GINAN YAML CONFIGURATION GUIDE

ID: GA-YML-01 Version 01 Dated 7 Jul 2023

Contact: clientservices@ga.gov.au

Ginan v2 – the new yaml file configuration guide.

See also:
https://geoscienceaustralia.github.io/ginan/page.html?c=on&p=defaultConfiguration.md
https://geoscienceaustralia.github.io/ginan/page.html?c=on&p=ginanConfiguration.md

The files

Ginan is controlled by "yaml ain't markup language" (yaml) configuration files. The Ginan yaml files are formatted in hierarchical sections that are delimited by indentation. For more information on the yaml format refer to the standard reference. Yaml files control all aspect s of a Ginan processing including; the data that Ginan looks for, how that data is processed and the type of outputs that are produced. Ginan yaml files can get very big and complex - there are over a thousand possible parameter settings within Ginan. Because of this, for version 2, yaml files can be broken up into parts – which are themselves yaml files. This means that you can dedicate individual files to certain tasks, such as having a file that controls the inputs, another that controls outputs and third that determines processing. Hopefully this allows for better and more concise control of Ginan. A user can use the same input and output yaml's, while studying the effects of different processing options.

For example, in this file ex204.yaml, four other yaml files are specified:

```
inputs:
    include_yamls:
        - large_post_network.yaml
        - network_clocks_and_biases.yaml
        - mongo_outputs.yaml
        - example_inputs.yaml

outputs:
    metadata:
        config_description:     "ex204"
```

ex204.yaml

The order that yaml files are specified is not critical. Ginan will look for information in the order it needs it, rather than the order in a file like ex204.

The Ginan GitHub repository contains examples of yaml files. A user is free to create their own versions – the obvious guiding principles are:

- Files should contain parameter settings that deliver the outcomes the user requires,
- Different yaml files shouldn't contain parameter settings that contradict each other,
- Be wary of behaviour that arises from the fact a parameter wasn't explicitly set and a default value was applied.

Another feature is that Ginan can operate using a hierarchy of information sources. For example, clock information could come from the Kalman filter. If that is missing, then Ginan can use a precise clock file. Failing that, the system can pick up the broadcast clock information.

Data and Metadata Inputs

```
inputs:
    root_directory: products/

    snx_files:  [ igs19P2062.snx, tables/igs_satellite_metadata_2203_plus.snx, 
IGS1R03SNX_20191950000_07D_07D_CRD.SNX ]
    atx_files:  [ igs20.atx                                     ]
    blq_files:  [ OLOAD_GO.BLQ                                  ]
    erp_files:  [ igs19P2062.erp                                ]
    egm_files:  [ tables/EGM2008.gfc                            ]
    jpl_files:  [ tables/DE436.1950.2050                        ]
    tide_files: [ tables/fes2014b_Cnm-Snm.dat                   ]

    satellite_data:
        nav_files: [ brdm1990.19p                               ]
        clk_files: [ IGS2R03FIN_20191990000_01D_30S_CLK.CLK     ]
        bsx_files: [ IGS2R03FIN_20191990000_01D_01D_OSB.BIA     ]
        sp3_files:
        - IGS2R03FIN_20191980000_01D_05M_ORB.SP3
        - IGS2R03FIN_20191990000_01D_05M_ORB.SP3
        - IGS2R03FIN_20192000000_01D_05M_ORB.SP3

    troposphere:
        orography_files: orography_ell_5x5
        gpt2grid_files:  gpt_25.grd
        vmf_files:
        - grid5/VMF3_20190718.H00
        - grid5/VMF3_20190718.H06
        - grid5/VMF3_20190718.H12
        - grid5/VMF3_20190718.H18
        - grid5/VMF3_20190719.H00
```

Metadata input configurations

This inputs yaml file describes all of the auxillary data and metadata inputs for this run of Ginan.

The `root directory` is where Ginan will look for all the metadata products to be used.

`snx_files`: contain a priori information about the stations that are to be used in the Ginan processing. From the SINEX files Ginan extracts information about the station location, receiver, and antenna type as well as satellites specific meta data necessary to process the observations. Various sinex file types can be read by Ginan including station sinex files and satellite meta data sinex files. `atx_files`: specify the relevant antex files. These files contain details on satellite and receiver antennas including data on phase center variation (PCV) and phase center offset (PCO).

`blq_files`: contain station specific ocean loading corrections.

`erp_files`: are the earth orientation parameters to be used in the processing.

The next three file types are new to version 2. Strictly speaking they are not needed unless you are estimating satellite orbital States, but we recommend that they are included:

`egm_files`: is the spherical harmonic coefficients representing the Earths gravity field.

`jpl_files`: the planetary ephemeris file is used to get the Sun and Moon position for determining the nominal attitude for the GNSS satellites and for computing the gravitational effect of the Sun, Moon and solar system planets on satellites

`tide_files`: the spherical harmonic coefficients representing global ocean tidal model, used for computing the gravitational effect of the ocean tides on satellites

Note that the egm and tide files are only relevant for the orbit propagator or integrator.

The `satellite_data:` files are relevant here because this is part of a post-processed example. These would not be needed for real-time processing unless there was a requirement to use a predefined set of biases for the analysis.

`satellite_data:` includes general satellite ephemeris and clock offset in the form of a navigation file, IGS RINEX clock format file and a bias file in the new IGS BIA sinex format. In this case SP3 files provide precise orbit positions.

The   section contains information about the troposphere model to be used in the data processing. Information about the location of the GPT2 troposphere model files and the VMF3 grid files are given.

Observation Inputs

The yaml below large_post_network.yaml demonstrates how to specify a range of observation inputs (RINEX files) from a number of stations, to be processed by Ginan. The `inputs_root: data/` identifies the location of the files. The `rnx_inputs:` tells Ginan the files are in RINEX format. This kind of yaml observation input might be used for post-processing a large network of stations.

```yaml
inputs:
    gnss_observations:
        inputs_root: data/
        rnx_inputs:
            - "AREG*.rnx"
            - "ASCG*.rnx"
            - "BAKO*.rnx"
            - "BOGT*.rnx"
            - "CEDU*.rnx"
            - "COCO*.rnx"
            - "CPVG*.rnx"
            - "CRO1*.rnx"
            - "CUSV*.rnx"
            - "DARW*.rnx"
            - "DGAR*.rnx"
            - "DJIG*.rnx"
            - "FAIR*.rnx"
            - "FFMJ*.rnx"
            - "GANP*.rnx"
            - "HERS*.rnx"
```

large_post_network.yaml

Alternatively, Ginan can be configured to take and process observations from real-time streams. This kind of configuration is shown in tiny_realtime_network.yaml below. The directory name has been replaced by the stream mount point in `inputs_root:` and the type of stream has been identified using `rtcm_inputs:`.

`satellite_data:` and broadcast correction streams can also be accessed in this way.

```
inputs:
    gnss_observations:
        inputs_root:    "https://<USER>:<PASS>@ntrip.data.gnss.ga.gov.au/"
        rtcm_inputs:
            - "ALIC00AUS0"
            - "COCO00AUS0"
            - "LHAZ00CHN0"
            - "KZN200RUS0"

    satellite_data:
        inputs_root:    "https://<USER>:<PASS>@ntrip.data.gnss.ga.gov.au/"
        rtcm_inputs:
            - "BCEP00BKG0"
            - "SSRA00BKG0"

processing_options:
    ssr_inputs:
        ssr_antenna_offset: APC
```

tiny_realtime_network.yaml

Outputs

This next file contains a range of configuration settings that determine what the user wants Ginan to output from the processing. In this case the file is called network_clocks_and_biases.yaml, but the user can give their file any name which is appropriate and meaningful to them.

The `root_directory:` is the directory where the user wants all the outputs to be put. In this case it is `outputs/<CONFIG>/` where `<CONFIG>` is an alias whose value is derived from the config_description: setting.

```
Outputs:
    metadata:
        config_description:     "ex204"
```

It could just be an absolute pathname as well.

Ginan can produce trace files which summarise information read or computed as the processing proresses. Trace files contain a lot of information on the internal workings of Ginan. A user can control the level of information in a trace file using the `level` parameter. The higher the number (up to 6) the more verbose the information written to the Trace files.

There are two types of trace file:

`output_stations:` informs the user as to what is happening in the processing for each station,

`output_network:` informs the user what the Kalman filter is doing to combine that data into a solution.

Then the user can turn flags to true to have certain types of data included in the trace files like `output_residuals:` and the `output_residual_chain:` will tell the user about the components used in the processing.The `output_config: true` line tells Ginan to write the contents of input yaml configuration setting file/s into the top of output trace files so the user has a record of the settings used.

`ppp_sol`, when true, gives the precise point positions in a file of their own.

```
outputs:
    root_directory:          outputs/<CONFIG>/
    trace:
        output_stations:             true
        output_network:              true
        level:                       3
        station_filename:            <STATION>_<YYYY><DDD><HH>.TRACE
        network_filename:            <STATION>_<YYYY><DDD><HH>.TRACE
        output_residuals:            true
        output_residual_chain:       true
        output_config:               true
    ppp_sol:
        output:                      true
```

If the user was running an ionosphere estimation solution and wanted to save the Global Ionosphere Model (GIM) spherical harmonic States, setting the `ionex output:` `true`, and specifying a directory and filename would save the output to that location.

Similarly a user can choose to output the slant total electron content estimates if they're doing uncombined undifferenced processing – `ionstec:`.

`bias_sinex:` allows a user to turn on saving estimated bias States to a standard bias sinex format file.. In the example below the satellite biases are true but the receiver biases are false (would not be produced) and the output interval is set to 900 seconds. Seconds are the default units.`clocks:` true, produce the clock file and include the ambiguity resolved clocks into the file.sinex output: true, turns on the standard position sinex file output. This file also contains the summary of the metadata that was used to produce the solution - all of the antenna offsets, receiver and antenna types are documented in the output syntax file. That makes it a good reference source.

```
    ionex:
        output:                      false
        directory:                   ./
        filename:                    AUSG<DDD>0.<YY>I
    ionstec:
        output:                      false
        directory:                   ./
        filename:                    IONEX.stec
    bias_sinex:
        output:                      true
        output_rec_bias:             false
        code_output_interval:        900
        phase_output_interval:       900
    clocks:
        output:                      true
        output_ar_clocks:            true
    sinex:
        output:                      true
```

`erp:` Earth Rotation Parameters written out into a standard ISO ERP file format.

`trop_sinex:` troposphere estimates into a troposphere sinex file format. Generally, it’s best to have the source of the troposphere estimates as `[KALMAN]` from the Kalman filter.

```
    erp:
        output:                     true
    trop_sinex:
        output:                     true
        sources:                    [KALMAN]
```

Processing options

Options start with the data interval – the example shows using 60 second data – `epoch_interval`.

`wait_next_epoch:` how long Ginan will wait for the next epoch of data to arrive. For post processing, the user can set that to be a very long time because there is usually no problem waiting any amount of time for all the rinex files to be read together as a single epoch. The example below will wait an hour for the next epoch interval to come in. If the user is processing in real time, that should be set to less than the sampling interval to be used. If one Hertz data is to be used, then Ginan would wait for the next one or two seconds for all of the data to come in and then process it – so 3 seconds in total.

`wait_all_stations:` If a user is doing network processing where they're generating clocks and biases for instance, Ginan will wait for thoese three seconds for the next epoch of data, then wait an additional one second for the whole network that is defined to come in, and then process what is there.

`fatal_level_message:` when set to 1 will stop Ginan processing on any error message. It can be set up to 3, which tells Ginan to keep processing no matter what the errors being generated.

`process_modes:` in version 1 of Ginan there was the concept of User and Network mode. All the version 2 code however is accessed using this `ppp: true` mode. Within this mode there are options to use the ionosphere-free or undifferenced-uncombined set by parameters discussed later.

```
processing_options:
    epoch_control:
        epoch_interval:             60
        wait_next_epoch:            3600     # Wait up to an hour for next data
point - When processing RINEX causes PEA to wait a long as need for last epoch
to be processed.
        wait_all_stations:          1
        fatal_message_level:        1
    process_modes:
        ppp:                        true
```

In `gnss_general:`

`elevation_mask:` hides observations from satellites below the specified number of degrees above the horizon.

`rec_reference_system:` in this example is set to GPS.

Then there are options about the information used from sinex files, `require_antenna_details:` in this example is set to true. It can be set false in which case Ginan will produce a solution without reference to antenna details such as PCV and PCO. A position can be produced but the height might be a bit off.

`require_apriori_positions`: is usually set to false because a user can get good position information from the single point position estimate using the code data. A good a priori position is not required unless the user is using that station as a reference station in their solution.

Ginan will provide alerts to say when processing has continued without antenna or a priori information.

```
    gnss_general:

        elevation_mask:                 10      # degrees
        rec_reference_system:           GPS
        require_antenna_details:        true    # (bool)
        require_apriori_positions:      false   # (bool)
```

`sys_options:` this section is giving the user options around the use of constellations. This example instructs Ginan to only use GPS – the other constellations are commented out and you could include Beidou. If a constellation is included, it makes sense that the user provides observation data from that constellation. If no observation data is provided for a constellation nothing will get processed. In this example: process GPS, do not do ambiguity resolution in this case, and do not reject satellites that are in eclipse.

There are options to set the way differential code biases (DCB) are handled for receiver and satellite. There are clock and code frequencies specified - setting `zero_receiver_dcb: true` would set the DCB between L1W and L2 to be zero, unless that receiver does not track L1W, in which case it's going to select L1C and L2W.

`network_amb_pivot:` in order to eliminate the ranked efficiency that comes from the phase bias and ambiguities, this fixes one ambiguity per satellite and one ambiguity per station, which eliminates sensitivity - trade off the ambiguities against the phase biases, essentially.

`clock_codes:` the user can define how they want their output clocks to be defined. In this case, the standard L1WL2 IGS convention for clocks has been specified, even though it's probably not the most obvious one to choose these days. But that's what the IGS uses so this goes with that convention. These are the codes to be used in this particular processing example - dual frequency using L1W.

```
        sys_options:
            gps:
                process:                true
                ambiguity_resolution:   false     # Do not turn on AR if using
ppp: - use_if_combo: true
                reject_eclipse:         false
                zero_receiver_dcb:      true
                #zero_satellite_dcb:     true
                network_amb_pivot:      true
                clock_codes:            [ L1W, L2W ]
                code_priorities:        [ L1W, L1C, L2W ]
            # gal:
            #    process:                true
            #    ambiguity_resolution:   false
            #    reject_eclipse:         false
            #    code_priorities:        [ L1C, L5Q, L1X, L5X ]
            # glo:
```

```
            #     process:                true
            #     ambiguity_resolution:   false
            #     reject_eclipse:         true
            #     code_priorities:        [ L1P, L1C, L2P, L2C ]
            # qzs:
            #     process:                true
            #     ambiguity_resolution:   false
            #     reject_eclipse:         true
            #     code_priorities:        [ L1C, L2L, L2X ]
```

`gnss_models:` this section is about applying models for GNSS processing. There are a large number of options. Following are explanations of some of the more significant options.

`sat_attitude:` The choices here are to use nominal, which is to use the nominal attitude based on the Sun's location all the time. Or the user can turn on Ginans yaw attitude modelling code which is based on the models used in standard IGS analysis software for all the constellations. At the moment Ginan has yaw attitude modelling completed for GPS Galileo.

`rec_attitude:` use [NOMINAL] for the time being. A sinex format receiver attitude model is under development.

`troposphere:` a VMF3 or GPT2 model can be selected given that the input model files are provided (in a previous section).

```
    gnss_models:

        sat_attitude:
            sources:            [MODEL]
        rec_attitude:
            sources:            [NOMINAL]
        troposphere:
            model:              vmf3   # gpt2
```

`ionospheric_component:` this is where the user can specify the use of an ionosphere free or uncombined processing model.

`common_ionosphere:` should be kept as true and means the same ionosphere model is used for code and carrier wave.

`use_if_combo:` if set to false uses data in an uncombined way. If true then will use an ionosphere linear combination choosing frequency observations that are present then based on the order specified previously.

There is no spatial correlation of the ionosphere – so no correlation between stations – but there is temporal correlation defined by process noise – coming in a later part of this file.

`model_error_checking:` is to determine what to do with bad data. Ginan doesn't ever actually remove observations from processing, it “down weights” them in the processing. This parameter set specifies that when a bad observation is found it will down weight the measurement variance by a factor of 10,000 in this case. So if the Kalman filter through its checking finds a bad observation, it will then apply multiply the variance the measurement variance by 10,000 and that eliminates it from having weight in the solution.

`ambiguities:` when to reset ambiguities and the solution when a problem is found.

`outage_reset_limit:` defines a data gap to be before it resets the ambiguity automatically In this case, the setting is 10. Ginan can have 10 missing observations before the ambiguity will reset automatically.

Because this example is using 60 second data, Ginan would allow 600 seconds before the ambiguity was reset in the filter.

`phase_reject_limit:` if Ginan detects a four Sigma outlier twice in a row (that number 2), it will then reset the phase ambiguity and start estimating a new one.

`reinit_on_all_slips:` if true means that if all channels lose lock simultaneously, then Ginan have a data gap and reset all the ambiguities - because it won't be able to determine which ones might be OK and which ones are not.

```
        ionospheric_component:
            common_ionosphere:  true   # Code and Phase measurment share the 
same ionosphere
            use_if_combo:       false
    model_error_checking:

        deweighting:
            deweight_factor:            10000
        ambiguities:
            outage_reset_limit:          10
            phase_reject_limit:          2
            reinit_on_all_slips:         true     # (bool) Any detected slips 
cause removal and reinitialisation of ambiguities
```

If ambiguity resolution was turned on (see above) this is where its options are defined. The methods are the same as those for version 1. The only difference is that rounding is not recommended for an uncombined solution.

`outlier_screening:` the number of iterations the Kalman filter should attempt to remove bad observations before it gives up. In this example 50 iterations. If the user sets that number too large, usually it doesn't hurt because the preprocessor does a pretty good job of getting all the bad data before it comes into the filter. If the preprocessor is not working well for some reason, then the filter will iterate removing bad observations weighting them down with that factor of 10,000 until it's either got them all or until it hits the 50 limit - and then it will just continue processing, even if it determines there is bad data in the in the solution, so its one of the things that's worthwhile checking the trace file - how often that limit is being hit. We haven't seen it hit 50. If it does, it probably means there's some other problem with the data and not just single outliers.

`max_prefit_removals:` the option before the data enters the filter to do a prefit scan and remove bad data at the prefit step, which is a bit cheaper in terms of computational time.

`sigma_check:` from before, a 4 sigma outlier is flagged as bad and will be down weighted in the filter.

`rts:` Ginan can do a Rauch-Tung-Striebel (RTS) backwards smoothing run. There is no point doing that for real-time processing because you never get to the "end" when the backward smoothing would start – so set to false. For post-processing it makes sense, but as mentioned previously, not when doing ambiguity resolution.

`minimum_constraints:` Ginan can apply a transformation of your estimated position states to the reference frame defined by a priori coordinates and the sinex file. Transformations can be translations, rotations and scales – in this case scale is set false. If the user sets minimum_constraints to true, they have the choice of doing the transformation every epoch, apply the minimum constraint and output the results every step of the common filter. Alternatively, if set to false, Ginan will just apply the minimum constraint when it gets to the end of the data, apply it and then do the backward smoothing. If the user wanted to get minimally constrained real-time products out, they would set this `once_per_epoch: true`, so that they would get a minimally constrained solution at every step of the of the filter.

`full_vcv:` Ginan provides the option to weight stations in that minimum constraint – the user can specify what the North-East-Up variance on the measurements going into the minimal constraint calculation should be. There are other options to weight that by the variance covariance matrix or some scaled version of the variance covariance matrix.

`max_filter_iterations:` this function has the same outlier filtering and removal option that the main filter has. A user can tell the filter to iterate the minimum constraint calculation up to 20 times removing bad stations from that minimum constraint calculation as it goes.

`sigma_threshold:` these options are all common – in this example the Sigma threshold for removing bad stations has been set to 3 instead of the default 4.

```
    #ambiguity_resolution:
    #    elevation_mask:                 15
    #    lambda_set_size:                200
    #    narrow_lane:
    #        mode:                       lambda_bie        # off, bootst, 
lambda, lambda_alt, lambda_al2, lambda_bie
    #        success_rate_threshold:     0.99
    #        solution_ratio_threshold:   30
    filter_options:

        outlier_screening:
            max_filter_iterations:      50
            max_prefit_removals:        3
            sigma_check:                true
        rts:
            enable:                     true
    minimum_constraints:

        enable: true
        once_per_epoch: false            # (bool) Perform minimum constraints 
on a temporary filter and output results once per epoch
        translation:
            estimated:  [true]

        rotation:
            estimated:  [true]

        scale:
            estimated:  [false]
            #sigma:      [1]
        #full_vcv:       true
        #scale_by_vcv:   true
        max_filter_iterations:              20
        max_prefit_removals:                3              # (int) Maximum 
number of measurements to exclude using prefit checks before attempting to 
filter
        outlier_screening:
            chi_square_mode:                none
            chi_square_test:                false
            sigma_check:                    true
            sigma_threshold:                3
            w_test:                         false
        station_noise:
            global: [0.01, 0.01, 0.01]
```

`station_options:` a user can define things in this section that were not in the metadata section. In this example there were some stations missing from the sinex file If there isn’t a sinex file entry for a particular site, a user can just add the site into the station options list along with receiver, antenna and eccentricity information for them. The other thing a user can do is set aliases for sites. In this example USN7 has been given the alias `[PIVOT]` and the example shows how that is used later.

These aliases can be anything a user wants. It's just it's just a way of using those sites for various things without having to relist them each time.

```
station_options:
    RAEG:
        receiver_type:  "LEICA GR25"
        antenna_type:   "LEIAR20         NONE"
        eccentricity:   [0.0000,   0.0000,   0.5793]
    DUMG:
        receiver_type:  "LEICA GR25"
        antenna_type:   "LEIAR25.R4      LEIT"
        eccentricity:   [0.0003,  -0.0005,   0.4278]
    GAMB:
        receiver_type:  "TRIMBLE NETR9"
        antenna_type:   "TRM59800.00     NONE"
        eccentricity:   [0.0000,   0.0000,   0.0000]
    GLPS:
        receiver_type:  "JAVAD TRE_G3TH DELTA"
        antenna_type:   "ASH701945B_M    SCIT"
        eccentricity:   [0.0000,   0.0000,   0.0083]
    KITG:
        receiver_type:  "SEPT POLARX5"
        antenna_type:   "TRM59800.00     SCIS"
        eccentricity:   [0.0000,   0.0003,   2.0374]

    USN7:
        aliases:        [PIVOT]

    AGGO:
        aliases:        [ADD_CLOCK_RATES]
```

`estimation_parameters:` this is very similar to Ginan version one. There are two parts – a `stations:` and a `satellite:` section.

In the stations: section, a user can define the error model to be used for the observations is. In this example, an elevation dependent model is to be applied to the observations to set the measurement variances or sigmas in this case - variance, the square root of variance code Sigma and phase Sigma.

A user can specify what states to estimate and the starting uncertainty and process noise to be applied to those states is. They can be defined per station or for all stations after the `global:` line - apply the following things for all stations that are in the solution. For example, all stations have position estimated with this uncertainty of 1 metre and no process noise. If the sites were kinematic you would set that process noise to be a non 0 number.

```
estimation_parameters:
    stations:
        error_model:        elevation_dependent
        code_sigmas:        [0.3]
        phase_sigmas:       [0.002]
        global:

            pos:
                estimated:          [true]
                sigma:              [1]
                proc_noise:         [0]
```

`clk:` estimated true, sigma is 1000 metres and process noise of 10 metres per square root second.

`amb:` sigma is 100 cycles, not metres.

`trop:` troposphere is estimated with a priori sigma and a process noise. The troposphere is probably one place where a First order Gauss-Markoff process noise would be useful where there is a slow highly correlated part of the atmosphere with a more variable part which could be parameterized.

The zenith tropospheric delay has gradients in the north, south and east west directions parameterised through the gradients.

`ion_stec:` in TCU – in this example there is an initial uncertainty of 100 TCU and then this is to be allowed to change 2 TCU per square root second - the constraint.

```
            clk:
                estimated:          [true]
                sigma:              [1000]
                proc_noise:         [10]
            amb:
                estimated:          [true]
                sigma:              [100]
                proc_noise:         [0]
            trop:
                estimated:          [true]
                sigma:              [0.3]
                proc_noise:         [0.0001]
            trop_grads:
                estimated:          [true]
                sigma:              [0.03]
                proc_noise:         [1.0E-6]
            ion_stec:
                estimated:          [true]
                sigma:              [100]
                proc_noise:         [2]
            phase_bias:
                estimated:          [true]
                sigma:              [10]
                proc_noise:         [0]
            code_bias:
                estimated:          [true]
                sigma:              [20]
                proc_noise:         [0]
            #GPS:
            #    L2W:
            #        amb:
```

```
            #            estimated:      [false]
            #            sigma:          [1e-8]
            #            proc_noise:     [0]
```

Earlier on in the example the station USN7 was given the name `PIVOT`. USN7 has a good atomic clock and in this example is set to be the clock reference through the label PIVOT. In this case nothing is to be estimated including code and phase biases.

```
        PIVOT:
            clk:
                estimated:          [false]    # Set reference (pivot) station 
clock
            code_bias:
                estimated:          [false]
            phase_bias:
                estimated:          [false]
        ADD_CLOCK_RATES:
            clk_rate:
                estimated:          [true]
                sigma:              [10]
                proc_noise:         [1e-8]
```

`satellite:` has a global option to affect everything for all the satellites - estimating the clocks, clock rate, phase and code biases for all the satellites as well.

This example is aimed at generating a solution and removing the rank deficiency, so set the code biases on the GPS system L1 and L2W to be 0, with a very small sigma. They will still appear in the output but because they have a very small sigma of 10 to the minus 8 they are constrained them onto that value in this solution for the GPS system.

```
    satellites:

        global:

            clk:
                estimated:          [true]
                sigma:              [1000]
                proc_noise:         [1]
            clk_rate:
                estimated:          [true]
                sigma:              [10]
                proc_noise:         [1e-9]
            phase_bias:
                estimated:          [true]
                sigma:              [10]
                proc_noise:         [0]
            code_bias:
                estimated:          [true]
                sigma:              [10]
                proc_noise:         [0]
        GPS:
            L1W:
                code_bias:
                    sigma:              [1e-8]      # this implements B(s,GPS-
L1W)=0
                    process_noise:      [0]
                    apriori_value:      [0]
            L2W:
                code_bias:
                    sigma:              [1e-8]      # this implements B(s,GPS-
L2W)=0
                    process_noise:      [0]
                    apriori_value:      [0]
    eop:
        estimated:  [true]
        sigma:      [10]
    eop_rates:
        estimated:  [true]
        sigma:      [10]
```

network_clocks_and_biases.yaml