node will hold a group of jobs.

pre-calculation

For the DA studies, there are some preliminary calculations are performed before submitting the actual jobs, the namely calculations are defined in 'config.py' file and can be modified at user's will:

```
def pre_calc(self, paramdict, pre_id):
    '''Further calculations for the specified parameters'''
    # The angle should be calculated before amplitude
    keys = list(paramdict.keys())
    status = []
    status.append(self.formulas('kang', 'angle', paramdict, pre_id))
    status.append(self.formulas('amp', ['ax0s', 'ax1s'], paramdict, pre_id))
    param_keys = list(paramdict.keys())
    [paramdict.pop(key) for key in param_keys if key not in keys]
    return all(status)

def formulas(self, source, dest, paramdict, pre_id):
    '''The formulas for the further calculations,
    this function should be customized by the user!
    @source The source parameter name
    @dest  The destination parameter name
    @paramdict The parameter dictionary, the source parameter in the dict
    will be replaced by destination parameter after calculation
    @pre_id The identified preprocess job id
    @return The status'''
```

2.1.7 Submission

If preparation of input files is succeeded, the user could find 'htcondor_run.sub', 'job_id.list', 'input.ini' and 'sub.db'(only for sqlite3) under preprocess_input (for MADX/one-turn jobs) or sixtrack_input (for actual sixtrack jobs) folder, then the user could submit the prepared jobs with the following command:

```
mystudy.submit(0, 5) #0 stand for preprocess job, 5 is the maximum number of trials
or
mystudy.submit(1, 5) # 1 stand for sixtrack job
```

This action will submit the preprocess or sixtrack jobs to HTCondor.

Note: HTCondor has a limitation, the maximum number of jobs presently set of 15000 for one cluser Id. If the total number of jobs exceeds this limitation, the submission will fail. There is an argument 'max_jobsubmit' in config file to control the maximum job number of one submission. If the total number of jobs is larger than max_jobsubmit, the program will split all the jobs in several clusers.

2.1.8 Collection

For sqlite3 DB, the user can collect the results with the following command:

```
mystudy.collect_result(0)
or
mystudy.collect_result(1)
```

This action will look through the preprocess_output (for preprocess jobs) or sixtrack_output (for actual sixtrack jobs) folder to get the results downloaded from HTCondor. In general, the subfolder name is the task_id or grouped task_ids. Before looking through the result folder, the program will query the job status (condor_q) firstly to check which jobs are not finished yet.

And due to the program could submit jobs to HTCondor from a local computer with 'spool' option.

```
mystudy.submit(1, '-spool')
```

So the program will also try to download results from HTCondor spool directory after checking the job status.

```
unfin = cluster.check_running(studypath)#clusterId.processId
......
cluster.download_from_spool(studypath)
```

If the user submit jobs to boinc, results can be downloaded from the boinc spool directory with the following codes:

```
if ('boinc' in cf['info'].keys()) and cf['info']['boinc']:
    content = "Downloading results from boinc spool!"
    logger.info(content)
    task_ids = download_from_boinc(info_sec)
```

The results from boinc will be copied to corresponding folders under sixtrack_output directory, then be parsed and pushed to db.

2.2 Python Module Index

p

Bibliography

[1] Test.