National Centre for Radio Astrophysics

Internal Technical Report

GMRT/TGC/12Sep2023

GMRT New Command-File Generator: The Python Backend and Validation Tests

Deepak Bhong, Yogesh Maan, Santaji Katore, Nilesh Raskar, Bhaswati Bhattacharyya, Visweshwar R. Marthi, Jitendra Kodilkar, Yashwant Gupta, and C. H. Ishwara Chandra

Email: deepak@gmrt.ncra.tifr.res.in, ymaan@ncra.tifr.res.in

Objective: To provide the technical details, usage information as well as details of the validation tests regarding the python backend of the new command-file generator.

Revision	Date	Modification/ Change
Ver. 1	12 Sept. 2023	Final Version

GMRT New Command-File Generator: The Python Backend and Validation Tests

Deepak Bhong, Yogesh Maan, Santaji Katore, Nilesh Raskar, Bhaswati Bhattacharyya, Visweshwar R. Marthi, Jitendra Kodilkar, Yashwant Gupta, and C. H. Ishwara Chandra

1 Introduction

The observatory provides its users with a graphical user interface (GUI) to create a command-file for a given observing session based on suitable inputs. However, before 2022, this GUI enabled command-file creation only for the interferometric (i.e., the continuum and line modes) observations. Moreover, the command files could be created only for the legacy GMRT backends, with only a brief extension to the GMRT wideband backends (GWB). Furthermore, shifting to the new Tango-based GMRT Control and monitoring system (TGC) also required the command-files to be created in Python for proper interface with various parts of the new system (as opposed to the previous 'online' mode). So, there was a crucial need to have a common command-file generator that enables planning the observations and creating the corresponding setup and command files for any of the available GMRT's observing modes, including various modes for the beamformer observations. To cater to the users' needs, a new GUI and a corresponding Python-based backend were developed and successfully tested. While more details about the GUI will be given elsewhere, this report mainly focuses on the details of the Python-based backend and its end-to-end validation tests.

2 The Python Backend to the Command-file Generator

The new GUI takes input from the user on all aspects of an observing session, including the observing mode, observation settings for the selected bands, source list, and the observing plan, and then outputs a human-readable but fixed-format file that contains all the user-specified information. The Python backend reads this fixed-format file and outputs four different files that are actually used to conduct an observing session: the correlator configuration file containing information regarding the correlator settings, a setup file containing information regarding the radio frequency (RF) settings, and the GMRT analog backend (GAB) settings, the source list, and the command-file.

2.1 Usage

The Python backend is currently named as `tgc_cmdgen.py`, and the information on its usage as well as several options can be obtained by executing it with the −`h` flag.

```
usage: new_cmd.py [-h] [-f FILE_NAME] [-b BACK_CORR] [-i] [-s INPUT_STRING] [-
F CMD_FILE_FORMAT] [-g GSB_BM_CONF]

options:
  -h, --help          show this help message and exit
  -f FILE_NAME        Input Configuraion file
  -b BACK_CORR        Backend correlator(GSB/GWB/BOTH)
  -i                  Take inputs from STDIN
  -s INPUT_STRING     Input the configuration-file's content in the form of string
  -g GSB_BM_CONF      GSB beam config
```

The following is a typical example of the backend on a setup file “gtac_Test1_29Aug2023_0900.txt” produced by the new GUI.

```
tgc_cmdgen.py -f gtac_Test1_29Aug2023_0900.txt -b GWB
```

2.2 Various Python Functions and Classes

The backend puts together various Python classes and functions to not only correctly parse the input files and generate the output files in the correct format, but also to provide the much-needed ease of adding a new feature. In fact, since the new command-file generator's release, many new features as well as provisions for new observing modes have been added to the GUI as well as the backend. A brief introduction to various Python classes and functions associated with the backend is given below.

1. `Source:` This class deals with a given source coordinates — it properly formats the coordinates and uses the method `process` to compute the rise and set times of the source.

2. `get_cdp_int:` This function calculates the post-integration time in the coherent dedispersion pipeline (CDP) mode, based on the given input number of channels, output number of channels, GWB bandwidth, and the required integration time.

3. `check_ra:` This function checks the validity of the given right ascension (RA) value.

4. `check_dec:` Similar to the above, checks the validity of the given declination (DEC) value.

5. `write_source_list:` This function reads the information provided by the user on various sources, checks for their correctness, and prepares the source-list file.

6. `write_cmd_file:` This is the main function that prepares and writes the actual command file based on the provided inputs. The function handles logic related to beam configurations, real-time RFI filtering, and correlator settings as well as different kinds of observing scans (e.g., `power_eq, flux_cal, phasing, phase_cal, target`, etc.).

7. `write_observe_source:` This function produces a properly formatted observing command string depending on the user-specified information, like format (new or the old ‘online’), sub-array, target-name, real-time RFI filtering, beam-mode observations, etc., and includes the commands to start the scan, track the source, start the data acquisition for each sub-array, phasing related commands for each sub-array, start the beam data recording in the specified mode, stop the data acquisition after the specified duration, etc.

8. `get_psr_rec_cmd:` This function parses the specified setup and recording information and prepares the pulsar/beam recording command strings for each beam within each of the sub-arrays.

9. `write_setup_file:` Generates a setup file for a project, configuring various settings such as RF, GAB, GWB beams, and correlator settings based on the provided input data.

10. `write_corr_csv:` This function populates a setup file with the correlator settings, either based on the project information input via the GUI or by the default values.

11. `parse_input_file:` This function parses the information input via the GUI (from the file produced by the GUI), using various functions.

12. `read_prj_info:` A function to read the content of the input file and extract project information from it.

13. `read_prj_setup:` This function reads the content from an input iterator, looks for lines containing project setup information stores the key-value pairs in a dictionary, and calls another function to read the setup for each sub-array.

14. `read_subar_setup:` This function reads the content from the input iterator and extracts setup information for a specified sub-array.

15. `read_beam_setup:` Function to read the content from the input iterator and extract setup information for a specified beam within a specified sub-array.

16. `read_src_list:` This function reads the input file and extracts a list of source entries.

17. `read_cmd_file:` This function reads the content from the input iterator and extracts command-file related information.

18. `read_special_req:` A function that reads the content from the input file and extracts the information from the "special requirements" section.

19. `check_pulsar_chain:` For a given beam setup, this function checks which branch of the pulsar backend needs to be used during the observation.

20. `gen_prj_setup:` This function generates a project setup by populating various parameters in the "prj_short_setup" dictionary based on input values from several other dictionaries.

21. `beam_int_to_fft:` Function to calculate the number of FFTs to be integrated for the input setup.

22. `get_gwb_io:` Function to calculate the GWB IO (input/output) budget for the given setup.

23. `get_gwb_tpa:` Function to calculate parameters related to the GWB TPA.

24. `write_prj_info:` This function writes project setup information to a specified file (or to console).

25. `cal_gwb_beam_datarate:` This function calculates the data rate for each GWB beam based on various input parameters, as well as the required disk space.

26. `cal_gwb_bm_datasize:` To determine the on-source time based on the input commands and calculate the data size for each beam using the data rate and on-source time.

27. `get_gwb_beam_datarate:` Function to calculate the beam data rate in MB per second based on the given parameters

28. `get_gwb_vis_datarate:` Function to calculate the output data rate for interferometry data based on the given parameters.

29. `get_band_code:` A function that assigns the internally used "band-code" for a given RF band.

30. `get_lta_record:` Using the relevant parameters from the input sub-array setup, this function generates a command string for recording data in the long-term archive (LTA) format.

2.3 Various python functionalities added to TGC (MNCScriptManager)

To ease the new command file generator's integartion with the TGC as well as to reduce the overall complexities, a number of functionalities were added to TGC. A brief description of some of these functionalities is provided below.

1. `sysCmd:` Passes system commands from TGC command file and maintain its log.

2. `init_proj:` To provide a user friendly interface to create project and associate it with sub array.

3. `start_corr:` To start correlator DAS chain in background from TGC.

4. `set_source_corr:` Combines addpsource and set_source commands, as well as avoids repeated execution of addpsource command if target source is observed repeatedly during observation session.

5. `mon_script:` To monitor state of currently running command file.

6. `stop_mon_script:` This function stops monitoring of command file.

7. `set_gwb_gac:` Select or deselect antennas in beamformer.

8. `rfi_fil_sleep:` To issue RFI counter reset and hold commands in specified intervals in background.

9. `get_ant_list:` To get the list of antennas as per specification e.g., csq, yant, want, cant and sant.

10. `delete_all_array:` Deletes all created sub-arrays.

11. `check_ant_agn:` Checks whether antenna is available with current Operator Stations.

12. `get_disk_space:` Get available disk space from remote host of specific directory or partition.

13. `remote_execute:` Execute command on remote host and get output of the command.

14. `get_gwb_lta_dir:` Get the directory which has max space available from GWB host for LTA file recording.

15. `get_gsb_lta_dir:` Get the directory which has max space available from GSB host for LTA file recording.

16. `get_gwb_beam_dir:` Get the directory with required space from BEAM recording hosts.

17. `start_acq:` Start GWB or GSB ACQ in background.

18. `check_acq:` Check whether ACQ is ready for init correlator command.

19. `start_collect:` Start collect in background on respective hosts.

20. `check_gab:` Check state of GAB system of each antenna, if it is found in a hanging state then issue reset command.

21. `observe_source:` Top level TGC command to change tracking target, change correlator source, start data recording and stop data recording after specified time.

3 Validation Tests of the Command-file Generator

The Python backend was tested and validated in multiple ways. First, GUI was used to input the setup and other information for a number of observing modes and setups. Each of the fixed-format setup file output from the GUI was then processed by the backend, and the output files were manually inspected to notice any anomaly. Furthermore, the output command-files were also matched with the ones produced by the older 'online' system for the same configurations (and after appropriately converting them to Python), and consistency was ensured. Finally, a number of end-to-end tests were performed. Here, by end-to-end, we mean that the setup files were prepared using the GUI for a given configuration and observing mode, the python backend was used to process these and produce the command files, which were then used with the system to actually conduct test observations, and the data were analyzed and inspected. These tests helped in resolving several bugs and adding features that would have otherwise been realized only during the actual observations. A selected list of these tests is given in Table 1 below.

4 Summary

We have presented here details of the Python backend that is used on the setup files produced from the user-facing GUI to produce the necessary command-files to conduct observations. The GUI and the Python backend together provide the new command-file generator system for GMRT, which has been released to the community for use since observing cycle 41. We have also provided a summary of the tests that were conducted to validate the command-file generator. While the command-file generator is fairly complete, new features are added as and when a need arise, e.g., when a new beam or functionality is added and released for use at GMRT.

Table 1: Validation tests of the Command-file Generator

Sr. No.	Test-description	Objective	Type-of-test	Date Completed
1	Single PA beam, 4K channels, 0.327 ms, 16 bits (target B2111+46)	The simplest mode — single subarray, single PA beam 16 bits, and a reasonable channels/sampling-time configuration.	End-to-end test.	25 Sept. 2021
2	PA and IA beams, 4K channels, 0.327 ms, 16 bits (target B2111+46)	Test for a single subarray, two beams, and also a different channels/sampling-time configuration.	End-to-end test.	25 Sept. 2021
3	CDPA (512 channels, 20 us), PA and IA beams, 1K channels, 40 us, 16 bits (targets B1937+21 and B1933+16)	Test for a single subarray, but three beams, and multiple sources.	End-to-end test.	28 Sept. 2021
4	CDPA, PA, and IA beams, the same configuration as above, 8 bits recording.	Same as above but with 8-bit recording mode.	End-to-end test.	23 Oct 2021
5	CDPA, PA, IA, and other PA beams, 16-bit recording.	Test for four beams simultaneously.	End-to-end test.	23 Oct 2021
6	PA (beam-bits 16 and recording bits 8) and CDPA (16 bits) beams, suitable targets, and a phase-cal with the beam-recording enabled.	Test for single subarray wherein only interferometric data are recorded for the phase-cal, and test the 16-bits correlator, 8-bit recording mode.	End-to-end test.	26 Nov. 2021
7	Two sub-arrays with CDPA, PA and IA beams, two targets one after the other.	Test for two subarrays (and multiple beams as well).	End-to-end test.	23 Oct 2021
8	PA (beam-bits 8 and recording bits 8) and CDPA (16-bits) beams.	Test the 8-bit correlator and 8-bit recording mode.	End-to-end test.	01 Dec. 2021
9	PASV recording	Test the PASV-recording mode withthe output of the command-file generator UI	Recording-only test	23 Nov. 2023