**SigProcOpenPython - User’s Guide**

VDocument/version: *SigProcOpenPython - user's guide v1.0.docx*

**Contents**

Introduction 1

ChangeLog 1

How to install and run SigProcOpenPython – Main steps 1

Create python3 environment with needed packages 1

Install SigProcOpenPython 2

Prepare SigProcOpenPython json configuration file 3

Prepare the f() function 4

Run SigProcOpenPython 5

# Introduction

SigProcOpenPython (SigProcOP) is the add-on of Safibra OptiGuard product with the ability to add the user defined function prepared in python.

The user defined function has a general form Y = f(x1, x2, … xn), where xi represents an input value coming from a defined SigProc device/sensor with a given timestamp.

SigProcOpenPython works as follows:,

* Listens for incoming packets from different devices/sensors packets SigProc, parses them and stores the data in the internal “collection buffer”
* When the stored data in the collection buffer are identified to hold all xi input values (based on the f() function definition), the processing of the value Y is done and the Y value is sent to the defined device/sensor

# ChangeLog

v1.0

* Initial version

# How to install and run SigProcOpenPython – Main steps

The main steps to install and run SigProcOpenPython are as follows (for details for each step, see below):

* Create python3 environment with needed packages
* Install SigProcOpenPython
* Prepare SigProcOpenPython json configuration file
* Prepare the f() function
* Run SigProcOpenPython

## 

# python3 environment

SigProcOpenPython can be run at any server with python 3 available. The instructions in this chapter expect Centos 7 server and Anaconda python installation. For other environment, appropriate changes needed to be done.

A specific user expected for the installation and SigProcOpenPython usage, ‘ml’ expected here.

*Installation steps:*

# Download and install miniconda

cd ~ ; mkdir -p Downloads

cd ~/Downloads

wget <https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh>

sh Miniconda3-latest-Linux-x86\_64.sh

# read and accept licence (yes)

# install to default /home/ml/miniconda3 location

# confirm "installer" to initialize Miniconda3 (yes)

# relogin as ml

# update miniconda

conda update conda

# create python environment named "spop"

conda create -n spop pip python=3.8 pandas dill

# activate python envionment spop

conda activate spop

# Installation of SigProcOpenPython

# login as ml user

cd ~

# download SigProcOpenPython from appropriate git repository

git clone https://github.com/marsal64/sigprocop.git

**Installation structure:**

/home/ml/sigprocop/

├── logs

│   └── sigprocopenpython\_run.log # run log

├── models # directory for models preparation

│   ├── addtwo.spop\_pred # example model 1 – sums two values

│   └── negats.spop\_pred # example model 2 – negates value

├── create\_spop\_predict.py # function for creating f()

├── sigprocoprun.py # main run code for SigProcOpenPython

├── sigprocop

│   ├── functionality.py # helper functionality code for SigProcOpenPython

│   ├── \_\_init\_\_.py

│   └── \_\_pycache\_\_ # python cache

├── sigprocop.json # example configuration file

├── sigprocop2.json # example configuration file 2

├── README.md

├── run\_sigprocop.sh # example run script

└── run\_sigprocop2.sh # example run script 2

# SigProcOpenPython json configuration file

Configuration .json file is mandatory.

Default name of the .json file is “sigprocop.json”, otherwise can be parametrized using --config parameter

Sn example of the configuration file for SigProcOpenPython with comments:

|  |  |
| --- | --- |
| { |  |
| "name": "SigProcOpenPython", | Name used for logging |
| "description": "configuration for ObjectSense\_01", | Comment - description |
|  |  |
| "maxdblines": 10000, | Maximum number of internal “collection database” messaging buffer where data are collected together by timestamps to form a dataset for mm and offsetx\_abs calculation. If exceeded, some values will be discarded |
| "dbsectoerase": 60, | In seconds. If any measurement coming from input channel is older than dbsectoerase seconds, it is discarded as obsolete |
| "dbmaintainsec": 60, | The interval after which the database messaging buffer is processed to find obsolete measurement data |
| "maxdatabuf": 1000000, | Size of the internal input messages streaming buffer in bytes |
|  |  |
| "listen": { |  |
| "ip": "10.23.18.161", | IP address where SigProcOpenPython listens |
| "port": 5555 | Port where SigProcOpenPython listens |
| }, |  |
|  |  |
| "inputs": [ |  |
| {"name": "ch1", |  |
| "fromdevice": "ProcessGuard\_15", | Measurement 1 device id |
| "fromsensor": "os\_01" | Measurement 1 sensor id |
| }, |  |
| {"name": "ch2", |  |
| "fromdevice": "ProcessGuard\_15", | ... |
| "fromsensor": "os\_02" | ... |
| }, |  |
| {"name": "ch3", |  |
| "fromdevice": "ProcessGuard\_15", | ... |
| "fromsensor": "os\_03" | ... |
| }, |  |
| {"name": "ch4", |  |
| "fromdevice": "ProcessGuard\_15", | ... |
| "fromsensor": "os\_04" | ... |
| }, |  |
| {"name": "ch5", |  |
| "fromdevice": "ProcessGuard\_15", | ... |
| "fromsensor": "os\_05" | ... |
| }, |  |
| {"name": "ch6", |  |
| "fromdevice": "ProcessGuard\_15", | ... |
| "fromsensor": "os\_06" | ... |
| }, |  |
| {"name": "ch7", |  |
| "fromdevice": "ProcessGuard\_15", | ... |
| "fromsensor": "os\_07" | ... |
| }, |  |
| {"name": "ch8", |  |
| "fromdevice": "ProcessGuard\_15", | Measurement 8 device id |
| "fromsensor": "os\_08" | Measurement 8 sensor id |
| } |  |
| ], |  |
|  |  |
| "model": { |  |
| "modelfile": "models/model\_mm.ml", | File with the f() function |
| "modelname": "SigProcOpenPython model 1" | Model comment |
| }, |  |
|  |  |
| "output": |  |
| {"ip": "10.23.18.73", | IP where SigProcOpenPython is sending output 1 (mm) |
| "port": 5555, | Port where SigProcOpenPython is sending output 1 (mm) |
| "device": "python", | Device id for output 1 (mm) |
| "sensor": "result\_01" | Sensor id for output 1 (mm) |
| } |  |
|  |  |
| } |  |

# Preparation of the f() function

To create f() “prediction” function, the script create\_spop\_predict.py is used.

This script must be amended by the end user to create a specific f() function as requested.

Each function should have a distinct name. The final function is exported using python module dill to the subdirectory “models”. The dill file has default extension spop\_predict and the file name must be entered to the .json configuration file to be used.

An example how to create negating f() function (fragment of create\_spop\_predict.py):

##########################################################################

### create a simple function which returns the negative value

MODELFILENAME = 'negate.spop\_predict'

print(f"\*\*\* preparing {MODELFILENAME}")

def spop\_predict(X): # do not change the function name

return -X

dill.dump(spop\_predict, open(MODELFILENAME, 'wb'))

print(f'\*\*\* {MODELFILENAME} dill file saved in the folder {CNAME}\n')

##########################################################################

To create own function:

* Modify MODELFILENAME with the requested function filename
* Modify the spop\_predict function code
  + One parameter is expected. If more than one input values are configured in .json, the X contains always all the values which can be addressed X[0], X[1], …
  + The code of the function can be thus, for example (returns sum of two values):

def spop\_predict(X):

return X[0] + X[1]

* + The function code can be simple, or - for example – a complex scikit-learn model
    - If some specific python modules are used, add them during the installation of the spop environment (see above)

# How to run SigProcOpenPython

SigProcOpenPython can be run manually as follows (ml user expected):

cd ~/sigprocop

conda activate spop

python sigprocoprun.py

or in debug mode:

python ogmlrun.py --debug

Additionally, other parameters can be used for ogmlrun.py as follows:

usage: ogmlrun.py [-h] [--paramfile PARAMFILE] [--debug] [--rundir RUNDIR]

[--nolog] [--mode MODE] [--port PORT]

optional arguments:

-h, --help show this help message and exit

--paramfile PARAMFILE

.json file with parameters.

--debug If present, runs in debug mode.

--rundir RUNDIR If present, does cd to the given directory. If not

present, it runs in the directory when the sigprocrun.py

script is present.

--nolog If present, disables logging.

Helper .sh files may be used to run, e.g.:

bash run\_sigprocop.sh