
Memorandum 007

# The NANOGrav TOA Generation Process

Ross Jennings

2021 September 29

http://nanograv.org/

# The NANOGrav TOA generation process

Ross Jennings

September 29, 2021

The goal of this document is to describe the process of data reduction and TOA generation used by NANOGrav, including the generation of intermediate data files, in sufficient detail to allow it to be reproduced exactly.

## 1 Getting the software and data

The data reduction and TOA generation process makes use of several pieces of software. The full list of requirements is:

- PSRCHIVE (http://psrchive.sourceforge.net/). This is the main tool used to view and manipulate data in PSRFITS format. It consists primarily of command-line utilities, written in C, but also has a Python interface and a scripting language, `psrsh`. Traditionally, PSRCHIVE has been installed by building it from source (see http://psrchive.sourceforge.net/installation.shtml for full instructions). However, successfully configuring the build tools can be challenging, especially if its dependencies are installed in non-standard locations. A much easier way to install it is using `conda`. Once a conda environment has been set up, PSRCHIVE can be installed simply by running

  ```
  conda install -c conda-forge psrchive
  ```

  This should automatically handle downloading and installing dependencies.

- `psrtools` (https://github.com/demorest/psrtools). This is a set of two command-line utilities (`autotoa` and `normalize_rms`) that depend on PSRCHIVE. Like PSRCHIVE, they are written in C, and can either be built from source or installed with conda:

  ```
  conda install -c demorest psrtools
  ```

- `nanopipe` (https://github.com/demorest/nanopipe). This is a collection of Python and `psrsh` scripts used to coordinate the various steps of the NANOGrav calibration and TOA generation pipeline. It can be installed by cloning the repository from GitHub:

```
git clone
    https://github.com/demorest/nanopipe.git
cd nanopipe
pip install .
```

- `toagen` (https://gitlab.nanograv.org/nano-time/toagen). This is a special-purpose repository which holds the full directory structure used for the calibration and TOA generation process, including configuration files which are not part of `nanopipe`. Currently access is limited to NANOGrav members. It is hosted on NANOGrav's internal GitLab server. If you are a NANOGrav member but have not used GitLab previously, you will have to log in to `gitlab.nanograv.org` with your NANOGrav username and password and add an SSH key (user icon in upper right > Settings > SSH keys) before you can access the repository. You can then clone it by running:

  ```
  git clone
      git@gitlab.nanograv.org:nano-time/toagen.git
  ```

  `toagen` is not an installable program or library, but rather contains a set of configuration files and Makefiles that are used in the TOA generation process, along with the current template profiles and .tim files for all NANOGrav pulsars. It does not contain the actual profile .fits files, as these are much to large to be version-controlled (see below).

- Profile data. The working copy of NANOGrav's profile data is stored on servers at WVU and is accessible from Bowser (`bowser.phys.wvu.edu`) or the notebook server (`notebook.nanograv.org`). VEGAS, GUPPI, and PUPPI data are stored under

  ```
  /nanograv/timing/data/
  ```

  The data files are grouped into directories by pulsar name (without the "B" or "J" prefix), backend, and year, and are always in a directory called `rawdata` at the lowest level. For example, the first part of an Arecibo obervation of PSR J1713+0747 taken on March 29, 2013 can be found at

  ```
  /nanograv/timing/data/1713+0747/puppi/2013/
    rawdata/puppi_56380_1713+0747_0542_0001.fits
  ```

  Calibration scans are found in the same directories, and can easily be identified by the presence of `cal` in the filenames. Flux calibrator observations are found in directories corresponding to the name of the flux calibrator.[1]

[1] A quasar, usually B1442+101 = J1445+099, but at Arecibo sometimes J1413+151, and at GBT 3C190 is used for one 2011 observation. There are multiple directories with slight variations on the name, and some with suffixes, but the most recent calibrator observations all seem to go into `1442+101`.

Some data from other telescopes used by NANOGrav are also available on the WVU servers, although not in the same place as the GBT and Arecibo data. YUPPI (VLA) data are stored under

```
/hyrule/data/users/pdemores/VLA/
```

and are grouped by semester and proposal number. Some CHIME data, grouped by pulsar, are also available, under

```
/hyrule/data/CHIME/NANOGrav/
```

As of September 29, 2021, only observations of PSR J1713+0747 from January to September of 2021 are available, but additional data should be copied over soon.

## 2 Setup and directory structure

The basic processing scripts are contained in `nanopipe`. As established in the instructions for `nanopipe` (https://github.com/demorest/nanopipe/blob/master/doc/basic_instructions.txt), the work takes place in a base directory which has subdirectories for each pulsar being processed. Within `toagen`, there is a separate base directory for each backend (ASP, GASP, PUPPI, GUPPI, YUPPI).

Flux calibrator observations are stored in a separate directory, called `fluxcal`, inside the base directory for each backend. Before processing the rest of the observations, one needs to create .fcal files from the flux calibrator observations. This is done using the PSRCHIVE tool `fluxcal`:

```
fluxcal -f -e fcal
```

The results can be checked using the PSRCHIVE calibrator viewer, `pacv`.

The base directory for each backend also contains several `psrsh` scripts (copied from `nanopipe`'s `config` directory): the basic list is

```
zap_minmax
zap_and_tscrunch
update_be_delay.psrsh
process_fluxcal
img_$be
```

Here `$be` is the backend name.

There should also be a configuration file called `make_psr_make.config.py` in the base directory for each backend, containing, at minimum, these lines:

```
ver_id = "$project_id"
tscrunch_arg = "-J ../zap_and_tscrunch"
```

Here `$project_id` is a string identifying the particular version of the processing being carried out.

This basic setup is customized in a backend-dependent way in the real NANOGrav TOA generation process, in ways that are captured in the `toagen` repository. Once the scripts and configuration files are in place, a makefile can be generated for each pulsar by running (from within the corresponding subdirectory)

```
make_psr_make $basedir > Make.psr
```

For `$basedir` one should substitute the full path of the base directory. The makefile can be used to produce TOAs by running

```
make -f Make.psr toas
```

(again from within the pulsar subdirectory). This command can be altered to run several processes in parallel by adding the option `-j $n_procs`, where `$n_procs` is the number of processes, or by replacing `toas` with another makefile target from this list:

```
rf
calibration
zap
scrunch
templates
toas
```

to only run part of the analysis.

# 3 Per-pulsar makefiles

The main process is laid out in the per-pulsar makefiles (`Make.psr`), which are generated by the `make_psr_make` script in `nanopipe`. The processing carried out by `Make.psr` breaks down into the following seven steps, most of which correspond directly to makefile targets:

1. Target `rf`: Combine the raw .fits files into .rf files. The raw data files must be copied in from where they are stored on WVU's servers. Several corrections are applied at this stage, namely:

   (a) Epoch error due to polyco `REF_MJD` precision in PSRFITS is corrected. This accounts for the effects of a bug in PSRCHIVE, discovered during the 9-year analysis, which meant that polyco reference MJDs were not stored using enough numerical precision.

   (b) Backend delays (stored in the `BE_DELAY` PSRFITS header item) are set appropriately. These measure extra signal latency within each backend, usually just digital filter latency, and are determined from the backend design (not measured experimentally). For GUPPI and PUPPI, the delay is a function of the number of channels and channel bandwidth in the observation.

(c) ADC ghost image corrections (Alam et al. 2021, §2.3.1; cf. Kurosawa et al. 2001) are applied.

(d) Par files are installed into the FITS header. This also means realigning the profiles according to the par file being installed. Usually the "predictive" par file from the previous data set is used here, if it is available.

(e) The DM value in the FITS header is set appropriately. This DM value is taken from the par file used above, and is used for de-dispersion prior to frequency averaging.

(f) Specified bands are zapped and time-frequency zapping (Offringa et al., 2010) is applied.

(g) The source and receiver names are fixed to make them uniform across the data set.

If this process fails, the file is noted in the list of cal failures. Finally, the database of calibrator archives is built up.

Commands:

```
psradd -T -j 'fix refmjd' -J
    ../update_be_delay.psrsh -J ../img_$be -j
    'install par $psr.basic.par' -j "e dm=$dm"
    -J ../zap_and_tscrunch_with_list_$tel -j "e
    rcvr:name=`fix_receiver_name $obsj.fits`"
    -j "e name=`get_proper_name $obsj.fits`" -o
    $obs.rf $obs1.fits $obs2.fits
psradd -T -j 'fix refmjd' -J
    ../update_be_delay.psrsh -J ../img_$be -J
    ../zap_and_tscrunch_with_list_$tel -j "e
    rcvr:name=`fix_receiver_name $calj.fits`"
    -j "e name=`get_proper_name $calj.fits`" -o
    $cal.cf $cal1.fits $cal2.fits
```

These commands (and the others quoted below) are taken from the `Make.psr` file, but reformatted to make them independently runnable and as generic as possible. The `psrsh` scripts

```
update_be_delay.psrsh
img_puppi
img_guppi
zap_and_tscrunch_with_list_gb
zap_and_tscrunch_with_list_ao
```

can be found in `nanopipe`'s `config` directory (however, the ones found in `nanopipe` use `zap median`, as was done for the 12.5-year and earlier data sets, rather than `zap tfzap`, used in `toagen` and the 15-year data set). The Python scripts

```
fix_receiver_name
get_proper_name
```

can be found in `nanopipe`'s `scripts` directory.

2. Target `calibration`: Calibrate the data (.rf files) to produce .calib files. This performs a very basic flux/polarization calibration (cal gain and phase correction) using `pac`. Matching with the calibrator observations is done on a per-channel basis, and the pulse phase of the cal transition is set to 0.5 turns. Calibrator Stokes parameters are derived from the fluxcal data. If this process fails, the file is noted in the list of cal failures. Then the data goes through another round of RFI zapping, removing a specified fraction of the band edge, as well as more specified frequency bands, but these seem to mostly be duplicates. Any zapping options in the variable `$XPAZ` are applied. Finally, if the file `$obs.calib` does not exist, the file `$obs.calibP` is removed.

   Commands:

   ```
   pac -w
   pac -a -j "config
       SquareWave::transition_phase=0.5" -x -e
       x.calib -d database.txt $obs.rf
   paz -m -E2.0 $zapchannels $(XPAZ) $obs.x.calib
   ```

   `$zapchannels` is a space-separated list of `-F` options specifying channels to zap on a per-receiver basis, configured in `make_psr_make`.

3. Target `zap`: Apply "min-max" zapping to produce .zap files.

   Command:

   ```
   psrsh -e zap ../zap_minmax $obs.x.calib
   ```

   The `psrsh` script `zap_minmax` can be found in `nanopipe`'s `config` directory. (But, just as for `zap_and_tscrunch_with_list_$tel` above, this is an older version of `zap_minmax` that uses `zap median` rather than `zap tfzap`. The `zap tfzap` version can be found in `toagen`.)

4. Target `scrunch` (1/2): Scale the weights so that the offpulse RMS of the weighted data is 1, producing .norm files.

   Command:

   ```
   normalize_rms -w $obs.x.zap
   ```

5. Target `scrunch` (2/2): Frequency average to 64 channels, and time average to a number of subintegrations set by the observation length (the length divided by 1800 s, rounded down, plus one), to produce .ff files.

   Command:

```
pam -e ff -f$factor --setnsub `psredit -Q -c
    length $obs.x.norm | awk '{print
    int($2/1800.0) + 1}'` $obs.x.norm
```

`$factor` varies depending on the backend. It is 8 for receivers with 512 channels (GUPPI and PUPPI L-band, PUPPI S-band) and 2 for GUPPI 820 MHz, which has 128 channels.

6. Target `templates`: Iteratively determine the template and TOAs using `autotoa` (an implementation of the method outlined in Chapter 2 of Demorest 2007), with a list of all .ff files for this pulsar-frontend-backend combination as input. Use a Gaussian of width 0.1 as the initial guess for the template, and perform at most 3 iterations. Then rotate the final profile by half a turn, and apply UD8 wavelet smoothing to it.

   Commands:

   ```
   for f in {$obs.ff}; do echo $f >>  $rcvr.fflist
   autotoa -g0.1 -i3 -S $rcvr.sum $rcvr.fflist
   pam -r0.5 -m $rcvr.sum
   psrsmooth -W -t UD8 $rcvr.sum
   ```

7. Target `toas`: Use `pat` to calculate the final narrowband TOAs from the .ff files and template and write them to a .tim file in tempo2/IPTA format, adding appropriate flags. Calculate errors using MCMC. Then use the same .ff files to calculate wideband TOAs based on the 12.5-year wideband templates, and write them to a .tim file with a similar format and flags.

   Commands:

   ```
   echo 'MODE 1' > $rcvr.nb.tim
   pat -A FDM -e err=num -C chan -C subint -C snr
       -C wt -C flux -C fluxe -f "tempo2 IPTA" -X
       "-proc 15y -pta NANOGrav -ver TEMP
       $(XFLAG)" -s $rcvr.sum.sm -M $rcvr.fflist
       >> $rcvr.nb.tim
   pptoas.py --print-flux --quiet
       --flags=proc,15y,pta,NANOGrav,ver,TEMP$(XFLAG)
       -m $rcvr.12y.x.avg_port.spl -d $rcvr.fflist
       -o $rcvr.wb.tim
   ```

## Acknowledgements

I thank Paul Demorest, who put together the current TOA generation process and wrote several of the software tools used to implement it, for answering a series of questions that made the current form of this document possible. I also thank Michael Lam and Haley Wahl for providing feedback on early drafts.

## References

Alam, M. F., Arzoumanian, Z., Baker, P. T., et al. 2021, ApJS, 252, 4, doi: 10.3847/1538-4365/abc6a0

Demorest, P. B. 2007, PhD thesis, University of California, Berkeley. https://www.cv.nrao.edu/~pdemores/thesis.pdf

Kurosawa, N., Kobayashi, H., Maruyama, K., Sugawara, H., & Kobayashi, K. 2001, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 48, 261, doi: 10.1109/81.915383

Offringa, A. R., de Bruyn, A. G., Biehl, M., et al. 2010, MNRAS, 405, 155, doi: 10.1111/j.1365-2966.2010.16471.x