developer.skatelescope.org Documentation

Release 0.1.0-beta

Marco Bartolini

Sep 22, 2022

Contents

PSS Test Vector Generation Instructions

The PSS Test Vector Generator created synthetic filterbank files containing simulated pulsar signals for the purpose of testing the pulsar and single pulse searching pipeline (PSS). The code can be used to generate single vectors or batches of vectors depending on user inputs. The code can be run in "production mode" - in which vectors are written to the official SKA public repository of test vectors, or in "non-production mode" where vectors are written to a location of the user's choice.

The code is essentially a wrapper around the existing file generation and single injection tools fast_fake and ft_inject_pulsar.

The code can be run natively (assuming fast_fake and ft_inject_pulsar are present in the user's path) or as a docker image. Instructions for generating vectors in each case are below.

There are three executables which depending on the mode (batch or single-vector) can be used to produced vectors.

- GenerateNoise.py - A wrapper for fast_fake. Produces a synthetic noise filterbank.
- GenerateGaussian.py - Produces a gaussian pulse as a 1-D ascii file. This pulse is injected into the noise file to simulated a pulsar filterbank.
- Generator.py - A tool for generating either single vectors, using an existing noise filterbank and profile file, or batches of vectors. In the latter case, GenerateNoise.py and GenerateGaussian.py are called by Generator.py.

Running the generator natively

To make a single vector

We first need to make a filterbank containing just noise, in which to inject a simulated pulsar signal.

‘‘‘bash python GenerateNoise.py –tobs 60 –nchans 16 -S 5050 -o noise_seed=5050.fil

1|2021-10-08T10:45:41|INFO|make_noise|GenerateNoise.py#164|Calling fast_fake with: fast_fake -T 60.0 -t 64.0 -m 56000.0 -F 1670.0 -f -0.078125 -b 8 -c 16 -s test -S 5050 -o noise_seed=5050.fil [fast_fake.c:101] FAST-FAKE - M.Keith 2014 [fast_fake.c:110] tobs = 60.000000s [fast_fake.c:111] tsamp = 64.000000us [fast_fake.c:112] mjdstart = 56000.000000 [fast_fake.c:113] freq chan 1 = 1670.000000MHz [fast_fake.c:114] freq offset = -0.078125MHz [fast_fake.c:115] output nbits = 8 [fast_fake.c:116] random seed = 5050 [fast_fake.c:117] output file = ‘noise_seed=5050.fil’ [fast_fake.c:128] write header [fast_fake.c:161] Generate 937500 samples, 0.014 GiB Complete: 99%. Sample: 928125 Real time 2.0s, Sim time 59.4s. Speed 7.08MiB/ss/s [fast_fake.c:247] Done! 1|2021-10-08T10:45:43|INFO|fexists|GenerateNoise.py#154|Noise file noise_seed=5050.fil produced 1|2021-10-08T10:45:43|INFO|make_noise|GenerateNoise.py#171|Noise generation completed ‘‘‘

This will produced a file called noise_seed=5050.fil containing 16 channels and corresponding to an observation duration (tobs) of 60 s. Run this with -h to see the full list of options. (If no arguments are provided a filterbank will be produced according to SKA_MID parameters.)

If you have header installed you can see the header parameters or the file produced.

‘‘‘bash header noise_seed=5050.fil

Data file : noise_seed=5050.fil Header size (bytes) : 241 Data size (bytes) : 15000000 Data type : filterbank (topocentric) Telescope : Parkes Datataking Machine : BPSR Source Name : test Frequency of channel 1 (MHz) : 1670.000000 Channel bandwidth (MHz) : -0.078125 Number of channels : 16 Number of beams : 1 Beam number : 1 Time stamp of first sample (MJD) : 56000.000000000000 Gregorian date (YYYY/MM/DD) : 2012/03/14 Sample time (us) : 64.00000 Number of samples : 937500 Observation length (seconds) : 60.0 Number of bits per sample : 8 Number of IFs : 1 ‘‘‘

Next we need to create a pulse to inject. We'll create a pulse normalised to the peak value (-p) using 128 bins with a duty cycle of 0.1 turns.

‘‘‘bash python GenerateGaussian.py -d 0.1 -n 128 -p

1|2021-10-08T10:50:53|INFO|generate_profile|GenerateGaussian.py#203|Inputs checked - can proceed 1|2021-10-08T10:50:53|INFO|write_out|GenerateGaussian.py#170|Writing file Gaussian_DC=0.1_BINS=128_FWHM_12.8.asc 1|2021-10-08T10:50:53|INFO|generate_profile|GenerateGaussian.py#240|======Results=
1|2021-10-08T10:50:53|INFO|generate_profile|GenerateGaussian.py#241|Demanded duty cycle: 0.1 1|2021-10-08T10:50:53|INFO|generate_profile|GenerateGaussian.py#242|Demanded FWHP: 12.8 bins. Actual FWHP: 12.799999755227066 bins 1|2021-10-08T10:50:53|INFO|generate_profile|GenerateGaussian.py#243|Demanded stdev: 5.435659521843323. Actual stdev: 5.435659521843311 1|2021-10-08T10:50:53|INFO|generate_profile|GenerateGaussian.py#244|Demanded mean: 64.0. Actual mean: 64.0 1|2021-10-08T10:50:53|INFO|generate_profile|GenerateGaussian.py#246|Demanded height: 1.0. Actual height: 1.0000000491263215 1|2021-10-08T10:50:53|INFO|generate_profile|GenerateGaussian.py#255|File generation complete ‘‘‘

This will produce a file in the present working directory called Gaussian_DC=0.1_BINS=128_FWHM_12.8.asc.

![Gaussian Pulse](example_pulse.png)

We can then call Generator.py in single-vector mode passing in our noise file, our pulse and the parameters of the pulsar we wish to simulate.

```
`bash python Generator.py -n noise_seed\=5050.fil -p Gaussian_DC\=0.
1_BINS\=128_FWHM_12.8.asc --dm 50 --freq 1 --sn 100 -l test_vectors `
```

Which will generate a filterbank containing a fake observation of a pulsar with a spin-frequency of 1 Hz, a dispersion measure (DM) of 50 pc/cc and a signal-to-noise ratio of 100, into the directory test_vectors. The console output will look something like this. . . .

```
`bash 1|2021-10-08T11:02:11|INFO|main|Generator.py#725|Reading parameters
from command line 1|2021-10-08T11:02:11|INFO|inject|Generator.py#602|Directory
test_vectors not found. Creating... 1|2021-10-08T11:02:11|INFO|inject|Generator.
py#604|Directory test_vectors created 1|2021-10-08T11:02:11|INFO|fexists|Generator.
py#444|File Gaussian_DC=0.1_BINS=128_FWHM_12.8.asc found 1|2021-10-08T11:02:11|INFO|inject
py#623|Bandwidth is:  1250000.0 1|2021-10-08T11:02:11|INFO|inject|Generator.
py#625|Obs duration is:  60.0 1|2021-10-08T11:02:11|INFO|inject|Generator.
py#626|Telescope gain:  1.0 K/Jy 1|2021-10-08T11:02:11|INFO|inject|Generator.
py#627|System temperature:  25.0 K 1|2021-10-08T11:02:11|INFO|inject|Generator.
py#628|Number of pols:  1 1|2021-10-08T11:02:11|INFO|get_dc_from_prof|Generator.
py#459|Duty cycle is 0.1 1|2021-10-08T11:02:11|INFO|inject|Generator.
py#641|Flux is 0.09622504486493763 Jy 1|2021-10-08T11:02:11|INFO|inject|Generator.
py#645|Vector name will be Default_2256a8a_1.0_0.1_50.0_0_Gaussian_100.
0_5050.fil 1|2021-10-08T11:02:11|INFO|form_command|Generator.py#498|Output
path is test_vectors/Default_2256a8a_1.0_0.1_50.0_0_Gaussian_100.0_5050.
fil 1|2021-10-08T11:02:11|INFO|form_command|Generator.py#529|Injector
command is:  ft_inject_pulsar noise_seed=5050.fil -p Gaussian_DC=0.
1_BINS=128_FWHM_12.8.asc -S 0.09622504486493763 --f0 1.0 -D 50.0 --accn
0 --pepoch 56000.0 --offset 0 --tsys 25.0 --gain 1.0 --npol 1 -x -o
test_vectors/Default_2256a8a_1.0_0.1_50.0_0_Gaussian_100.0_5050.fil
1|2021-10-08T11:02:11|INFO|inject|Generator.py#651|Injecting signal
into noise_seed=5050.fil 1|2021-10-08T11:02:14|INFO|inject|Generator.
py#653|Injection complete `
```

and if we look in test_vectors:

‘‘‘bash ls test_vectors

Default_2256a8a_1.0_0.1_50.0_0_Gaussian_100.0_5050.fil ‘‘‘

We see a filterbank with a naming scheme of:

[vector type]_[gitlab hash]_[frequency]_[duty cycle]_[DM]_[acceleration]_[shape]_[S/N]_[Noise seed].fil

To make a batch of vectors

Generator.py has a "batch mode". In this mode you can make a larger number of vectors (or just one!) with the parameters of the required noise, "telescope", profiles and pulsar signals all contained in a yaml file which can be passed into Generator.py using the -B flag. The examples/ directory contained some example yaml files. A simple example is here:

‘‘‘bash — noise:

tobs: 10 tsamp: 64 mjd: 56000.0 fch1: 1670.0 chbw: -0.078125 nbits: 8 nchans: 16 name: "noise"

telescope: tsys: 25 npol: 1 gain: 1

profiles: nbins: 128 widths:

- 0.05
- 0.1

vectors:

"0": vtype: "DDTR-MID" dms:

- 100
- 1

frequencies:

- 1000.0
- 1.0

accelerations:

- 0.0

snrs:

- 50.0
- 100.0

‘‘‘

Upon parsing the yaml file, Generator.py will call the noise and profile generators for you and then make a vector for every combination of profile widths, dms, frequencies, accelerations and S/Ns - so 16 vectors will result from the config file above.

`bash python Generator.py -B example.yaml -S 6060 -l test_vectors `

resulting in the following console output, showing each of the vectors being produced.

‘‘‘bash 1|2021-10-08T12:35:55|INFO|main|Generator.py#719|Running in Batch mode. Reading parameters from example.yaml 1|2021-10-08T12:35:55|WARNING|check_load|Generator.py#248|You are generating 16 vectors totalling 0.04 GB of disk space Are you sure you want to continue? (Yes/yes): Yes 1|2021-10-08T12:35:58|INFO|generate_batch|Generator.py#291|Using seed 6060 1|2021-10-08T12:35:58|INFO|generate_batch|Generator.py#299|Calling noise generator: noise_seed=6060.fil 1|2021-10-08T12:35:58|INFO|make_noise|GenerateNoise.py#164|Calling fast_fake with: fast_fake -T 10 -t 64 -m 56000.0 -F 1670.0 -f -0.078125 -b 8 -c 16 -s noise -S 6060 -o noise_seed=6060.fil [fast_fake.c:101] FASTFAKE - M.Keith 2014 [fast_fake.c:110] tobs = 10.000000s [fast_fake.c:111] tsamp = 64.000000us [fast_fake.c:112] mjdstart = 56000.000000 [fast_fake.c:113] freq chan 1 = 1670.000000MHz [fast_fake.c:114] freq offset = -0.078125MHz [fast_fake.c:115] output nbits = 8 [fast_fake.c:116] random seed = 6060 [fast_fake.c:117] output file = 'noise_seed=6060.fil' [fast_fake.c:128] write header [fast_fake.c:161] Generate 156250 samples, 0.002 GiB Complete: 100%. Sample: 156200 Real time 1.0s, Sim time 10.0s. Speed 2.38MiB/s [fast_fake.c:247] Done! 1|2021-10-08T12:35:59|INFO|fexists|GenerateNoise.py#154|Noise

file noise_seed=6060.fil produced 1|2021-10-08T12:35:59|INFO|make_noise|GenerateNoise.py#171|Noise generation completed 1|2021-10-08T12:35:59|INFO|generate_batch|Generator.py#325|Profile directory 'profiles' created 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#203|Inputs checked - can proceed 1|2021-10-08T12:35:59|INFO|write_out|GenerateGaussian.py#170|Writing file profiles/Gaussian_DC=0.05_BINS=128_FWHM_6.4.asc 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#240|=====
1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#241|Demanded duty cycle: 0.05 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#242|Demanded FWHP: 6.4 bins. Actual FWHP: 6.399999877613538 bins 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#243|Demanded stdev: 2.7178297609216613. Actual stdev: 2.7178297609216573 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#244|Demanded mean: 64.0. Actual mean: 64.0 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#246|Demanded height: 1.0. Actual height: 1.0000000491263221 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#255|File generation complete 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#203|Inputs checked - can proceed 1|2021-10-08T12:35:59|INFO|write_out|GenerateGaussian.py#170|Writing file profiles/Gaussian_DC=0.1_BINS=128_FWHM_12.8.asc 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#240|=====
1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#241|Demanded duty cycle: 0.1 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#242|Demanded FWHP: 12.8 bins. Actual FWHP: 12.799999755227066 bins 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#243|Demanded stdev: 5.435659521843323. Actual stdev: 5.435659521843311 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#244|Demanded mean: 64.0. Actual mean: 64.0 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#246|Demanded height: 1.0. Actual height: 1.0000000491263215 1|2021-10-08T12:35:59|INFO|generate_profile|GenerateGaussian.py#255|File generation complete 1|2021-10-08T12:35:59|INFO|inject|Generator.py#602|Directory test_vectors not found. Creating... 1|2021-10-08T12:35:59|INFO|inject|Generator.py#604|Directory test_vectors created 1|2021-10-08T12:35:59|INFO|fexists|Generator.py#444|File profiles/Gaussian_DC=0.05_BINS=128_FWHM_6.4.asc found 1|2021-10-08T12:35:59|INFO|inject|Generator.py#623|Bandwidth is: 1250000.0 1|2021-10-08T12:35:59|INFO|inject|Generator.py#625|Obs duration is: 10.0 1|2021-10-08T12:35:59|INFO|inject|Generator.py#626|Telescope gain: 1 K/Jy 1|2021-10-08T12:35:59|INFO|inject|Generator.py#627|System temperature: 25 K 1|2021-10-08T12:35:59|INFO|inject|Generator.py#628|Number of pols: 1 1|2021-10-08T12:35:59|INFO|get_dc_from_prof|Generator.py#459|Duty cycle is 0.05 1|2021-10-08T12:35:59|INFO|inject|Generator.py#641|Flux is 0.08111071056538127 Jy 1|2021-10-08T12:35:59|INFO|inject|Generator.py#645|Vector name will be DDTR-MID_b41c1f7_1000.0_0.05_100_0.0_Gaussian_50.0_6060.fil 1|2021-10-08T12:35:59|INFO|form_command|Generator.py#498|Output path is test_vectors/DDTR-MID_b41c1f7_1000.0_0.05_100_0.0_Gaussian_50.0_6060.fil 1|2021-10-08T12:35:59|INFO|form_command|Generator.py#529|Injector command is: ft_inject_pulsar noise_seed=6060.fil -p profiles/Gaussian_DC=0.05_BINS=128_FWHM_6.4.asc -S 0.08111071056538127 –f0 1000.0 -D 100 –accn 0.0 –pepoch 56000.0 –offset 0 –tsys 25 –gain 1 –npol 1 -x -o test_vectors/DDTR-MID_b41c1f7_1000.0_0.05_100_0.0_Gaussian_50.0_6060.fil 1|2021-10-08T12:35:59|INFO|inject|Generator.py#651|Injecting signal into noise_seed=6060.fil 1|2021-10-08T12:36:00|INFO|inject|Generator.py#653|Injection complete 1|2021-10-08T12:36:00|INFO|inject|Generator.py#600|Output directory test_vectors found 1|2021-10-08T12:36:00|INFO|fexists|Generator.py#444|File profiles/Gaussian_DC=0.05_BINS=128_FWHM_6.4.asc found 1|2021-10-08T12:36:00|INFO|inject|Generator.py#623|Bandwidth is: 1250000.0 1|2021-10-08T12:36:00|INFO|inject|Generator.py#625|Obs duration is: 10.0 1|2021-10-08T12:36:00|INFO|inject|Generator.py#626|Telescope gain: 1 K/Jy 1|2021-10-08T12:36:00|INFO|inject|Generator.py#627|System temperature: 25 K 1|2021-10-08T12:36:00|INFO|inject|Generator.py#628|Number of pols: 1 1|2021-10-08T12:36:00|INFO|get_dc_from_prof|Generator.py#459|Duty cycle is 0.05 1|2021-10-08T12:36:00|INFO|inject|Generator.py#641|Flux is 0.16222142113076254 Jy 1|2021-10-08T12:36:00|INFO|inject|Generator.py#645|Vector name will be DDTR-MID_b41c1f7_1000.0_0.05_100_0.0_Gaussian_100.0_6060.fil 1|2021-10-08T12:36:00|INFO|form_command|Generator.py#498|Output path is test_vectors/DDTR-MID_b41c1f7_1000.0_0.05_100_0.0_Gaussian_100.0_6060.fil 1|2021-10-08T12:36:00|INFO|form_command|Generator.py#529|Injector command is: ft_inject_pulsar

noise_seed=6060.fil -p profiles/Gaussian_DC=0.05_BINS=128_FWHM_6.4.asc -S 0.16222142113076254 –f0 1000.0 -D 100 –accn 0.0 –pepoch 56000.0 –offset 0 –tsys 25 –gain 1 –npol 1 -x -o test_vectors/DDTR-MID_b41c1f7_1000.0_0.05_100_0.0_Gaussian_100.0_6060.fil 1|2021-10-08T12:36:00|INFO|inject|Generator.py#651|Injecting signal into noise_seed=6060.fil 1|2021-10-08T12:36:01|INFO|inject|Generator.py#653|Injection complete 1|2021-10-08T12:36:01|INFO|generate_batch|Generator.py#386|Injection complete

1|2021-10-08T12:36:01|INFO|inject|Generator.py#600|Output directory test_vectors found 1|2021-10-08T12:36:01|INFO|fexists|Generator.py#444|File profiles/Gaussian_DC=0.05_BINS=128_FWHM_6.4.asc found 1|2021-10-08T12:36:01|INFO|inject|Generator.py#623|Bandwidth is: 1250000.0 1|2021-10-08T12:36:01|INFO|inject|Generator.py#625|Obs duration is: 10.0 1|2021-10-08T12:36:01|INFO|inject|Generator.py#626|Telescope gain: 1 K/Jy 1|2021-10-08T12:36:01|INFO|inject|Generator.py#627|System temperature: 25 K 1|2021-10-08T12:36:01|INFO|inject|Generator.py#628|Number of pols: 1 1|2021-10-08T12:36:01|INFO|get_dc_from_prof|Generator.py#459|Duty cycle is 0.05 1|2021-10-08T12:36:01|INFO|inject|Generator.py#641|Flux is 0.08111071056538127 Jy 1|2021-10-08T12:36:01|INFO|inject|Generator.py#645|Vector name will be DDTR-MID_b41c1f7_1.0_0.05_100_0.0_Gaussian_50.0_6060.fil 1|2021-10-08T12:36:01|INFO|form_command|Generator.py#498|Output path is test_vectors/DDTR-MID_b41c1f7_1.0_0.05_100_0.0_Gaussian_50.0_6060.fil 1|2021-10-08T12:36:01|INFO|form_command|Generator.py#529|Injector command is: ft_inject_pulsar noise_seed=6060.fil -p profiles/Gaussian_DC=0.05_BINS=128_FWHM_6.4.asc -S 0.08111071056538127 –f0 1.0 -D 100 –accn 0.0 –pepoch 56000.0 –offset 0 –tsys 25 –gain 1 –npol 1 -x -o test_vectors/DDTR-MID_b41c1f7_1.0_0.05_100_0.0_Gaussian_50.0_6060.fil 1|2021-10-08T12:36:01|INFO|inject|Generator.py#651|Injecting signal into noise_seed=6060.fil 1|2021-10-08T12:36:02|INFO|inject|Generator.py#653|Injection complete . . . . . 1|2021-10-08T12:36:17|INFO|cleanup|Generator.py#255|Deleting noise_seed=6060.fil 1|2021-10-08T12:36:17|INFO|cleanup|Generator.py#257|Deleting profiles ‘‘‘

Looking again in our test_vectors directory we see our 16 vectors.

```
`bash ls test_vectors/ DDTR-MID_b41c1f7_1000.0_0.05_100_0.0_Gaussian_100.
0_6060.fil DDTR-MID_b41c1f7_1.0_0.05_100_0.0_Gaussian_100.0_6060.
fil DDTR-MID_b41c1f7_1000.0_0.05_100_0.0_Gaussian_50.0_6060.
fil DDTR-MID_b41c1f7_1.0_0.05_100_0.0_Gaussian_50.0_6060.fil
DDTR-MID_b41c1f7_1000.0_0.05_1_0.0_Gaussian_100.0_6060.fil DDTR-MID_b41c1f7_1.
0_0.05_1_0.0_Gaussian_100.0_6060.fil DDTR-MID_b41c1f7_1000.0_0.
05_1_0.0_Gaussian_50.0_6060.fil DDTR-MID_b41c1f7_1.0_0.05_1_0.
0_Gaussian_50.0_6060.fil DDTR-MID_b41c1f7_1000.0_0.1_100_0.0_Gaussian_100.
0_6060.fil DDTR-MID_b41c1f7_1.0_0.1_100_0.0_Gaussian_100.0_6060.fil
DDTR-MID_b41c1f7_1000.0_0.1_100_0.0_Gaussian_50.0_6060.fil DDTR-MID_b41c1f7_1.
0_0.1_100_0.0_Gaussian_50.0_6060.fil DDTR-MID_b41c1f7_1000.0_0.1_1_0.
0_Gaussian_100.0_6060.fil DDTR-MID_b41c1f7_1.0_0.1_1_0.0_Gaussian_100.
0_6060.fil DDTR-MID_b41c1f7_1000.0_0.1_1_0.0_Gaussian_50.0_6060.fil
DDTR-MID_b41c1f7_1.0_0.1_1_0.0_Gaussian_50.0_6060.fil `
```

Note that the code will stop and prompt the user with information about the number and size of the test vector dataset and will continuue with the generation if the user responds “yes”. This is a safeguard as test vectors can be very large (30 GB for a standard SKA-MID vector). There is a –nocheck option that can be passed that will override this (for use in automated generation pipelines).

Note also that in the yaml file there is a “vtype” key and the value appears as the first fields in the test vector name. This is the “vector type” which is a marker describing the intended test case for the vector. This can be any string, however in production mode this can only be one of a set of pre-defined strings.

Production mode.

In PSS we are populating an official repository of pulsar and single pulse test vectors which will available for testing

various components of the PSS pipeline. These vectors are stored on our official test vector server "dokimi". When adding new vectors to this repository we use production mode by passing the -P option (whether in batch of single-vector mode). When this mode is enabled, vectors are stored in a pre-defined location on dokimi's filesystem. We also hold directories containing the "latest" versions of each vector (as symlinks), such that if we make a vector of a particular vtype with the same parameters as a vector already held on the server, the old version will be preserved, but the latest version symlink will be updated to point to the new version. The versions are distibuishable from the gitlab hash in the filename allowing old vectors to be easily recovered.

This mode is intended for the use of the PSS team who have access to dokimi. If a production repo is required elsewhere, the path to the repository will need to be updated to reflect the filesystem on your server. The variable TOP_LEVEL_REPO in Generator.py can be edited to point to your production location.

In this mode, permitted vtypes are:

- DDTR-MID
- FDAS-ACC-MID
- FDAS-FOP-MID
- FDAS-PER-MID
- FDAS-SLOW-MID
- FLDO-MID

Note that if –location is passed in addition to -P, -P will be ignored and vectors will be written to a non-production location instead.

The PSS Test Vector Generator docker image

All relevant dependences and scripts needed to run the Test Vector Generator is included in a docker image, set up through the Docker File. To make the image run

```
`bash docker build -t artefact.skao.int/ska-pss-test-vector-generator:latest .
`
```

Then deploy the image with

```
`bash docker run -it artefact.skao.int/ska-pss-test-vector-generator:latest `
```

. . . and then follow the steps describing the modes above.