Deliverable D3.2
Location algorithm software (release note)

Project acronym:	PERFORMING RAIL
Starting date:	01/12/2020
Duration (in months):	30
Call (part) identifier:	H2020-S2R-CFM/OC-IP/CCA-201X-0X
Grant agreement no:	101015416
Due date of deliverable:	Month 12
Actual submission date:	31-12-2021
Responsible/Author:	miquel.garcia@rokubun.cat (ROK)
Dissemination level:	PU/CO
Status:	Draft

Document history		
Revision	*Date*	*Description*
1.0		First issue

Report contributors		
Name	**Beneficiary Short Name**	**Details of contribution**
Miquel Garcia	ROK	Document preparation.

Funding
This project has received funding from the Shift2Rail Joint Undertaking (JU) under grant agreement No 101015416. The JU receives support from the European Union's Horizon 2020 research and innovation programme and the Shift2Rail JU members other than the Union.

Table of Contents

1. Executive Summary

One of the objectives of WP3 was the provision of a software that implements the GNSS Hazards outlined in D3.1 of the project. This software is to be integrated in the Railway signaling simulator in order to simulate the impact of various GNSS hazards such as lack of visibility of signal degradation, in the signaling system. Since the software will not be delivered to third parties (only to the University of Birmingham for its integration in the Signaling simulator), this release note is submitted instead. As such, it details the features of the software, its architecture, as well as the instructions to have an instance of the GNSS simulator up and running.

2. Abbreviations and acronyms

Abbreviation / Acronyms	Description
CSV	Comma Separated Values
GLS	GNSS Location Simulator
GNSS	Global Navigation Satellite System
PVT	Position Velocity and Time
RSS	Railway Signaling Simulator
TCP	Transmission Control Protocol

3. Background

The present document constitutes the Deliverable D3.2 “Design document of the Location algorithms” and it incorporates the work undertaken under T.3.3. The task follows T3.1 (with some brief overlap to consolidate the design in terms of implementation) and consists in the implementation of the location algorithms. The implementation language for these algorithms is C (for the data processing and mathematical algorithms) and Python (for data exchange, interfacing, ...). Rokubun’s guidelines for software implementation will be adopted in this task, which includes:

- A Continuous Integration and Deployment (CI/CD) philosophy:
- Single Responsibility Principle, where the software design phase will emphasize implementation of methods that perform only a single task. This enforces modularity, reusability of code and conforms to DRY (Don’t Repeat Yourself) philosophy.
- “Keep It Simple” approach to avoid large and non-reusable code.
- Usage of software version control (GIT) and a DevOps platform (Gitlab) that allows project management, track issue and unit test orchestration.
- Usage of Docker virtualization containers to ensure a sandbox development (maximizing the traceability of the development environment)

This approach ensures a structured, repeatable and automatable software development process.

There is no specific protocol for the delivery of software. As agreed with the project technical officer as well as the project coordinator (as per ad-hoc meeting on Monday 21st December 2021), this release note is delivered instead of the software. Therefore, this document should be treated as proof of delivery for the software component, whose performance and functionality will be demonstrated in the upcoming testing campaign of WP5 (2022).

4. Objective/Aim

This document constitutes the release note of the software component developed during T3.2 of the Performing Rail project. It contains an overview of the architecture as well as the features of the software. This software-based GNSS simulator will be used during the validation phase of the project to simulate various GNSS feared events and its impact in the context of the railway moving block.

The following deliverables are considered as input for this WP:

- PERFORMINGRAIL D3.1 Location algorithm description

This deliverable will be used as inputs for the Signaling simulator tests conducted in WP5.

5. Overview of the software

The GNSS location simulator (GLS) is a tool developed in the context of Performing Rail with the objective of simulating a GNSS receiver as well as the potential hazards that can occur in GNSS signals. The main features of the simulator are the following:

- Simulation of GNSS raw measurements (pseudoranges, carrier phases and Doppler measurements) for a given static position or trajectory
- Supports multi-constellation (GPS, Galileo, Beidou)
- Supports multi-frequency (L1, E1, L2, E5a, E5b, …)
- Simulates GNSS orbits and clocks, ionospheric and tropospheric delays, geometric range, …
- Compatible with real time simulations
- Simulates signal degradation such as signal noise as well as limited visibility (i.e. GNSS hazards referred in D3.1 deliverable of the Performing Rail project)
- The simulator also incorporates a processing engine that estimates the PVT (Position Velocity and Time) solution using GNSS raw measurements.
- Provision of the formal error of the PVT estimates.
- Software encapsulated in a Docker[1] container to ensure portability across platforms, freeze dependencies and ensure traceability.

The simulator has been designed to be incorporated into the Railway signalling simulator (RSS) developed by University of Birmingham, to study the impact of malfunctioning GNSS receivers (or into railway operations.

In summary, the processing workflow of GLS is as follows:

1. RSS will send the train position and epoch to be simulated (i.e. true/reference position) to the GLS.
2. GLS will compute the GNSS raw measurements from the given position and epoch, incorporating all the signal terms and delays (geometric range, atmospheric errors, …) as well as potential degradations in the GNSS signal (orbit and clock errors, ionosphere model mismatch, limited visibility, noise increase, …)
3. The simulated GNSS raw measurements will then be fed to the positioning engine that will estimate the PVT solution based on these GNSS raw measurements.
4. The estimated PVT solution will then be fed back to RSS

Note that in the ideal case (no GNSS errors simulated), the position sent to GLS (step 1) should be almost equal to the one computed by GLS (step 4). However, due to the presence of errors and delays, this will usually differ.

[1] https://www.docker.com/

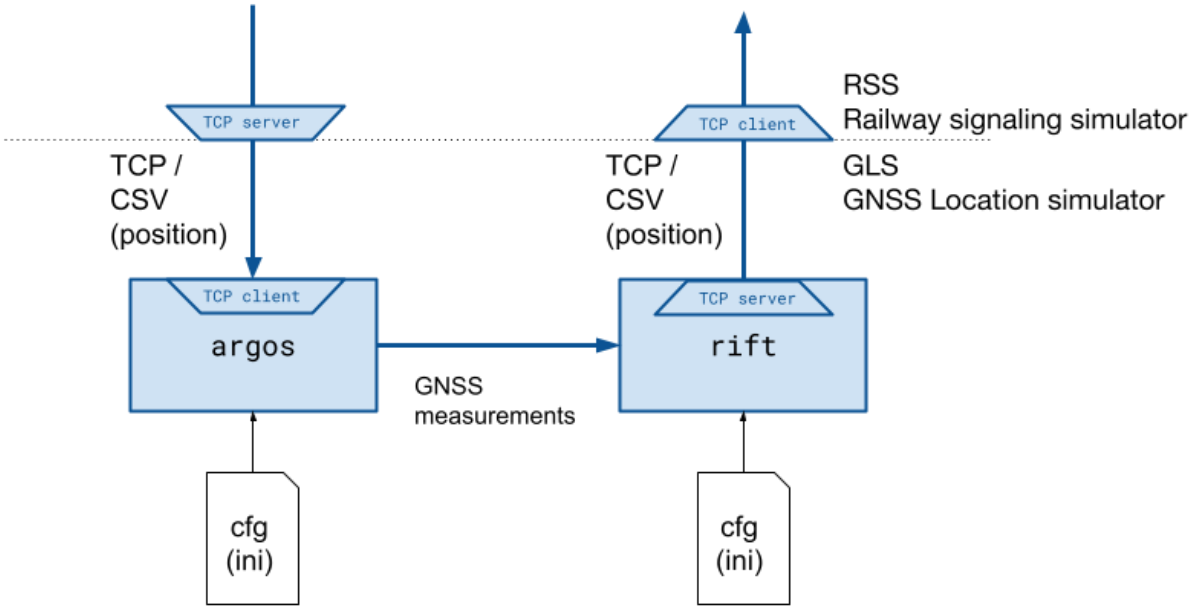
6. Requirements

The following software requirements are needed for the GLS:

- Rokubun core, version 13.14.0-r8 or newer. This requirement contains the two fundamental components needed by GLS: (a) measurement simulator and (b) positioning engine. This core incorporates the GNSS hazard simulation implemented during WP3.
- Docker, version 20.10.10: Required to build the images that encapsulate the simulation software.
- docker-compose, version 1.29.2: required to deploy the Docker image that encapsulates the simulation software. Albeit not used in production, this component is employed fundamentally in the development of the software

7. Software architecture

A system-level overview of the architecture is shown in the following figure (interfacing with the Railway Signaling Simulator RSS):



Block diagram of interfaces between Rokubun's GNSS simulator and the railway signalling simulator

As already mentioned in D3.1, RSS will communicate to GLS via a TCP protocol using a CSV format to submit and receive position and epochs. GLS is composed of 2 different entities, which are part of Rokubun core software for GNSS data processing and incorporates the GNSS hazards outlined in D3.1 and implemented in WP3. These entities are:

- **argos**, which simulates the GNSS raw measurements given an input position and a certain scenario (defined by GNSS orbits and clocks as well atmospheric conditions)
- **rift** is the actual position engine that processes the GNSS raw measurements (simulated by **argos**) in order to deliver a position (PVT solution).

These components reside in Rokubun core both as executable programs as well as libraries so that they can be incorporated into bigger software components such as e.g. GLS. In fact, GLS is a TCP server implemented in C that calls both **argos** and **rift** library methods. This TCP server is a software component that has been specifically designed and implemented for Performing Rail and its software repository (git) resides in Rokubun internal Gitlab repository under the address:

- http://gitlab.rokubun.tech/customer/performingrail/gnss_simulator

A copy of this source code can be requested to the author of this document.

The simulation software accepts several configuration files that drive the execution of the simulator and are responsible for the type of simulation to be exercised. These configuration files constitute the *simulator scenarios*, which are briefly described in the following section.

8. Scenario definition

A GLS scenario determines various factors to be considered in the simulation:
- GNSS orbits and clocks errors
- Ionospheric delay
- Tropospheric modeling
- GNSS Hazard to simulate
- Constellations and satellites to simulate or exclude
- …

This is done by using 2 configuration files, one for **argos** (to simulate the GNSS raw measurements) and another for **rift** (to compute the PVT solution from the GNSS raw measurements). These configuration files, in INI format and described in the Annex of this document may contain several files (such as broadcast files, orbits and clocks in SP3 format, ionospheric maps, visibility masks, ….). All these files constitute a scenario and should be packaged within the same folder. Within the scenario folder, there are 2 mandatory files (that should respect the name provided):
- `argos.ini`, with the configuration of the **argos** simulator (see Appendix A for details on the configuration).
- `rift.ini`, with the configuration of the **rift** positioning engine (see Appendix B for details on the configuration).

As mentioned above, additional files (listed within the ini files) such as e.g. broadcast file should be present in the folder.

The simulator scenarios considered for the different test cases are stored in a git repository, within internal Rokubun Gitlab server:

- http://gitlab.rokubun.tech/customer/performingrail/scenarios

A copy of the scenario repository can be requested to the author of this document.

9. Docker integration and execution

As briefly mentioned above, Docker is used to encapsulate the software in a package that can be then run in any platform (Windows, Linux, Mac OS X) without the need to recompile the source code or install the necessary dependencies in the host system. A new Docker image is created whenever changes are made to the source code (either Rokubun core or GLS simulation server code) and uploaded to the internal Rokubun repository hosted at Amazon Web Services

- `387039469323.dkr.ecr.eu-central-1.amazonaws.com/gnss-simulator`

Access to this Docker image can be requested to the author of this document. This section explains the process of downloading the Docker image and executing it. To run the simulator, follow these steps in your command line:

1. Write an INI config file in your home directory `~/.aws/credentials` with the following contents (these are your credentials to access the AWS registry)

```
[default]
aws_access_key_id = <key_id>
aws_secret_access_key = <token>
```

Values for to the `<key_id>` and `<token>` can be requested to the author of this document

2. Login (using the IAM user credentials defined by the previous file) into Rokubun AWS registry, by issuing the following command:

```
$(aws ecr get-login --no-include-email --region eu-central-1)
```

3. Pull (i.e. get) the Docker image of the GNSS Location simulator from the AWS registry:

```
docker pull 387039469323.dkr.ecr.eu-central-1.amazonaws.com/gnss-simulator
```

4. Run the container with the GNSS location server with the following command:

```
docker run -p 9999:8888 -t
387039469323.dkr.ecr.eu-central-1.amazonaws.com/gnss-simulator
```

If you want to run a specific scenario, bind the folder containing the files of the scenario into the predefined `/scenario` folder of the container, like so:

```
scenario_folder=/path/to/scenario
docker run -p 9999:8888 -v ${scenario_folder}:/scenario -t
387039469323.dkr.ecr.eu-central-1.amazonaws.com/gnss-simulator
```

This will instantiate a container with the GNSS simulator server and will listen to messages in port 9999 of the host machine (i.e. machine running the Docker container). The port (`-p`) option implies that docker will forward any content from host machine port 9999 to container port 8888.

5. For a client application that needs to send messages to the container and receive a reply, you can use the same container, that already includes a client application for testing (which would play the equivalent role as the RSS simulator):

```
image="387039469323.dkr.ecr.eu-central-1.amazonaws.com/gnss-simulator"
server_container=`docker ps --filter ancestor=${image} -q`
server_ip=`docker inspect --format '{{range
.NetworkSettings.Networks}}{{.IPAddress}}{{end}}' ${server_container}`
message="2017-08-03,09:40:00.000,41.402434220,2.194859688,53.9370,RX1"
docker run ${image}  gnss_simulator submit -P 8888  -H ${server_ip} -I
${message} 2> /dev/null
```

Note, however, that other clients written in Python, Java, ... will also be able to communicate with the container in a similar fashion.

A. Appendix A: `argos` configuration file

The following file contains the description of the INI configuration file format for **`argos`**.

```
[global]

; Timer options, allows to select the epochs either by specifying an epoch
; or by an input file with the epochs.
; Note that only one option has to be present (either Interval or FromFile).
; if both are present in the configuration file, an exception will be
; thrown
; timer specified as interval:
;timer=interval,<starting epoch>,<ending epoch>,<time step>
timer=interval,2015-08-03 00:00:00.0,2015-08-03 23:59:59.0,30
; timer specified by means of a file
; The simulation epochs will be extracted from the file.
; The file must at least contain 2 columns to specify the epochs
; (1) the number of integer seconds elapsed since J2000 and (2) the
; fractional part of the seconds (double)
;timer=file,/path/to/file_with_epochs

; Trajectory input (overrides timer and receiver position information).
; Specify a position file if you wish
; to simulate a trajectory instead of a fixed point. The position file
; shall have an epoch and the coordinates of the rover(s) to simulate
; Supported formats: Rokubun GDF containing position entries
trajectory=${PATH}/test/executables/argos/moving_platform.gdf

; ephemerides file, can be repeated so that multiple broadcast files can be added
brdc_file = BRDC00WRD_R_20192710000_01D_MN.rnx

; Elevation threshold in degrees. Satellites under this elevation will not
; be output. This parameter is optional. If not set, the minimum elevation
; will be 0
min_elevation= 5

; File that contains the visibility mask. This mask will prevent satellites from
; being simulated
visibility_mask_file = visibility_mask.txt

; list of excluded constellations (separated by a comma). Constellations expressed
; with 3-letter RINEX codes
exclude_constellations=GLO,BDS

; list of excluded satellites (separated by a comma)
exclude_satellites=R13,E12,E30

; channels whose code, phase and Doppler will be simulated
channels=1C,2P

; Code of the noise observables (the value specified is the standard deviation
; in meters of the white additive Gaussian noise process). If not specified, 0 will
be
; assumed (i.e no noise)
code_noise=1.0

; Phase of the carrier observables (the value specified is the standard deviation
; in meters of the white additive Gaussian noise process). If not specified, 0 will
```

```
be
; assumed (i.e no noise)
phase_noise=0.03

[ionosphere]
; ionosphere model to compute the pre-fit residual (and/or stec apriori),
; see argos INI file for further reference
; ionosphere model [klobuchar;ionex;constant] (default: klobuchar)
; For ionex model, specify also the IONEX file
; for constant model, specify also the constant VTEC, in TECU.
model = klobuchar
;model = klobuchar[,/path/to/brdc/file]
;model = ionex,/path/to/ionex/file
;model = constant,10

[output]
; Optional file where the output of Argos, concerning the range model,
; will be output.
; If this option is not specified, no output will be generated
model=argos_model.txt

[satellite]
orbit_file = igs18561.sp3
clock_file = igs19601.clk_30s

; Information on WiFi Access Point location. If this block is contained, the
; time of arrival (RTT) of WAP will be also simulated
[wifi]
wap_position_file = positions.csv
;hwbias_file: bias.txt
;rtt_noise: 3.0

[receiver: SIM1]
; ECEF position of the receiver to simulate
position= 4797616.1400,166568.8800,4185526.7100

; File to input the clock, note that the input epochs must match
; the ones being simulated by Argos, otherwise the clock will not be
; properly handled. The format for this file must be a parameter
; data file format
clock_file=sim1_clk.pd

troposphere_model=niell,0.3,0.01,0.02

; More than 1 receiver can be added
[receiver: SIM2]
position= 4797616.1400,166568.8800,4185526.7100

troposphere_model=saastamoinen
```

B. Appendix B: `rift` configuration file

The following file contains the description of the INI configuration file format for **`rift`**.

```
[global]
; ephemerides file, can be repeated so that multiple broadcast files can be added
brdc_file = BRDC00WRD_R_20192710000_01D_MN.rnx
; File with constraints to apply to the filter. This option is repeatable.
constraint_file = constraints.txt
; Weighting scheme for incoming measurements [constant;sin;sqrtsin;snr] [default:snr]
; - constant: all measurements weight the same
; - sin: weight depends on the sine of the elevation
; - sqrtsin; weight depends on the square root of the sine of elevation
; - snr: weight depends on the SNR of the incoming measurements, regardless the
elevations
data_weight=snr
; list of excluded constellations (separated by a comma). Constellations expressed
; with 3-letter RINEX codes
exclude_constellations=GLO,BDS
; list of excluded satellites (separated by a comma)
exclude_satellites=R13,E12,E30
; minimum elevation [degrees] (default: 5)
min_elevation = 5
; Ignore the error of the measurement provided by the input data and apply the
; default ones (3m for code and 3cm for phase) [yes;no] (default: no)
override_measurement_sigma=yes

; Update the apriori of the receivers on every new fix.
update_apriori_on_fix=yes

[observables]
; observable to process with associated process noise in meters, after
; a comma (repeatable)
; (if not specified, all measurements will be used, taking the
; default measurement noise for the code and phase)
C1C=1.0
L1C=0.03

[ionosphere]
; ionosphere model to compute the pre-fit residual (and/or stec apriori),
model = klobuchar
;model = klobuchar[,/path/to/brdc/file]
;model = ionex,/path/to/ionex/file
;model = constant,10
; if the ionosphere needs to be estimated, specify the stochastics
[static;whitenoise;randomwalk].
; if not specified, the ionosphere will be just modelled (not estimated).
; For random walk, please also specify the process noise
; As a reference, when computing the STEC variation (each second) using Klobuchar
model,
; values of 0.003 TECU per second are obtained, so a value of 0.1 TECU/sqrt(s)
; would cover this variation
stec_stochastics=randomwalk,0.1

[troposphere]
; troposphere model to be used  [niell;saastamoinen]
; - niell model accepts, optionally, the wetz delay and also the East and North
;        gradients (separated by comma)
model = niell,0.1,0.001,0.001
#model = saastamoinen
; Wet tropospheric zenith delay stochastics. If this setting is
; specified, the troposphere will be estimated. If this setting
; is absent, the troposphere will be fixed.
; In order to specify the rate for the RANDOMWALK process, make it
; rate independent in units of meters/sqrt(s). The data rate will
```

```
; be then added when building the process noise matrix. Typical
; values for random walk process noise are 1cm2/h, which amounts,
; in sigma, to 1.67e-4 meters/sqrt
wetz_stochastics = randomwalk,1.67e-4
; Optionally to the wetz, the user can also specify the East and North Gradients
; of the troposphere wet delay
gradient_east_stochastics = randomwalk,1.67e-4
gradient_north_stochastics = randomwalk,1.67e-4

[output]
; File with the filter solution. Format will be defined by the file extension.
; If the solution file is not defined, standard output will be used.
; Supported formats: csv, nmea, gdf, pd
solution=solution.csv
; Force type of solution file and ignore the one defined by the extension
solution_filetype= csv
; File to store the prefit residuals
prefit=prefit.txt
; File to store the postfit residuals
postfit=postfit.txt
; File to store the model data (see argos INI configuration file for further
reference)
model=model.txt
; Output the measurements that are fed to the processing engine and about to
; be assimilated (note that the measurements output will correspond to the ones
; that have been potentially edited)
measurements=measurements.gdf


[satellite]
orbit_file = igs18561.sp3
clock_file = igs19601.clk_30s

[wifi]
wap_position_file = positions.csv
; Multiplier used to scale the WiFi measurement noise [default: 1.0]
noise_scale = 1.0

[receiver: BAS1]
; Input file for this receiver
input = bas1.rnx
position = 4796983.777,160308.749,4187339.9903
; Format of the coordinates  [llh;xyz] (default: xyz)
coordinates_type = xyz
; Sigma (3D) of the a-priori position, in meters
position_3D_sigma = 0.001

[receiver: ROVR]
; Input file for this receiver
input = rovr.rnx
input_format = RINEX_OBS

; Position stochastics, which define which are the stochastics to be
; used for position (static, kinematic, random walk, ...). If no
; stochastics are defined, the position will not be estimated (i.e.
; assumed fixed)
; Other examples of position-stochastics are:
; - White noise stochastics (for e.g. kinematic positioning)
;   "position-stochastics: whitenoise"
; - Random walk stochastics. In this case the process noise has to
;   be estimated
```

```
;   "position-stochastics: RANDOMWALK 0.1"
position_stochastics = whitenoise

; Estimation of the velocity, if present
; The velocity will be estimated if the input data contains Doppler
; measurements
velocity_stochastics = randomwalk,0.1

; Boolean to select wether the receiver position and or clock
; shall be propagated (i.e. deterministic update) using their
; corresponding first derivatives (i.e. velocity and clock drift
; respectively). This setting makes sense only if the velocity
; or clockdrift stochastics are set.
state_propagation = yes

; Stochastics for the clock estimation.
; In general, these should be estimated as whitenoise
clock_stochastics = whitenoise

; Estimation of the clock drift, if present. This will be estimated
; if the Doppler measurements are present.
clock_drift_stochastics = randomwalk,0.1

; Stochastics for the phase clock estimation
; Use this parameter when you need to estimate 2 separate clocks for
; the code and phase measurements. In general this will not be necessary,
; but in certain cases (smartphone data processing, which biases between
; code and phase) this might be required. If this configuration line is
; not used, phase clock will be assumed to be equal to the "regular"
; receiver clock.
phase_clock_stochastics = whitenoise

; Binding section, that is receiver dependent. In this section it
; will be possible to bind certain types of parameters such as the
; troposphere or the ionosphere. This is useful to perform RTK-like
; processing.
; Each binding will be defined with the binding type and the binding
; station, which can be defined as a specific station (will
; raise an error if the station is not listed in the configuration file
;
; Possible binding typs:
; - troposphere
; - ionosphere
;
; Possible entries to specify the binding station:
; - <station_name> will pick the specified station. The program will exit
;                  if the station is not present in the configuration file
;
; The bindings will be only specified to the rover stations. There is
; no need to specify the bindings at the two stations (i.e. in this
; case, the receiver BAS1 does not have to include the bindings to
; the ROVR station because they have been added in ROVR already)
troposphere_binding = nearest
ionosphere_binding = BAS1

; Enable data editing to detect time gaps and flag cycle slips when data is
; absent by a period longer than the one specified (in seconds)
edit_time_gap = 10

; Enable pseudorange and carrier phase boundary checks. Values are separated
; by comma ',' and need to respect the following order: pseudorange min,
```

```
; pseudorange max, carrier phase min and carrier phase max. Values should be
; provided in meters.
edit_bound_check = 18e6,50e6,-40e8,+40e8

; Ignore all measurements under the SNR specified (in dB-Hz)
edit_snr_min = 20

; Threshold for the carrier phase cycle slip detection based on time
; difference of the iono combination (LI). This configuration applies only
; for multi-frequency input data (ignored for single frequency
; data). Value in meters.
edit_li_delta = 1.0

; Cycle slip detection based on median jump. Define the threshold to be applied
; (in meters) and, optionally, the degree of the polynomial for extrapoation
; (if not provided will default to 2). Separate these values by a comma ','
edit_median_jump = 0.5,3

; Force integer epochs at the given sampling rate. Oftentimes the receiver will
; report the epochs with a slight offset due to the receiver clock. For instance,
; the fractional part of the second might be e.g. 0.096, 0.196, 0.296 instead
; of 0.1, 0.2, 0.3 for a sampling rate of 10Hz (0.1s). The first set of seconds
; are "non-integers" while the second set are "integer epochs". If the
; Doppler observable is available, it is recommended to correct the time tags
; and thus the observables to the nearest integer epoch. Note that to do this
; it is necessary to know the sampling rate
edit_itegerize_at_rate = 0.1

; Wi-Fi editing section
; Minimum Received Signal Strength (RSS, in dB)
wifi_edit_min_rss=-90

; Minimum Round Travel Time value (in meters). RTT values lower than this value
; will be removed. Bear in mind that the hardware biases of the access points
; may drive the RTT values below 0!
wifi_edit_min_rtt=0

; Maximum Round Travel Time value (in meters)
wifi_edit_max_rtt=30

; Maximum allowed noise for the Round Travel Time measurements (in meters)
wifi_edit_max_rtt_sigma=2

[ssr_corrections]
input=igs03_20211028104903.rtcm3
; SSR corrections can be provided either with RTCM3 SSR corrections or Galileo HAS.
; If extension of input is .rtcm3, it will autodetect filetype, but otherwise it will
; throw an error and exit if input_type is not specified.
ssr_provider=RTCM3
; Input type can either be a stream of data (for processing in real time - reference
time
; is going to be extracted from the machine time) or a file (where reference time is
going
; to be obtained from the filename, such as in the example). STREAM format is going
to be
; processed as an input_stream (similar to receiver, using the multiplexer). On the
other hand,
; FILE type is going to be processed at start.
input_type=FILE
; For RTCM3 providers, it may be necessary to provide the reference time if using
files. This is
```

```
; because it is intended to work real-time and reference epoch is not provided, only
tow/tod (in
; HAS, this reference is actually provided, so it is not necessary to fill this
parameter). If
; RTCM3 files do not specify reference_epoch, RIFT will try to extract it from its
filename, e.g.
; igs03_20211028104903.rtcm3 = 2021:10:28 10:49:03.000
reference_epoch=2021:10:28 10:49:03.000
```