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

Xiaohan Lu, Alessio Mereghetti

**Abstract**

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

Geneva, Switzerland
December 10, 2019

# Acknowledgement

I would like to thank my colleagues at CERN to help 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 [?] 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 library, hence, it can be imported into a python terminal or into custom-made python scripts. The library also comes with python wrapper scripts, such that specific commands can be issued directly from the login terminal.

You can download the pySixDesk package 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.

## 1.1 Getting started

pySixDesk is a library of utilities and it doesn't provide a user workspace. Hence, please keep separated the library from your own workspace.

**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>/)
```

**Simple usages**

This short guide will explain how to set up a quick toy study. By default the jobs will be submitted to HTCondor. If you want to use a different management system, you need to create a new cluster class with the interface (Cluster) defined in the submission.py module and specifiy it in the config.py script.

1. prepare the workspace. To do so, you have to create an instance of the parent class StudyFactory, which handles the workspace. If no argument is given, the default location ./sandbox is used:

   ```
   from pysixdesk import Study
   from pysixdesk import WorkSpace
   myWS = WorkSpace('./myWS')
   ```

2. prepare necessary folders (e.g. ./myWS/studies/test) and copy template files (including config.py) for a study. If not argument is given, the default study name is test (if no studies are present) or study_??? (with ??? being a zero-padded index of the study, calculated from the existing ones):

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

3. edit the config.py file to add scan parameters;

4. load definition of study in config.py and create/update database:

   ```
   myStudy = myWS.load_study('myStudy')
   myStudy.update_db() # only needed for a new study or when parameters are changed
   ```

5. parepare and submit MADX jobs and sixtrack one turn jobs or jobs for generating fort.2 for collimation, and collect results:

   ```
   myStudy.prepare_preprocess_input()
   myStudy.submit(0, 5) # 0 stands for preprocess job, 5 is trial number
   myStudy.collect_result(0) # collect results locally
   ```

6. prepare and submit actual sixtrack jobs, and collect results:

   ```
   myStudy.prepare_sixtrack_input()
   or
   myStudy.prepare_sixtrack_input(True) #True: submit jobs to Boinc
   myStudy.submit(1, 5) # 1 stands for sixtrack job, 5 is trial number
   myStudy.collect_result(1) # 1 stands for sixtrack job
   or
   myStudy.collect_result(1, True) # True: collect results from boinc spool
       directory
   ```

## 1.2 Parameter prepare

After initializing the study folder, config.py need to be edited to prepare the parameters for the study.

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

**General parameters**

There are some general parameters for the study as following:

```
# 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 the custom cluster, please make sure 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
```

Select the database:

```
self.db_info['db_type'] = 'sql'
# self.db_info['db_type'] = 'mysql'
```

sqlite3 is file-based local database, and mysql is server-based database.

Set the paths of executables:

```
self.paths['madx_exe']="/afs/cern.ch/user/m/mad/bin/madx"
self.paths['sixtrack_exe']="/afs/cern.ch/project/sixtrack/build/sixtrack"
```

The user should make sure the nodes of the selected cluster could access the executables, or the jobs will fail.

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
```

## 1.3 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
```

# Bibliography

[1] LBL diffential algebra package and LieLib routines courtesy of É. Forest.