Sebastien Allegyer, Rupert Brown, John Donovan, Ken Harima, Aaron Hammond, Lavanya Kumarappan, Tao Li, Ronald Maj, Bogdan Matviichuk, Simon McClusky, Michael Moore, Thomas Papanikolaou, Tzupang Tseng, Umma Zannat

Ginan

VERSION 1.0-ALPHA

Geoscience Australia and Frontier-SI

PUBLISHED BY GEOSCIENCE AUSTRALIA AND FRONTIER-SI

TUFTE-LATEX.GOOGLECODE.COM

First printing, July 2021

Contents

Welcome

Ginan is the fifth-brightest star in the Southern Cross (Epsilon Crucis) . It represents a red dilly-bag filled with special songs of knowledge. Indigenous Australians often used songs to convey and to pass on knowledge to others, song were also often used as a way to navigate the country.

We hope that you find this software tool kit will convey our understanding on how to process GNSS signals and will also help you to navigate the country!

THE STORY Ginan was found by Mulugurnden (the crayfish), who brought the red flying foxes from the underworld to the sky. The bats flew up the track of the Milky Way and traded the spiritual song to Guyaru, the Night Owl (the star Sirius). The bats fly through the constellation Scorpius on their way to the Southern Cross, trading songs as they go.

The song informs the people about initiation, which is managed by the stars in Scorpius and related to Larawag (who ensures the appropriate personnel are present for the final stages of the ceremony).

The brownish-red colour of the dilly bag is represented by the colour of Epsilon Crucis, which is an orange giant that lies 228 light years away.

This manual is divided into three major sections: the preliminary matter which gives a quick overview on how to install and use the software with examples. The next part contains the background theory on the models that have been implemented into Ginan. The back matter contains information about configuration files, and file formats used by the software. This manual is far from complete, and we will be adding to it as we continue our development work on Ginan. Our intention is to provide examples on how to use the tool kit , and then to provide the background theory as we can.

List of Figures

List of Tables

Introduction to Ginan GNSS Processing Toolkit

Ginan is a collection of source code that is currently made up two distinct software repositories, the POD [1] and the PEA[2]. Using the POD and PEA together will allow you to estimate your own satellite orbits from a global tracking network.

[1] https://bitbucket.org/geoscienceaustralia/pod/wiki/Home

[2] https://bitbucket.org/geoscienceaustralia/pea/wiki/Home

The POD (precise orbit determination) contains all of the source code needed to determine a GNSS satellite's orbit. You can establish the initial conditions of an orbit from a broadcast ephemeris file, or from an IGS SP3 file. It can then estimate it's own orbital trajectory based upon the models specified in configuration files, and output an SP3 file, or provide a partial files which can then be updated from a tracking network.

The PEA (parameter estimation algorithm) takes raw observations in RINEX format or in RTCM format, to estimates the parameters you are interested in. You can run it a single user mode, taking in orbit and clocks supplied by real-time streams to SP3 files obtained from the IGS to estimate your own position in static and kinematic mode. You can also run the PEA in a network mode, and take in a global network of observations to determine your own orbits and satellite clocks to support your application.

The software is aimed at supporting Australia's implementation of a national positioning infrastructure that supports the objective of 'instantaneous GNSS positoning anywhere, anytime, with the highest possible accuracy and the highest possible integrity.

Carrier phase ambiguity. The underlying signals transmitted by the Global Navigation Satellite Systems (GNSS) can be considered as waves, just like repeating sine waves from high school mathematics. Measurements of these waves are referred to as carrier phase observations, and they are used to provide the precise distance, with mm precision and accuracy, between the orbiting satellites and user's receiver that are subsequently used to compute position. However, a complicating factor is that carrier phase observations have an ambiguous component where the total whole number of waves, or integer cycles, between the satellite and the user's receiver cannot be measured, only the fractional part. The unknown number of integer cycles is called the carrier phase ambiguity. Fortunately, the ambiguities can be estimated, and the mathematical and statistical solution to this problem is known as integer ambiguity estimation. While there is a long history of research in this area, which has largely focused on GPS applications, the most optimal solution to this problem when simultaneously combining data from all the GNSS remains unresolved.

Atmosphere delay of GNSS signals. The Earth is surrounded by layers of gases held by Earth's gravity. Signals, such as those transmitted by GNSS, propagated from space are delayed as they pass through the atmosphere. In the troposphere, the region from the Earth's surface to approximately 20 km altitude, the delay is proportional to temperature, pressure and humidity. The ionosphere, the region from 50-1000 km altitude, causes delay as a function of the frequency of the signal. The composition of both the troposphere and ionosphere vary both in space and time, and this variability currently limits the accuracy, speed and reliability of positioning. But it's not all bad news, and like a CAT scan in medical science, the new GNSS signals and satellites can potentially be combined to provide a more complete three dimensional picture of the atmospheric delay as a function of time. Models that more completely remove the nuisance atmospheric signals will lead to improved accuracy, speed and reliability of positioning.

Precise Point Positioning (PPP) and Real Time Kinematic (RTK). Conventional positioning technologies almost exclusively use a technique called Real Time Kinematic (RTK). The RTK technique takes information from nearby Continuously Operating Reference Stations (CORS) to generate corrections for measurements made by users. One of the most important of these is the carrier phase ambiguity correction. The ability to correctly determine carrier phase ambiguities with RTK is determined by many factors, such as the distance between the reference stations and the user and also atmospheric effects. Consequently, RTK relies on relatively dense CORS networks with a typical spacing of 30 to 70 km. An alternative to RTK is Precise Point Positioning (PPP). The PPP technique, rather than directly using measurements from nearby reference stations, uses global satellite orbit and clock information such as that provided by the International GNSS Service. The major advantage of PPP is that it doesn't require a dense CORS network nearby the user's location just access to global products. Unfortunately, the PPP technique can have difficulties in resolving carrier phase ambiguities in real time, but additional research focused on greater exploitation of multi-GNSS data and more regional approaches may in the future overcome this limitation.

Installation

To Install

In this section we will describe how to install the PEA and POD from source. An alternative option to installing all of the dependencies and the source code would be to use one of our docker images available from Docker Hub. Instructions on how to do this are in (see Docker).

PEA

Dependencies the following packages need to be installed with the minimum versions as shown below. This guide will outline the preferred method of installation.

CMAKE > 3.0 requires openssl-devel to be installed (requires openssl-devel) YAML > 0.6 Boost > 1.70 gcc > 4.1 Eigen3 Build To build the PEA Precise Estimation Algorithm...

We suggest using the following directory structure when installing the Ginan toolkit. It will be created by following this guide.

The following is an example procedure to install the dependencies necessary to run the pea on a base ubuntu linux distribution

Update the base operating system:

```
$ sudo apt update
$ sudo apt upgrade
```

Install base utilities gcc, gfortran, git, openssl, blas, lapack, etc

```
$ sudo apt install -y git gobjc gobjc++ gfortran libopenblas-dev openssl
    curl net-tools openssh-server cmake make \
liblapack-dev gzip vim libssl1.0-dev python3-cartopy python3-scipy python3-
    matplotlib python3-mpltoolkits.basemap
Create a temporary directory structure to make the dependencies in:
$ sudo mkdir -p /data/tmp
$ cd /data/tmp
```

YAML We are using the YAML library to parse the configuration files used to run many of the programs found in this library (https://github.com/jbeder/yaml-cpp). Here is an example of how we have installed the yaml library from source:

```
$ cd /data/tmp
$ sudo git clone https://github.com/jbeder/yaml-cpp.git
$ cd yaml-cpp
$ sudo mkdir cmake-build
$ cd cmake-build
$ sudo cmake .. -DCMAKE\_INSTALL\_PREFIX=/usr/local/ -DYAML\_CPP\_BUILD\_TESTS=OFF
$ sudo make install yaml-cpp
$ cd ../..
$ sudo rm -fr yaml-cpp
```

Boost We rely on a number of the utilities provided by boost (https://www.boost.org/), such as their time and logging libraries.

```
$ cd /data/tmp/
$ sudo wget -c https://dl.bintray.com/boostorg/release/1.73.0/source/boost_1_73_0.tar.gz
$ sudo gunzip boost_1_73_0.tar.gz
$ sudo tar xvf boost_1_73_0.tar
$ cd boost_1_73_0/
$ sudo ./bootstrap.sh
$ sudo ./b2 install
$ cd ..
$ sudo rm -fr boost_1_73_0/ boost_1_73_0.tar
```

Eigen3 is used for performing matrix calculations, and has a very nice API.

```
$ cd /data/tmp/
$ sudo git clone https://gitlab.com/libeigen/eigen.git
$ cd eigen
$ sudo mkdir cmake-build
$ cd cmake-build
$ sudo cmake ..
$ sudo make install
$ cd ../..
$ sudo rm -rf eigen
Installing PEA
PEA Executable
$ cd /data/acs/
```

Clone the repository via https:

```
$ git clone https://bitbucket.org/geoscienceaustralia/pea.git
```

You should now have...

Prepare a directory to build in, its better practise to keep this separated from the source code.

```
$ cd pea/cpp
$ mkdir -p build
$ cd build
```

Run cmake to find the build dependencies and create the makefile. You have the choice of adding in a couple of compile options. Using the flag -DENABLE_MONGODB=TRUE will set up the mongodb utilities, adding the flag -DENABLE_OPTIMISATION=TRUE will set up the compiler to run optimisation O3. Enabling the optimisation flag will speed up the processing by a factor of 3, however this can lead to compile errors depending on the system you are compiling on, if this happens remove this option.

```
$ cmake ..
or to enable MONGODB utilities
$ cmake -DENABLE_MONGODB=TRUE ..
and to enable Optimisation
$ cmake -DENABLE_MONGODB=TRUE -DENABLE_OPTIMISATION=TRUE ..
```

Now build the pea

```
$ cmake --build $PWD --target pea
```

To change to build location substitute your preferred destination for $PWD , e.g /usr/local/bin

Alternatively to the command above you can make the code in parallel using:

```
$ make -j 5 all
```

where the -j flag controls how many jobs can be run at the same time.

Check to see if you can execute the pea:

```
$ ./pea
```

and you should see something similar to:

```
PEA starting...
Options:
  --help                       Help
  --verbose                    More output
  --quiet                      Less output
  --config arg                 Configuration file
  --trace_level arg            Trace level
  --antenna arg                ANTEX file
  --navigation arg             Navigation file
  --sinex arg                  SINEX file
  --sp3file arg                Orbit (SP3) file
  --clkfile arg                Clock (CLK) file
  --dcbfile arg                Code Bias (DCB) file
  --ionfile arg                Ionosphere (IONEX) file
  --podfile arg                Orbits (POD) file
  --blqfile arg                BLQ (Ocean loading) file
  --erpfile arg                ERP file
  --elevation_mask arg         Elevation Mask
  --max_epochs arg             Maximum Epochs
  --epoch_interval arg         Epoch Interval
  --rnx arg                    RINEX station file
  --root_input_dir arg         Directory containg the input data
  --root_output_directory arg  Output directory
  --start_epoch arg            Start date/time
  --end_epoch arg              Stop date/time
  --dump-config-only           Dump the configuration and exit
PEA finished
```

The documentation for the pea can be generated similarly using doxygen if it is installed.

```
$ sudo apt-get install doxygen
$ cd pea/cpp/build
$ make doc_doxygen
```

The docs can then be found at doc_doxygen/html/index.html

POD from source

Dependencies

The open basic linear algebra library (Openblas.x86_64,liblas-libs.x86_64) (You may need to run the command ln -s /usr/lib64/libopenblas.so.3 /usr/lib64/libopenblas.so) A working C compiler (gcc will do), a working C++ compiler (gcc-g++ will do) and a fortran compiler (we have used gfortran) Cmake (from cmake.org) at least version 2.8 If the flags set in CMakeLists.txt do not work with your compiler please remove/replace the ones that don't

Build To build the POD ...

```
$ cd pod
$ mkdir build
$ cd build
$ cmake3 ..
$ make >make.out 2>make.err
$ less make.err (to verify everything was built correctly)
```

You should now have the executables in the bin directory: pod crs2trs brdc2ecef

Test To test your build of the POD ... - You may not need the ulimit command but we found it necessary

```
$ cd ../pod/test
$ ulimit -s unlimited
$ ./sh_test_pod
```

At the completion of the test run, the sh_test_pod script will return any differences to the standard test resuts

Configuration File The POD Precise Orbit Determination (./bin/-pod) uses the configuration file:

Docker

Docker is "an open platform for developing, shipping, and running applications". It is a convenient way for us to provide all of the dependencies and the latest release source code so that we can use the ginan tool kit straight out of the box. In order for this to work, we will first need to install the docker engine onto our local machine. If we are running a different operating system instructions on how to install docker can be found at docker desktop downlod link, these also include alternative methods of installing on ubuntu and has links to recommended best practices. To find more information on docker have a look at the getting started guide provided by docker.

Ubuntu Docker dependency installation guide

If we are running ubuntu, we can install a docker engine. A summary of the commands to download and install docker involve setting up the ubuntu repository system to link with the docker repsotory are given below.

```
$ sudo apt -y update
$ sudo apt -y install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Add the dockers official GPG key:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
     dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
$ echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
     https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" |
     sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Then update the repository management system and install the packages.

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Verify that the Docker engine is installed correctly by running the *hello-world* image.

```
$ docker run hello-world
```

Then we will need to change the ownership:

```
$ sudo usermod -aG docker root
$ sudo usermod -aG docker ${USER}
```

You will need to log out and back in for the permissions to take effect.

Using Docker

Once we have docker installed on our local machine we will need to download our image for ginarn:

```
$ docker pull gamichaelmoore/ginarn-base-ubuntu-20.04
```

Then we can run the image as follows:

```
$ docker run -it -v /data:/data gamichaelmoore/ginarn-base-ubuntu-20.04 bash
```

This gives us a run-time environment where the dependencies of ginan are installed. Here, the `-v` option mounts a volume inside the docker instance at `/data`, which maps to the `/data` folder of the host machine. This way this folder can be shared between the host and the container, and the results can persist.

You should now see a `bash` prompt running inside the docker container.

Building ginan

Note

The instructions here are for separate `pea` and `pod` git repositories. The docker image for the consolidated repository is still a work-in-progress

To clone the latest version from the git repositories, run the following from inside the container:

```
mkdir -p /data/acs
pushd /data/acs
# clone pea and pod
# use your own credentials to clone pea
git clone https://git@bitbucket.org/geoscienceaustralia/pea.git
git clone https://anonymous:pipelines@bitbucket.org/geoscienceaustralia/pod.
    git
popd
```

Now, with all the dependencies set up already, we can build the executables:

```
# building pea and pod
cd /data/acs/pea/cpp
mkdir build
cd build
cmake ..
cmake --build /data/acs/pea/cpp/build --target pea
cd /data/acs/
cd pod
mkdir build && cd build
cmake ..
make
export PATH=$PATH:/data/acs/pod/bin
```

This docker image comes with a pre-built conda environment that enables plotting. To use it:

```
conda activate gn37
```

Keeping a container running

If we instantiate a container this way, our session will finish when we quit the bash prompt. The changes we make to the container will also be lost, except the changes that persist outside of the container, that is, the `/data` folder in our example.

Therefore, it is sometimes useful to keep a container running, and connect to it and disconnect from it as needed.

To start up a docker container in the disconnected mode, run:

```
docker run -d -v /data:/data gamichaelmoore/ginarn-base-ubuntu-20.04 sleep
    infinity
```

we can verify that the container is running in the background:

```
docker ps
```

This will show a container ID. Docker conveniently also provides an alias as a “name“. We can start a new bash shell inside the container by:

```
docker exec -it <name> bash
```

where `<name>` is the name or ID of the running docker container.

Python

Python Installation for Plotting, Processing, etc.

Lastly, to run many of the included scripts for fast parsing of .trace/.snx files, plotting of results, automatic running of the PEA based on input date/times and stations, etc. then a number of python dependencies are needed.

The file `python/ginan_py37.yaml` has a list of the necessary python dependencies.

The best way to take advantage of this is to install the Miniconda virtual environment manager.

This will allow you to pass the `.yaml` file into the `conda` command and automatically set up a new python environment.

To install miniconda, enter the following commands:

```
$ wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ bash Miniconda3-latest-Linux-x86_64.sh
```

Listing 1: Installing conda

And follow the installation instructions on screen (choosing all defaults is fine).

Create virtual environment, after installation you can create the *ginan37* python environment. First open a new terminal session and enter:

```
$ conda env create -f <dir_to_pea>/python/ginan_py37.yaml
```

Listing 2: Example showing how to create a conda environment

You have now created the virtual python environment *ginan37* with all necessary dependencies. Anytime you wish you run python scripts, ensure you are in the virtual environment by activating:

```
$ conda activate gin37
```

Listing 3: Activating a conda environment

And then run your desired script.

Examples to run the PEA from a python script

The following examples will show how to use the python scripts in the `source` directory to automatically gather files and run the pea to produce results (as .TRACE files and plots).

Run the PEA

The easiest way to run the is to choose a reference station, a start time, end time and directories to store the gathered files and resultant outputs.

Doing this for station *HOB200AUS* and processing 2 days of data, starting 20 Dec 2020 00:00 and ending 21 Dec 2020 23:59:30, we can run the following code (assuming the `pwd` is your pea directory):

```
python3 python/source/run_pea_PPP.py HOB200AUS 2020-12-20_00:00:00
2020-12-21_23:59:30 <directory_to_download_files_to> <results_output_directory>
-md -del_yaml
```

This will produce a .TRACE file that contains XYZ position information for that station and estaimes for the Zenith Total Delay (ZTD). The flags are as follows:

- `-md` is used to allow dates to be input in *YYYY-MM-DD* format as above. If the flag is not selected, the format must be *YYYY-DOY* (where *DOY* is day-of-year).

- `-del_yaml` is used to delete the `.yaml` config file after the pea run. Without this flag, the `.yaml` file will be kept (in `pwd`)

Near-Real-Time (NRT) Daily Run

In this example, we will see how to run the pea to produce ZTD and station coordinate results for reference station *HOB200AUS*, using the most rececnt data available on GA and CDDIS servers.

Assuming the pwd is your `pea` directory, then the following code should run to produce a .TRACE file and plots of ZTD and XYZ station coordinates for the most recent day:

```
python3 python/source/run_pea_PPP.py HOB200AUS _ _ <directory_to_download_files_to>
<results_output_directory> -rapid -m_r -plt_m_r
```

This differs to the code above in that the flags `-rapid`, `-m_r` and `plt_m_r` have been selected.

- `-rapid` will find the rapid versions of files - these are less accurate than final versions but are available sooner at CDDIS

- `-m_r` will get the code to search the CDDIS database and find the most recent rapid files available for `clk` and `sp3` files. The date and time is then selected based on this, allowing us to _ _ for the start and end times

`--plt_m_r` will plot the ZTD and XYZ results as `.png` files and place in a `plots` directory

Site metadata utility

Proper geodetic processing requires knowing of the site's receiver and antenna information. PEA extracts this information from the input sinex file or a set of sinex files. In addition, user can download igs.snx file that contains historical records of ∼500 sites and is regenerated daily from the logfiles available at `ftp://ftp.igs.org/pub/station/log/`. We developed a similar tool, log2snx.py, that does parsing of available igs logfiles and produces a sinex file as needed by PEA.

```
$ python3 scripts/log2snx.py  -h
  usage: log2snx.py [-h] [-l LOGGLOB] [-r RNXGLOB [RNXGLOB ...]] [-o OUTFILE
     ]
  [-fs FRAME_SNX] [-fd FRAME_DIS] [-fp FRAME_PSD]
  [-d DATETIME] [-n NUM_THREADS]

IGS log files parsing utility. Globs over log files using LOGGLOB expression
and outputs SINEX metadata file. If provided with frame and frame
discontinuity files (soln), will project the selected stations present in
     the
frame to the datetime specified. How to get the logfiles: rclone sync
igs:pub/sitelogs/ /data/station_logs/station_logs_IGS -vv How to get the
     frame
files: rclone sync itrf:pub/itrf/itrf2014 /data/ITRF/itrf2014/ -vv --include
"*{gnss,IGS-TRF}*" --transfers=10 rclone sync igs:pub/ /data/TRF/ -vv
--include "{IGS14,IGb14,IGb08,IGS08}/*" see rclone config options inside
     this
script file Alternatively, use s3 bucket link to download all the files
     needed
s3://peanpod/aux/

optional arguments:
-h, --help            show this help message and exit
-l LOGGLOB, --logglob LOGGLOB
        logs glob path (required)
-r RNXGLOB [RNXGLOB ...], --rnxglob RNXGLOB [RNXGLOB ...]
        rinex glob path (optional)
-o OUTFILE, --outfile OUTFILE
        output file path (optional)
-fs FRAME_SNX, --frame_snx FRAME_SNX
        frame sinex file path (optional)
-fd FRAME_DIS, --frame_dis FRAME_DIS
        frame discontinuities file path (required with
        --frame_snx)
-fp FRAME_PSD, --frame_psd FRAME_PSD
        frame psd file path (optional)
-d DATETIME, --datetime DATETIME
        date to which project frame coordinates, default is
        today (optional)
-n NUM_THREADS, --num_threads NUM_THREADS
        number of threads to run in parallel
```

Listing 4: log2snx.py help message

Sinex preview utility

To quickly visualise and compare networks, a fast sinex preview utility has been developed - snx2map.py. It accepts any number of sinex files, extract estimated coordinates of the sites present and outputs a map in the form of html page.

```
$ python3 scripts/snx2map.py  -h
usage: snx2map.py [-h] [-i SNXPATH [SNXPATH ...]] [-o OUTDIR]

Parse sinex SITE/ID block and create html map.

optional arguments:
  -h, --help            show this help message and exit
  -i SNXPATH [SNXPATH ...], --snxpath SNXPATH [SNXPATH ...]
                        path to sinex file (.snx/.ssc). Can be compressed
    with
                        LZW (.Z)
  -o OUTDIR, --outdir OUTDIR
                        path to output dir
```

Listing 5: snx2map.py help message

Latency Tool

To run currently do something like...

```
nohup python source/main/python/npi/latency.py 2>&1 &
```

A simple configuration file `config/config.yaml` looks like...

```
streams:
- type: NTRIP
  protocol: http
  host: auscors.ga.gov.au
  port: 2101
  username: a_username
  password: a_password
  count: 10
  stations:
  - {format: RTCM3, id: COCO7}
```

and again with more streams looks something like...

```
streams:
- type: NTRIP
  protocol: http
  host: auscors.ga.gov.au
  port: 2101
```

```
  username: a_username
  password: a_password
#  count: 10
  stations:
  - {id: TID17}
  - {id: 00NA7}
  - {id: 01NA7}
  - {id: ABMF7}
```


Messages

- 1033 Received and Antenna Description
- 1077 GPS MSMs
- 1087 GLONASS MSMs
- 1097 Galileo MSMs
- 1117 QZSS MSMs
- 1127 BDS MSMs

MSM (1,2,3,4,5,6,7)

MSM message contains - message header - satellite data - signal data

The message header contains - message number DF002 uint12 12-bits - reference station id DF003 uint12 12-bits - GNSS epoch time ??? uint30 30-bits - multiple message bit DF393 bit(1) 1-bit

GNSS Epoch Time - GPS is –TOW DF004 DF004 GPS Epoch Time (TOW) 0-604,799,999ms 1ms uint30 milliseconds from beginning of GPS week (midnight GMT on Saturday night/Sunday morning measured in GPS time) - GLONASS is – GLONASS DAY OF WEEK DF416 int3 3-bits DF416 GLONASS Day of Week 0-7 1 uint3 0=sunday, 1=monday, . . . , 6=saturday, 7=not known – GLONASS EPOCH TIME DF034 uint27 27-bits DF034 GLONASS Epoch Time 0-86,400,999ms 1ms uint27

- Galileo is – TOW DF248 DF248 GALILEO Epoch Time 0-604,799,999ms 1ms uint30 - QZSS is – TOW DF428 DF428 QZSS Epoch Time 0-604,799,999ms 1ms uint30

- BeiDou is – TOW DF+002 DF427 BeiDou Epoch Time 0-604,799,999ms 1ms uint30

RTCM v3

The following is a Hex-ASCII example of a message type 1005 (Stationary Antenna Reference Point, No Height Information).

D3 00 13 3E D7 D3 02 02 98 0E DE EF 34 B4 BD 62 AC 09 41 98 6F 33 36 0B 98

The parameters for this message are: Reference Station Id = 2003 GPS Service supported, but not GLONASS or Galileo ARP ECEF-X = 1114104.5999 meters ARP ECEF-Y = -4850729.7108 meters ARP ECEF-Z = 3975521.4643 meters

ACS Metrics

RTCM v3

An `RTCM v3` stream of data consists of a set of `RTCM v3` packets.

An `RTCM v3` packet consists of:

- 8-bit preamble
- 16-bits ** 6-bit reserved (should be `0`s) ** 10-bit length
- variable length message ** 12-bit message type
- 24-bit message CRC

x	x	x
Preamble	8-bits	

- 8-bit pre-amble character
- The start or an `RTCM v3` packet is marked by the `0xD3` pre-amble character

  Following this is

Reading RTCM3 Data…

- RTCM3 packet starts with 0xd3 byte so first skip to start of packet

  See [https://pdfs.semanticscholar.org/bf93/21e569cc0f009982af3158a6489accc8f3e5.pdf]

  RTCM v3 frame looks like: - preamble 8-bits (0xD3) - reserved 6-bits - message length 10-bits - message variable length - CRC 24-bits

  So to parse it we do 1. find the start of packet byte - i.e. the preamble byte - i.e. 0xd3

2. read the next 2 bytes (aka 16-bits) and extract the message length from the right-hand 10-bits

RTCM v3

d3 74 63

```
               09   8765 4321
```

1101 0011 | 0111 0100 | 0110 0011 |
so the length is 63 (hex which is 99 decimal)
d3 74 63 18 d1 94 65 0d 43 4c 53 14 b1 2c 4f
1101 0011 | 0111 0100 | 0110 0011 | 1101 0011 | 0111 0100 | 0110 0011
00011000 00011000
byte 0 = preamble = 0xD3
bytes 1 & 2 = reserved (6-bits) + message length (10-bits)

```
byte1 & 0x03 << 8 | byte2

byte1 = 00011000
byte2 = 00011000

00011000 & 0x03 = 00000000
```

bytes 3. . . n-1 = variable length message
bytes n. . . n+2 = 24-bit CRC (CRC24)

RTCM v3 Message

reserved : 1-bit message number : 12-bits uint12
18 d1 is. . .
123456789012 0001100011010001

RTCM v3

GPS RTK Observable Messages

message header (aka number aka type): - 1001 - 1002 - 1003 - 1004

DATA FIELD	DF NUMBER	DATA TYPE	NO. OF BITS
Message Number (e.g.,“1001”= 0011 1110 1001)	DF002	uint12	12
Reference Station ID	DF003	uint12	12
GPS Epoch Time (TOW)	DF004	uint30	30
Synchronous GNSS Flag	DF005	bit(1)	1
No. of GPS Satellite Signals Processed	DF006	uint5	5

DATA FIELD	DF NUMBER	DATA TYPE	NO. OF BITS
GPS Divergence-free Smoothing Indicator	DF007	bit(1)	1
GPS Smoothing Interval	DF008	bit(3)	3
TOTAL			64

Station Antenna Reference Point Messages

- 1005
- 1006

1005

DATA FIELD	DF NUMBER	DATA TYPE	NO. OF BITS
Message Number (e.g. "1005"= 0011 1110 1101)	DF002	uint12	12
Reference Station ID	DF003	uint12	12
...			
TOTAL			152

1006

DATA FIELD	DF NUMBER	DATA TYPE	NO. OF BITS
Message Number (e.g. "1006"= 0011 1110 1110)	DF002	uint12	12
Reference Station ID	DF003	uint12	12
...			
TOTAL			168

Antenna Description Messages

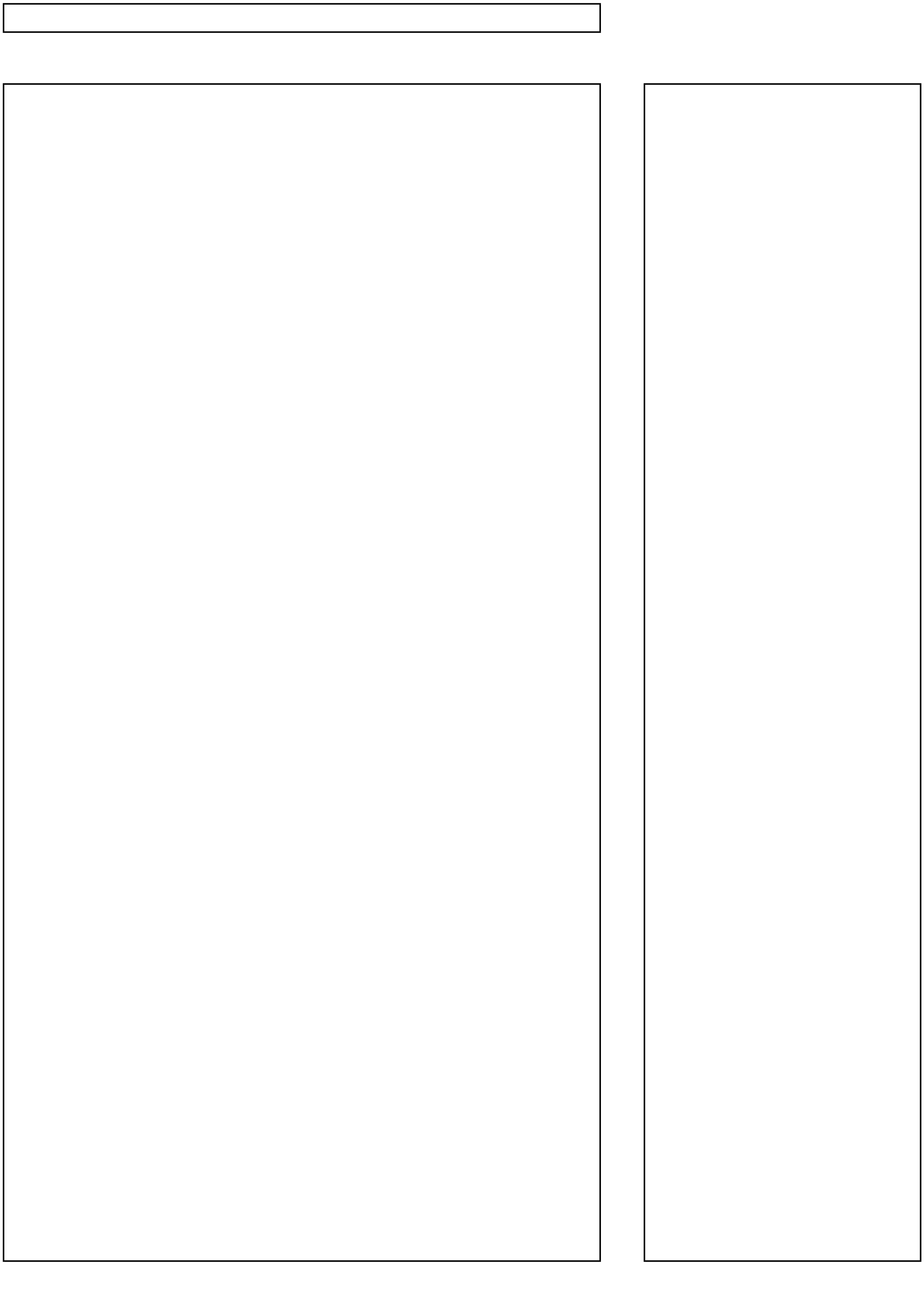
- 1007
- 1008

1007

DATA FIELD	DF NUMBER	DATA TYPE	NO. OF BITS
Message Number (e.g. "1007"= 0011 1110 1111)	DF002	uint12	12
Reference Station ID	DF003	uint12	12
...			
TOTAL			40+8*N

The Python RTCM3 library knows about the following messages: - 1001 - 1002 - 1003 - 1004 - 1005 - 1006 - 1008 - 1009 - 1010 - 1011 - 1012

- 1033



POD Examples

Processing Example 1

In this example the pod will perform a dynamic orbit determination for PRN04 over a 6 hour arc. The full gravitational force models are applied, with a cannonball model SRP model.

To run the POD ...

```
$ bin/pod
```

This should output the following to stdout...

```
Orbit Determination
Orbit residuals in ICRF : RMS(XYZ)   1.6754034501980351E-002   5.2908718335411935E-002   1.567611559903477
Orbit Determination: Completed
CPU Time (sec)   298.48134399999998
External Orbit comparison
Orbit comparison: ICRF
RMS RTN   2.8094479714173427E-002   2.4358145601708528E-002   4.4097979280889953E-002
RMS XYZ   1.6754034501980351E-002   5.2908718335411935E-002   1.5676115599034774E-002
Orbit comparison: ITRF
RMS XYZ   3.9069978513805753E-002   3.9343671258381237E-002   1.5660654272651970E-002
Write orbit matrices to output files
CPU Time (sec)   349.19307899999995
The results above show that our orbits arcs, over 6 hours, are currently within 2-5 cm of the final combin
```

The processing also produces the following output files... /beginverbatim /endverbatim

Processing Example 2 - ECOM2 SRP

In this example we will change the SRP model to use the ECOM2 model.

Edit the EQM.in file so that the Solar Radiation Pressure configuration section now looks:

! Solar Radiation Pressure model: ! 1. Cannonball model ! 2. Boxwing model ! 3. ECOM (D2B1) model SRP_model 3

Then edit VEQ.in, so that the Non-gravitational forces now looks like:

! Solar Radiation Pressure model: ! 1. Cannonball model ! 2. Box-wing model ! 3. ECOM (D2B1) model SRP_model 3

run the POD ...

```
$ bin/pod
```

This should output the following to stdout...

```
Orbit Determination
Orbit residuals in ICRF : RMS(XYZ)   2.0336204859568077E-002   8.4715644601919167E-003   3.968793232271467
Orbit Determination: Completed
CPU Time (sec)   299.68054799999999
External Orbit comparison
Orbit comparison: ICRF
RMS RTN   2.8182836396022540E-002   2.4598832384842121E-002   2.5879201921952168E-002
RMS XYZ   2.0336204859568077E-002   8.4715644601919167E-003   3.9687932322714677E-002
Orbit comparison: ITRF
RMS XYZ   1.8757217704973204E-002   1.1635302426688266E-002   3.9702619816620370E-002
Write orbit matrices to output files
CPU Time (sec)   350.88653299999999\
```

Example 3 - (pod/examples/ex3)

GPS IGS SP3 file orbit fitting, orbit prediction and comparison to next IGS SP3 file

Example 4 - (pod/examples/ex4):

Integration of POD initial conditions file generated by the PEA

Example 5 - (pod/examples/ex5):

ECOM1+ECOM2 hybrid SRP model In each example directory (ex1/ex2/ex3/ex4) there is a sh_ex? script that when executed will run the example and compare the output with the expected solution.

The POD is designed to do some stuff.

Processing in Precise Point Positioning mode

Precise Point Positioning(PPP)

Pea PPP Processing examples

In this example we will process 24 hours of data from a permanent reference frame station. The algorithm that will be use an L1+L2 and L1+L5 ionosphere free combination. The log files and processing results can be found in /data/acs/pea/output/exs/EX01_IF/.

```
$ ./pea --config ../../config/Ex01-IF-PPP.yaml
$ grep "\$POS" /data/acs/pea/output/exs/EX01_IF/EX01_IF-ALIC.TRACE
```

And you should see the following:

```
<snip> $POS,2062,431940.000,0,-4052052.7956,4212836.0144,-2545104.6423,0.00000043966020,0.00000039738502,0
```

Using the Ionosphere-free observable to process a static data set - the float solution

In this example we will process 24 hours of data from a permanent reference frame station. The algorithm will use an L1+L2 and L1+L5 ionosphere-free combination. The log files and processing results can be found in '<path to pea>/output/exs/EX01_IF/'.

```
$ ./pea --config ../../config/EX01-IF-PPP.yaml
```

The pea will then have the following output in <path to pea>/output/exs/EX01_IF/ :

EX01_IF20624.snx - contains the station position estimates in SINEX format EX01_IF-ALIC2019199900.TRACE - contains the logging information from the processing run

```
$ grep "REC_POS" /data/acs/pea/output/exs/EX01_IF/EX01_IF-ALIC201919900.TRACE > ALIC_201919900.PPP
```

This will pipe all of the receiver position results reported in the station trace file to a seperate file for plotting.

```
$ python3 /data/acs/pea/python/source/pppPlot.py --ppp /data/acs/pea/output/exs/EX01_IF/ALIC_201919900.PPP
```

This will then create the plots alic_pos.png, a time series of the difference between the estimated receiver position and the median estimated position. And the plot alic_snx_pos.png, a time series of the difference between the estimated receiver position and the IGS SINEX solution for Alic Springs on this day.

Single Frequency Processing

```
$ ./pea --config ../../config/Ex01-SF-PPP.yaml
```

```
$ grep "\$POS" /data/acs/pea/output/exs/EX01_SF/EX01_SF-ALBY202011500.TRACE
```

And you should see something similar to the following:

```
<snip>
$POS,2062,431940.000,0,-4052052.7956,4212836.0144,-2545104.6423,0.00000043966020,0.00000039738502,0.000000
$POS,2062,431970.000,0,-4052052.7956,4212836.0144,-2545104.6423,0.00000043965772,0.00000039738393,0.000000
```

Processing realtime

To process a continuous GPS station in real-time you will need to access the data stream from a NTRIP stream and the correction products from a NTRIPCaster. Geoscience Australia is running a caster that provides global data stream, a dense network of stream covering the Australian region, and correction procdust provide by IGS Analysis Centres. You will need to apply for an AUSCORS account, or use the new NTRIPCaster that streams using https.

You can apply for an account at the following link : New GA Account link

Once your have your account and password you can test that you can successfuly connect to the NTRIPCaster by using the following curl command:

```
$ curl https://ntrip.data.gnss.ga.gov.au/MOBS00AUS0 --http0.9 -i -u'<username>:<password> --output -'
```

Coding Standards

Coding Standards for C++

Code style

Overall we are aiming for

- Write for clarity
- Write for clarity
- Use short, descriptive variable names
- Use aliases to reduce clutter.

Bad

```
//check first letter of satellite type against stomething

if (obs.Sat.id().c_str()[0]) == 'G')
    doSomething();
else if (obs.Sat.id().c_str()[0]) == 'R')
    doSomething();
else if (obs.Sat.id().c_str()[0]) == 'E')
    doSomething();
else if (obs.Sat.id().c_str()[0]) == 'I')
    doSomething();
```

Good

```
char& sysChar = obs.Sat.id().c_str()[0];

switch (sysChar)
{
    case 'G':   doSomething();   break;
    case 'R':   doSomething();   break;
    case 'E':   doSomething();   break;
    case 'I':   doSomething();   break;
}
```

Spacing, Indentation, and layout

- Use tabs, with tab spacing set to 4.

- Use space or tabs before and after any

```
+ - * / = < > == != % etc.
```

- Use space, tab or new line after any , ;
- Use a new line after if statements.
- Use tabs to keep things tidy - If the same function is called multiple times with different parameters, the parameters should line up.

Bad

```
trySetFromYaml(mongo_metadata,output_files,{"mongo_metadata" });
trySetFromYaml(mongo_output_measurements,output_files,{"mongo_output_measurements" });
trySetFromYaml(mongo_states,output_files,{"mongo_states" });
```

Good

```
trySetFromYaml(mongo_metadata,             output_files, {"mongo_metadata"              });
trySetFromYaml(mongo_output_measurements,  output_files, {"mongo_output_measurements" });
trySetFromYaml(mongo_states,               output_files, {"mongo_states"                });
```

Statements

* One statement per line - unless you have a very good reason

Bad

```
z[k]=ROUND(zb[k]); y=zb[k]-z[k]; step[k]=SGN(y);
```

Good

```
z[k]    = ROUND(zb[k]);
y       = zb[k]-z[k];
step[k] = SGN(y);
```

Example of a good reason:

* Multiple statements per line sometimes shows repetetive code more clearly, but put some spaces so the separation is clear.

Normal

```
    switch (sysChar)
    {
        case ' ':
        case 'G':
            *sys = E_Sys::GPS;
            *tsys = TSYS_GPS;
            break;
        case 'R':
            *sys = E_Sys::GLO;
            *tsys = TSYS_UTC;
            break;
        case 'E':
            *sys = E_Sys::GAL;
            *tsys = TSYS_GAL;
            break;
        //...continues
    }
```

Ok

```
    if      (sys == SYS_GLO)    fact = EFACT_GLO;
  else if (sys == SYS_CMP)    fact = EFACT_CMP;
  else if (sys == SYS_GAL)    fact = EFACT_GAL;
  else if (sys == SYS_SBS)    fact = EFACT_SBS;
  else                        fact = EFACT_GPS;
```

Ok

```
    switch (sysChar)
    {
        case ' ':
        case 'G':   *sys = E_Sys::GPS;      *tsys = TSYS_GPS;    break;
        case 'R':   *sys = E_Sys::GLO;      *tsys = TSYS_UTC;    break;
        case 'E':   *sys = E_Sys::GAL;      *tsys = TSYS_GAL;    break;
        case 'S':   *sys = E_Sys::SBS;      *tsys = TSYS_GPS;    break;
        case 'J':   *sys = E_Sys::QZS;      *tsys = TSYS_QZS;    break;
    //...continues
    }
```

Braces

New line for braces.

```
    if (pass)
    {
        doSomething();
    }
```

Comments

- Prefer ‘//‘ for comments within functions
- Use ‘/* */‘ only for temporary removal of blocks of code.
- Use ‘/** */‘ and ‘///<‘ for automatic documentation

Conditional checks

- Put '&&' and '||' at the beginning of lines when using multiple conditionals.
- Always use curly braces when using multiple conditionals.

```
if  ( ( testA     > 10)
    &&( testB   == false
      ||testC   == false))
{
    //do something
}
```

* Use variables to name return values rather than using functions directly

Bad

```
if (doSomeParsing(someObject))
{
    //code contingent on parsing success? failure?
}
```

Good

```
bool fail = doSomeParsing(someObject);
if (fail)
{
    //This code is clearly a response to a failure
}
```

Variable declaration

- Declare variables as late as possible - at point of first use.
- One declaration per line.
- Declare loop counters in loops where possible.
- Always initialise variables at declaration.

```
int  type  = 0;
bool found = false;         //these have to be declared early so they
 can be used after the for loop

for (int i = 0; i < 10; i++)
{
    bool pass = someTestFunction();    //this pass variable isnt
 declared until it's used - good
    if (pass)
    {
        type  = typeMap[i];
        found = true;
        break;
```

```
        }
    }

    if (found)
    {
        //...
    }
```

Function parameters

- One per line.
- Add doxygen compatible documentation after parameters in the cpp file.
- Prefer references rather than pointers unless unavoidable.

```
void function(
        bool        runTests,           ///< Run unit test while processing
        MyStruct&   myStruct,           ///< Structure to modify
        OtherStr* otherStr = nullptr) ///< Optional structure object to populate (cant use reference because its optional)
{
  //...
}
```

Naming and Structure

- For structs/classes, use ‘CamelCase‘ with capital start
- For member variables, use ‘camelCase‘ with lowercase start
- For config parameters, use ‘lowercase_with_underscores‘
- Use suffixes (‘_ptr‘, ‘_arr‘, ‘Map‘, ‘List‘ etc.) to describe the type of container for complex types
- Be sure to provide default values for member variables.
- Use heirarchical objects where applicable.

```
struct SubStruct
{
    int    type = 0;
    double val  = 0;
};

struct MyStruct
{
    bool          memberVariable = false;
    double        precision      = 0.1;

    double                    offset_arr[10]  = {};
```

```
    OtherStruct*               refStruct_ptr   = nullptr;

    map<string, double>        offsetMap;
    list<map<string, double>>  variationMapList;
    map<int, SubStruct>        subStructMap;
};

//...

MyStruct myStruct = {};

if (acsConfig.some_parameter)
{
    //..
}
```

Testing

- Use TestStack objects at top of each function that requires automatic unit testing.
- Use TestStack objects with descriptive strings in loops that wrap functions that require automatic unit testing.

```
void function()
{
    TestStack ts(__FUNCTION__);

    //...

    for (auto& obs : obsList)
    {
        TestStack ts(obs.Sat.id());

        //...
    }
}
```

Documentation

- Use doxygen style documentation for function and struct headers and parameters
- '/**' for headers.
- '///<' for parameters

```
/** Struct to demonstrate documentation.
* The first line automatically gets parsed as a brief description, but
 more detailed descriptions are possible too.
*/
struct MyStruct
{
```

```
        bool    dummyBool;                  ///< The thing to the left is
     documented here
    };

    /** Function to demonstrate documentation
    */
    void function(
            bool        runTests,           ///< Run unit test while
     processing
            MyStruct&   myStruct,           ///< Structure to modify
            OtherStr* otherStr = nullptr) ///< Optional string to populate
    {
      //...
    }
```

STL Templates

- Prefer maps rather than fixed arrays.
- Prefer range-based loops rather than iterators or 'i' loops, unless unavoidable.

Bad

```
    double double_arr[10] = {};

    //..(Populate array)

    for (int i = 0; i < 10; i++)    //Magic number 10 - bad.
    {

    }
```

```
    map<string, double> doubleMap;

    //..(Populate Map)

    for (auto iter = doubleMap.begin(); iter != doubleMap.end(); iter++)
     //long, undescriptive - bad
    {
      if (iter->first == someVar)     //'first' is undescriptive - bad
      {
        //..
      }
    }
```

Good - Iterating Maps

```
    map<string, double> offsetMap;

    //..(Populate Map)

    for (auto& [siteName, offset] : doubleMap)  //give readable names to map
      keys and values
    {
        if (siteName.empty() == false)
```

```
        {

        }
    }
```

Good - Iterating Lists

```
    list<Obs> obsList;

    //..(Populate list)

    for (auto& obs : obsList)         //give readable names to list elements
    {
        doSomethingWithObs(obs);
    }
```

Special Case - Deleting from maps/lists

Use iterators when you need to delete from STL containers:

```

    for (auto it = someMap.begin(); it != someMap.end();  )
    {
        KFKey key = it->first;        //give some alias to the key/value so they're readable

        if (measuredStates[key] == false)
        {
        it = someMap.erase(it);
      }
      else
      {
        ++it;
      }
    }
```

Namespaces

Commonly used std containers may be included with 'using'

```
    #include <string>
    #include <map>
    #include <list>
    #include <unordered_map>

    using std::string;
    using std::map;
    using std::list
    using std::unordered_map;
```

Equation Conventions

In this manual we will be adhering to the following conventions

List of Symbols

- i Receiver identification or r
- j Satellite identification or s
- k Epoch number or t
- q GNSS type (GPS,GALILEO,GLONASS,QZSS)
- c Speed of light [m/s]
- x Vector of parameters to be estimated [m]
- y Vector of observations [m]
- A Design matrix
- σ Standard deviation of observable
- Δ Increment to a priori values [m]
- ωavelength or $\lambda_1, \lambda_2, \lambda_5$
- f_1, f_2, f_5 frequency
- αmbguity or N Real valued ambguity and $\bar{N}$ Integer part of real valued ambguity
- α level of significance
- βiases
- ζlock offsets
- δtlock error [s]
- κorrection - relativity

- ιonosphere or I
- τroposphere or T, T_h, T_w [m]
- M elevation dependent mapping function for the troposphere wet delay
- ξ phase wind-up error
- ϵrror in observations and unmodelled effects [m]
- ϕ_i^j Carrier phase observable (times c) [m]
- P_i^j Pseudo range observable [m]

Lets try this for example: For an undifference, uncombined float solution, the linearized observation equations for pseudorange and phase observations from satellite s to receiver r can be described as:

$$\Delta P_{r,f}^{q,s} = u_r^{q,s}.\Delta x + c.(\delta t_r^q - \delta t^{q,s}) + M_r^{q,s}.T_r + \gamma_f^q.I_{r,1}^{q,s} + d_{r,f}^q - d_f^{q,s} + \epsilon_{P,f}^q$$

$$\Delta \phi_{r,f}^{q,s} = u_r^{q,s}.\Delta x + c.(\delta t_r^q - \delta t^{q,s}) + M_r^{q,s}.T_r - \gamma_f^q.I_{r,1}^{q,s} + \lambda_f^q.N_{r,f}^{q,s} + b_{r,f}^q - b_f^{q,s} + \epsilon_{L,f}^q$$

where$\Delta P_{r,f}^{q,s}$ and $\Delta\phi_{r,f}^{q,s}$ are the respective pseudorange and phase measurements on the frequency f(f=1,2), from which the computed values are removed; $u_r^{q,s}$ is the receiver-to-satellite unit vector; Δx is the vector of the receiver position corrections to its preliminary position; δt_r^q and $\delta t^{q,s}$ are the receiver and satellite clock errors respectively; c is the speed of light in a vaccum $M_r^{q,s}$ is the elevation dependent mapping function for the troposphere wet delay from the corresponding zenith one T_r; $I_{r,1}^{q,s}$ is the ionosphere delay along the line-of-sight from a receiver to a satellite at the first frequency and $\gamma_f^q = (\lambda_f^q/\lambda_1^q)^2$; λ_f^q is the wavelength for the frequency f of a GNSS q; $N_{r,f}^{q,s}$is the phase ambguity $d_{r,f}^q$ and $b_{r,f}^q$ are the receiver hardware delays of code and phase observations respectively; $d_f^{q,s}$ and $b_f^{q,s}$ are the satellite hardware delays of code and phase observations, respectively; $\epsilon_{P,f}$ and $\epsilon_{L,f}$ are the code and phase measurement noises respectively.

[type=acronym]

PEA YAML Configuration

PEA processing options (pod_options)

Input File Options

```
input_files:

root_input_directory: /data/acs/pea/proc/exs/products

atxfiles:      [  igs14_2045_plus.atx                     ]  # Antenna models
snxfiles:      [  igs19P2062.snx                          ]  # meta data and apriori coords
blqfiles:      [  OLOAD_GO.BLQ                            ]  # ocean loading is applied
navfiles:      [  brdm1990.19p                            ]  # gnss broadcast file
sp3files:      [  gag20624.sp3                            ]  # precise orbit data
erpfiles:      [  igs19P2062.erp                          ]  # earth orintation parameters
#dcbfiles:      [  CAS0MGXRAP_20191990000_01D_01D_DCB.BSX   ]  # monthly DCB file
#clkfiles:     [  jpl20624.clk                            ]  # satellie and receiver clock
orbfiles:      [  gag20624_orbits_partials.out            ]  # need this when estimating orbits (overrides .sp3 file)
```

Listing 6: yaml input files configuration example

RINEX station data

There are numerous ways that the *pea* can access GNSS RINEX observations to process. You can specify individual rinex files to process, set it up so that it will search a particular directory, or you can use a command line flag *-rnx* <*rnxfilename*> to add an additional file to process. The data should be uncompressed (gunzipped, and not in hatanaka format), that is the *pea* expecting to just accept RINEX format].

```
station_data:

  root_stations_directory: /data/ginarn/proc/data

```

```
  rnxfiles:
    - ALIC00AUS_R_20191990000_01D_30S_MO.rnx
```

Listing 7: pea yaml processing one station example

To process one RINEX file, you need to first specify the root dircetory of where the data is being stored in *root_stations_directory*, and then the name of the RINEX file as a single entry under *rnxfiles:*, as shown in the listing reflst:pea-yaml-single-station.

```
$ pea --rnx CAS100ATA_R_20191990000_01D_30S_MO.rnx
```

Listing 8: Example showing how to add a RINEX file to the processing list form the command line

If you wanted to process another file, located in the same *root_stations_directory*, this can be achieved at the command line using the *-rnx <rnxfilename* flag, see listing reflst:pea-yaml-add-station.

```
station_data:

root_stations_directory: /data/acs/pea/proc/exs/data

rnxfiles:
- "*.rnx"                           #- searching all in file_root directory
```

Listing 9: yaml input files configuration example

Real-time streams

To process data in real-time you will need to set up the location, username annd password for the caster that you will be obtaining the input data streams from in the configuration file.

The pea supports obtaining streams from casters that use NTRIP 2.0 over http and https.

```
station_data:

  stream_root: "http://<usedname>:<password>@auscors.ga.gov.au:2101/"

  streams:
    - BCEP00BKG0
    - SSRA00CNE0
    - STR100AUS0
```

Listing 10: yaml input files configuration example

As shown in listing: , the caster url, username and password are specified within double quotes with the *stream_root* tag. In this example the streams are being obtained from the auscors caster run by Geoscience Australia. The broadcast information is being obtained from the stream *BCEP00BKG0* being supplied by BKG, and corrections to the utlra-rapid predicted orbit are being obtained from the stream *SSRA00CNE0*. The real-time data being processed is for the continuous GNSS station located at Mount Stromlo obtained from the stream *STR100AUS0*.

You can test your username and password is working correctly by running the curl command:

```
curl https://ntrip.data.gnss.ga.gov.au/ALIC00AUS0 -H "Ntrip-Version: NTRIP
    /2.0" -i  --output - -u <user>
```

output files

```
output_files:

root_output_directory:          /data/acs/pea/output/<CONFIG>/

log_level:                      warn                            #debug,
    info, warn, error as defined in boost::log

output_trace:                   true
trace_level:                    2
trace_directory:                ./
trace_filename:                 <CONFIG>-<STATION><YYYY><DDD><HH>.TRACE

output_residuals:               true

output_persistance:             false
input_persistance:              false
persistance_directory:          ./
persistance_filename:           <CONFIG>.persist

output_config:                  false

output_summary:                 true
summary_directory:              ./
summary_filename:               PEA<YYYY><DDD><HH>.SUM

output_ionex:                   false
ionex_directory:                ./
ionex_filename:                 IONEX.ionex
iondsb_filename:                IONEX.iondcb

output_clocks:                  true
clocks_directory:               ./
clocks_filename:                <CONFIG>.clk
output_AR_clocks:               true

output_sinex:                   true
sinex_directory:                ./
```

Listing 11: yaml input files configuration example

output options

```
output_options:

config_description:             EX03_AR
analysis_agency:                GAA
analysis_center:                Geoscience Australia
```

```
analysis_program:               AUSACS
rinex_comment:                  AUSNETWORK1
```

Listing 12: yaml input files configuration example

processing Options

```
processing_options:

#start_epoch:               2019-07-18 00:00:00
#end_epoch:                 2017-03-29 23:59:30
#max_epochs:                300        #0 is infinite
epoch_interval:             30          #seconds

process_modes:
user:                   false
network:                true
minimum_constraints:    false
rts:                    false
ionosphere:             false
unit_tests:             false

process_sys:
gps:            true
#glo:           true
gal:            false
#bds:           true

elevation_mask:     10   #degrees

tide_solid:         true
tide_pole:          true
tide_otl:           true

phase_windup:       true
reject_eclipse:     true            #  reject observation during satellite
     eclipse periods
raim:               true
antexacs:           true

cycle_slip:
thres_slip: 0.05

max_inno:   0
max_gdop:   30

troposphere:
model:      gpt2    #vmf3
gpt2grid:   gpt_25.grd
#vmf3dir:    grid5/
#orography:  orography_ell_5x5

ionosphere:
corr_mode:      iono_free_linear_combo
iflc_freqs:     l1l2_only   #any l1l2_only l1l5_only

pivot_station:        "USN7"
#pivot_satellite:      "G01"

code_priorities: [  L1C, L1P, L1Y, L1W, L1M, L1N, L1S, L1L,
```

```
L2W, L2P, L2Y, L2C, L2M, L2N, L2D, L2S, L2L, L2X,
L5I, L5Q, L5X]
```

Listing 13: yaml input files configuration example

Network filter parameters

```
network_filter_parameters:

process_mode:               kalman      #lsq
inverter:                   LLT         #LLT LDLT INV

max_filter_iterations:      3
max_filter_removals:        3

rts_lag:                    -1      #-ve for full reverse, +ve for limited epochs
rts_directory:              ./
rts_filename:               <CONFIG>-Orbits.rts
```

Listing 14: yaml input files configuration example

Default filter parameters: stations

```
default_filter_parameters:

  stations:
    error_model:        elevation_dependent         #uniform elevation_dependent
      code_sigmas:        [0.30]
      phase_sigmas:       [0.003]
    pos:
      estimated:          false
      sigma:              [1.0]
      proc_noise:         [0]
      #apriori:                                   # taken from other source, rinex file etc.
      #frame:              xyz #ned
      #proc_noise_model:   Gaussian
      #clamp_max:          [+0.5]
      #clamp_min:          [-0.5]
    clk:
      estimated:          true
      sigma:              [0]
      proc_noise:         [1.8257418583505538]
      #proc_noise:         [10]
      #proc_noise_model:   Gaussian
    clk_rate:
      estimated:          false
      sigma:              [10]
      proc_noise:         [1e-4]
      #clamp_max:          [1000]
      #clamp_min:          [-1000]
    amb:
      estimated:          true
      sigma:              [100]
      proc_noise:         [0]
    trop:
      estimated:          true
```

```yaml
      sigma:              [0.1]
      proc_noise:         [0.000083333]
    trop_grads:
      estimated:          false
      sigma:              [0.1]
      proc_noise:         [0.0000036]
```

Listing 15: yaml input files configuration example

Default filter parameters: satellites

```yaml
    satellites:

    clk:
      estimated:          true
      sigma:              [0]
      proc_noise:         [0.03651483716701108]

    clk_rate:
      estimated:          false
      sigma:              [0.01]
      proc_noise:         [1e-6]

    orb:
      estimated:          true
      sigma:              [5e-1, 5e-1, 5e-1, 5e-3, 5e-3, 5e-3, 5e-1]
```

Listing 16: yaml input files configuration example

Default filter parameters: eop

```yaml
  eop:
    estimated:  true
    sigma:      [30]
    #proc_noise: [0.0000036]
```

Listing 17: yaml input files configuration example

Ambiguity Resolution Options

```yaml
  ambiguity_resolution_options:
    Min_elev_for_AR:            15.0
    GPS_amb_resol:              true
    GAL_amb_resol:              false
    #Set_size_for_lambda:        10

  WL_mode:                    iter_rnd        # AR mode for WL: off, round,
     iter_rnd, bootst, lambda, lambda_alt, lambda_al2, lambda_bie
    WL_succ_rate_thres:         0.9999
    WL_sol_ratio_thres:         3.0
    WL_procs_noise_sat:         0.00001
    WL_procs_noise_sta:         0.0001

  NL_mode:                    iter_rnd        # AR mode for WL: off, round,
     iter_rnd, bootst, lambda, lambda_alt, lambda_al2, lambda_bie
    NL_succ_rate_thres:         0.9999
    NL_sol_ratio_thres:         3.0
    NL_proc_start:              86310
```

```
bias_read_mode:             30            # +1: read DSB biases, +2:
   read OSB biases, +4: read code biases, +8: read phase biases, +16: read
    satellite bias, +32: read station bias,
  bias_output_rate:          300.0
```

Listing 18: yaml ambiguity configuration example

POD YAML Configuration

The YAML configuration file for the POD allows you to specify how the and what data the POD will process and what results and statistics to report at the end. In order to use the yaml configuration file you will need to specify this at the command line, with the *-y <yaml_filename>* otherwise it will default to the traditional *POD.in, EQM.in and VEQ.in* option files.

```
$ pod -y example_configuration.yaml
```

Listing 19: calling the yaml configuration file

POD processing options (pod_options)

These options will control how the pod will process the input files, with four different options available. Only one of the options listed below can be set to true, the remainder must be set to false.

Option	Values	Comments
pod_mode_fit	true or false	Orbit Determination (pseudo-observations; orbit fitting)
pod_mode_predict	true or false	Orbit Determination and Prediction
pod_mode_eqm_int	true or false	Orbit Integration (Equation of Motion only)
pod_mode_ic_int	true or false	Orbit Integration and Partials (Equation of Motion and Variational Equations) initial condition integration

Table 6: POD YAML: processing options

```
pod_options:
# Example YAML showing different processing options
```

```
#-----------------------------------------------
pod_mode_fit:     true
pod_mode_predict: false
pod_mode_eqm_int: false
pod_mode_ic_int:  false
```

Listing 20: pod_options yaml configuration example

1. **pod_mode_fit** - this is used to fit an existing sp3 file (this is sometimes referred to as pseudo observations) with the parameters that are set later on. See pod example 1
2. **pod_mode_predict** - determine an orbit from observations and then predict the orbits path
3. **pod_mode_eqm_int** - determine the equations of motion only
4. **pod_mode_ic_int** -set up the initial conditions

Time scale(time_scale)

Option	Values	Comments
TT_time	true or false	Terrestrial (TT)
UTC_time	true or false	Universal (UTC)
GPS_time	true or false	Satellite (GPS)
TAI_time	true or false	Atomic (TAI)

Table 7: POD YAML: Time scale options

```
 time_scale:
   TT_time:  false
   UTC_time: false
   GPS_time: true
   TAI_time: false
```

Listing 21: time_scale yaml configuration example

1. **TT_time** -
2. **UTC_time** -
3. **GPS_time** -
4. **TAI_time** -

Initial Conditions (IC)

IC input format (ic_input_filename)

one is true, the other is false. If icf selected the ic_filename value specifies the path to the file.

Option	Values	Comments
sp3	true or false	sp3 format file
icf	true or false	initial conditions file

Table 8: POD YAML: Initial Conditions input format options

```
ic_input_format:
  sp3:  true   # Input a-priori orbit in sp3 format
  icf:  false  # Input a-priori orbit in POD Initial Conditions File (ICF)
    format
  ic_filename: some_file
```

Listing 22: ic_input_format yaml configuration example

IC input reference system (ic_input_refsys)

reference system for the initial conditions one is true, the other is false.

Option	Values	Comments
itrf	true or false	terestrial
icrf	true or false	celestial
kepler	true or false	polar form of celestial

Table 9: POD YAML: Initial Conditions reference system

```
 ic_input_refsys:
  itrf: true   # Initial Conditions Reference Frame: ITRF, ICRF
  icrf: false  # Initial Conditions Reference Frame: ITRF, ICRF
  kepler: false
```

Listing 23: ic_input_refsys yaml configuration example

Using Pseudo observartions

These options are used to control how pseudo observations are used by the POD.

Option	Values	Comments
pseudobs_orbit_filename	filename	*path to the observations file*
pseudobs_interp_step	int	Interval (sec) of the interpolated orbit
pseudobs_interp_points	int	Number of data points used in Lagrange interpolation (at least 2)

Table 10: POD YAML: Using pseudo observations

```
pseudobs_orbit_filename: igs19424.sp3 # Pseudo observations orbit filename
pseudobs_interp_step:    900          # Interval (sec) of the interpolated
    orbit
pseudobs_interp_points:  12           # Number of data points used in
   Lagrange interpolation
```

Listing 24: pseudo observation model yaml configuration example

Orbit arc length

Option	Values	Comments
orbit_arc_determination	int	*number of hours to integrate*
orbit_arc_prediction	int	*number of hours to predict at end of orbit arc*
orbit_arc_backwards	int	*number of hours to check before start of orbit arc*

Table 11: POD YAML: Orbit arc options

```
# Orbit arc length (in hours)
  orbit_arc_determination: 24  # Orbit Estimation arc
  orbit_arc_prediction:    12  # Orbit Prediction arc
  orbit_arc_backwards:     2   # Orbit Propagation backwards arc
```

Listing 25: orbit arc length yaml configuration example

External Orbit Comparison

In this section only one of the following options listed below can be set to true, the remainder must be set to false.

```

# External Orbit Comparison
  ext_orbit_enabled: true
  ext_orbit_type_sp3:       false       # Orbit data in sp3 format
                                        # (including position and velocity
     vectors)
  ext_orbit_type_interp:    true        # Interpolated orbit based on
     Lagrange
                                        # interpolation of sp3 file
  ext_orbit_type_kepler:    false       # Keplerian orbit
  ext_orbit_type_lagrange:  false       # 3-day Lagrange interpolation
  ext_orbit_type_position_sp3:    false  # Position and SP3 file
  ext_orbit_filename:       igs19424.sp3 # External (comparison) orbit
     filename
  ext_orbit_interp_step:    900         # Interval (sec) of the
     interpolated/Kepler orbit
  ext_orbit_interp_points:  12          # Number of data points used
                                        # in Lagrange interpolation
```

Listing 26: orbit arc length yaml configuration example

Option	Values	Comments
ext_orbit_enabled	true or false	
ext_orbit_type_sp3	true or false	
ext_orbit_type_interp	true or false	
ext_orbit_type_kepler	true or false	
ext_orbit_type_lagrange	true or false	
ext_orbit_type_position_sp3	true or false	
ext_orbit_filename	filename	*path to the orbit file*
ext_orbit_interp_step	int	Interval (sec) of the interpolatedKepler orbit
ext_orbit_interp_points	int	Number of data points used in Lagrange interpolation (at least 2)

Table 12: POD YAML: External orbit options

External orbit reference frame (ext_orbit_frame)

Option	Values	Comments
itrf	true or false	terrestrial
icrf	true or false	celestial
kepler	true or false	kepler orbital elements

Table 13: POD YAML: External orbit reference system

```
 ext_orbit_frame:
  itrf: true           # External orbit reference frame - ITRF
  icrf: false          # External orbit reference frame - ICRF
  kepler: false
```

Listing 27: external orbit reference frame yaml configuration example

Earth Orientation Parameters

In this section only one of the following options listed below can be set to true, the remainder must be set to false.

EOP type

```
  EOP_soln_c04:   true        # IERS C04 solution : EOP_sol = 1
 EOP_soln_rapid: false        # IERS rapid service/prediction center (RS/PC
   ) Daily : EOP_sol = 2
 EOP_soln_igs:   false        # IGS ultra-rapid ERP + IERS RS/PC Daily (dX,
   dY) : EOP_sol = 3. Need both rapid_file AND igs_file
 EOP_soln_c04_file: eopc04_14_IAU2000.62-now
 EOP_soln_rapid_file: finals2000A.daily
```

Table 14: POD YAML: Earth Orientation Parameter solution options

Option	Values	Comments
EOP_soln_c04	true or false	C04 is the IERS solution
EOP_soln_rapid	true or false	Rapid is the rapidprediction center solution
EOP_soln_igs	true or false	igs is the ultra-rapid solution using partials. To use this you need both the rapid file and partials file.
EOP_soln_c04_file	filename	
EOP_soln_rapid_file	filename	
ERP_soln_igs_file	filename	
EOP_soln_interp_points	int	

```
ERP_soln_igs_file: igu18543_12.erp
EOP_soln_interp_points: 4    # EOP solution interpolation points
```

Listing 28: eop estimation options

IAU Precession-Nutation model

Table 15: POD YAML: EOP model options

Option	Values	Comments
eop_soln_interp_points	int	number of data points to be used in an eop interpolation (at least 2!)
iau_model_2000	true or false	
iau_model_2006	true or false	

```
# IAU Precession-Nutation model:
iau_model_2000: true         # IAU2000A: iau_pn_model = 2000
iau_model_2006: false        # IAU2006/2000A: iau_pn_model = 2006

```

Listing 29: eop model

Input files

```
# Gravity model file
gravity_model_file: goco05s.gfc
# goco05s.gfc, eigen-6s2.gfc, ITSG-Grace2014k.gfc

# Planetary/Lunar ephemeris - JPL DE Ephemeris
```

Table 16: POD YAML: Input files

Option	Values	Comments
gravity_model_file	filename	
DE_fname_header	filename	Emphemeris header file
DE_fanme_data	filename	Emphemeris data file
ocean_tides_model_file	filename	
leapsec_filename	filename	leapseconds to be added
satsinex_filename	filename	sinex file with satellite meta-data

```
DE_fname_header: header.430_229
DE_fname_data:   ascp1950.430

# Ocean tide model file
ocean_tides_model_file: fes2004_Cnm-Snm.dat
# FES2004 ocean tide model

# Leap second filename
leapsec_filename: leap.second

# Satellite metadata SINEX
satsinex_filename: igs_metadata_2063.snx
```

Listing 30: yaml example for general input files

Output options

Table 17: POD YAML: Output options

Option	Values	Comments
sp3_velocity	true or false	if you wish to write out the velocities for comparison
partials_velocity: true or false	if you wish to write velocity vector partials to the output file	

```
# Write to sp3 orbit format: Option for write Satellite Velocity vector
sp3_velocity: false       #  Write Velocity vector to sp3 orbit

#-------------------------------------------------------------------
# Write partials of the velocity vector w.r.t. parameters into the orbits_partials output file:
partials_velocity: false   # Write out velocity vector partials wrt orbital state vector elements
```

Listing 31: yaml example for output file optionss

Variational Equation Options

Table 18: POD YAML: VEQ ref system

Option	Values	Comments
veq_integration	true or false	pod mode overides it anyway. Ignore.
ITRF **ICRF** **kepler**	true or false true or false true or false	reference_frame one must be true

```
# Variational Equations
veq_integration: false

#-----------------------------------------------------------------------
# Reference System for Variational Equations'  - Partials & Parameter
    Estimation
veq_refsys:
itrs: true      # ITRS: Terrestrial Reference System
icrs: false     # ICRS: Celestial Reference System
kepler: false
```

Listing 32: yaml example for variational equation options

General Options

Table 19: POD YAML: general options

Option	Values	Comments
estimator_iterations	int	integrate this number of times, using the generated initial conditions from the previous run as a start point

```
  # Parameter Estimation
  estimator_iterations: 2
```

Listing 33: yaml example for output file options

Apriori solar radiation models

```
srp_apriori_model:
  no_model:   false
  cannon_ball_model:   true
  simple_boxwing_model:   false
  full_boxwing_model:     false
```

Listing 34: yaml example for apriori srp model options

Option	Values	Comments
no_model	true or false	
cannon_ball_model	true or false	see **??**
simple_boxwing_model	true or false	
full_boxwing_model	true or false	

Table 20: POD YAML: Apriori SRP model

Estimated Solar radiation models

Option	Values	Comments
ECOM1	true or false	
ECOM2	true or false	
hybrid	true or false	mix of ECOM1 and ECOM2
SBOXW	true or false	Simple box wing model
EMPirical models	true or false	Empirical is independent of the other four

Table 21: POD YAML: Estimated SRP models

```
srp_apriori_model:
  no_model:              false
  cannon_ball_model:     true
  simple_boxwing_model:  false
  full_boxwing_model:    false
```

Listing 35: yaml example for apriori srp model options

gravity_model

Type of gravity model to apply, only one option can be true.

Option	Values	Comments
central_force	true or false	
static_gravity_model	true or false	
time_variable_model	true or false	
iers_geopotential_model	true or false	

Table 22: POD YAML: Gravity Models

```
gravity_model:
  central_force:          false   # Central force gravity field
      : gravity_model = 0
  static_gravity_model:   false   # Static global gravity field model
      : gravity_model = 1
  time_variable_model:    true    # Time-variable global gravity field
    model : gravity_model = 2
```

```
iers_geopotential_model: false   # IERS conventional geopotential model
     : gravity_model = 3
```

Listing 36: yaml example for gravitational force model options

stochastic pulse (pulse)

Do not mix pulses in R/T/N (terrestrial) with pulses in (X/X/Z)

Option	Values	Comments
enabled	true or false	then if true:
epoch_number	int	number of epochs to apply pulses each day
offset	int	seconds until the first pulse of the day
interval	int	seconds between each pulse (after the first)
directions		
x_direction	true or false	
y_direction	true or false	
z_direction	true or false	
r_direction	true or false	
t_direction	true or false	
n_direction	true or false	

Table 23: POD YAML:stochastic pulse options

```
pulse:
enabled:      false
epoch_number:     1    # number of epochs to apply pulses
offset:       43200    # since the start of day
interval:     43200    # repeat every N seconds
directions:
  x_direction:   true
  y_direction:   true
  z_direction:   true
  r_direction:   false
  t_direction:   false
  n_direction:   false
reference_frame:
  icrf:          true
  orbital:       false
```

Listing 37: yaml example for gravitational force model options

EQM options

Integration Step

```
# Numerical integration method
```

Option	Values	Comments
RK4_integrator_method	true or false	Do not use RK4 for veq as it is not implemented
RKN7_integrator_method	true or false	only one can be true
RK8_integrator_method	true or false	
integrator_step	int	step size in seconds

Table 24: POD YAML: Integration Step models

```
# Runge-Kutta-Nystrom 7th order RKN7(6): RKN7, Runge-Kutta 4th order: RK4,
    Runge-Kutta 8th order RK8(7)13: RK8
integration_options:
  RK4_integrator_method: false
  RKN7_integrator_method: true
  RK8_integrator_method: false
  integrator_step: 900        # Integrator stepsize in seconds
```

Listing 38: yaml example for gravitational force model options

Gravity Field

Option	Values	Comments
enabled	true or false	and if true:
gravity_degree_max		maximum model terms in spherical harmonic expansion
timevar_degree_max		maximum time variable model terms in spherical harmonic expansion

Table 25: POD YAML: Gravity Models

```
# Gravitational Forces
gravity_field:
  enabled: true
  gravity_degree_max:      15      # Gravity model maximum degree/order (d/o
    )
  timevar_degree_max:      15      # Time-variable coefficients maximum d/o
```

Listing 39: yaml example for gravitational force model options

planetary_perturbations:

Option	Values	Comments
enabled	true or false	Uses the emphemeris

Table 26: POD YAML: planetary perturbations

```
# Planetary Gravitational Forces
  planetary_perturbations:
    enabled: true
```

Listing 40: yaml example for planetary pertubations

tidal_effects:

Option	Values	Comments
solid_tides_nonfreq	True or False	frequency independent Solid Earth Tides
solid_tides_freq	True or False	frequency dependent Solid Earth Tides
ocean_tides	True or False	uses the ocean tides file
solid_earth_pole_tides	True or False	tide induced earth spin rotation not about the centre of the ellipsoid
ocean_pole_tide	True or False	ocean response to the above
ocean_tides_degree_max	True or False	maximum model term in spherical harmonic expansion

Table 27: POD YAML: tidal effects

```
    tidal_effects:
  enabled: true
  solid_tides_nonfreq:    true   # Solid Earth Tides frequency-independent
     terms
  solid_tides_freq:       true   # Solid Earth Tides frequency-dependent
     terms
  ocean_tides:            true   # Ocean Tides
  solid_earth_pole_tides: true   # Solid Earth Pole Tide
  ocean_pole_tide:        true   # Ocean Pole Tide
  ocean_tides_degree_max: 15     # Ocean Tides model maximum degree/order
```

Listing 41: yaml example for tidal effects

relativistic_effects:

non_gravitational_effects:

Models to be applied:

```
# Non-gravitational Effects
non_gravitational_effects:
  enabled: true
```

Option	Values	Comments
enabled	true or false	Lens Thinning, SchwarzChild and deSitter effects, there are no means to separate these effects currently. The Lens Thirring effect is calculated but subsequently ignored in the POD.

Table 28: POD YAML:relativistic_effects

Option	Values	Comments
solar_radiation	true or false	radiation push from the sun
earth_radiation	true or false	radiation push from the earth
antenna_thrust	true or false	reverse thrust from antenna radiation

Table 29: POD YAML: non gravitational effects

```yaml
  solar_radiation: true
  earth_radiation: true
  antenna_thrust:  true
```

Listing 42: yaml example for non gravitational effects

Empirical parameters

```yaml
# Non-gravitational Effects
srp_parameters:
  ECOM_D_bias:  true
  ECOM_Y_bias:  true
  ECOM_B_bias:  true
  EMP_R_bias:   true
  EMP_T_bias:   true
  EMP_N_bias:   true
  ECOM_D_cpr:   true
  ECOM_Y_cpr:   true
  ECOM_B_cpr:   true
  ECOM_D_2_cpr: false
  ECOM_D_4_cpr: false
  EMP_R_cpr:    true
  EMP_T_cpr:    true
  EMP_N_cpr:    true
  cpr_count:    1
```

Listing 43: yaml example for srp parameters

NB EQM and VEQ srp parameters MUST be identical. May move into pod_options in future. overrides are not implemented yet. Ig-

Option	Values	Comments
ecom_d_bias	true or false	
ecom_y_bias	true or false	
ecom_b_bias	true or false	
ecom_d_cpr	true or false	(only ECOM1hybrid)
ecom_y_cpr	true or false	(only ECOM1hybrid)
ecom_b_cpr	true or false	
ecom_d_2_cpr	true or false	(only ECOM2hybrid)
ecom_d_4_cpr	true or false	(only ECOM2hybrid)
emp_r_bias	true or false	
emp_t_bias	true or false	
emp_n_bias	true or false	
emp_r_cpr	true or false	
emp_t_cpr	true or false	
emp_n_cpr	true or false	
cpr_count	int	empirical cpr count

Table 30: POD YAML: Configuration options for solar radiation pressure models

nore for now. We imagine overrides at the system, block (sat type) and individual satellite level

*overides**

***This section has not yet been implemented in the POD, and is a placehodler for future versions.**

In this section put any system, block or PRN overrides that are different to the ones chosen before

```
overrides:
  system:
  GPS:
  srp_apriori_model:
  no_model:   false
  cannon_ball_model:   true
  simple_boxwing_model:   false
  full_boxwing_model:     false
  GAL:
  srp_apriori_model:
  no_model:   false
  cannon_ball_model:   false
  simple_boxwing_model:   false
  full_boxwing_model:     true
  GLO:
  srp_apriori_model:
  no_model:   false
  cannon_ball_model:   false
  simple_boxwing_model:   true
  full_boxwing_model:     false
  BDS:
  srp_apriori_model:
  no_model:   false
  cannon_ball_model:   false
  simple_boxwing_model:   true
```

```yaml
  full_boxwing_model:     false
block:
  GPS-IIF:
    srp_apriori_model:
    no_model:   false
    cannon_ball_model:   false
    simple_boxwing_model:   true
    full_boxwing_model:     false
  # GPS BLK IIF use ECOM2 parameters
  srp_parameters:
    ECOM_D_bias:   true
    ECOM_Y_bias:   true
    ECOM_B_bias:   true
    ECOM_D_2_cpr: false
    ECOM_D_4_cpr: false
    ECOM_B_cpr:   true
prn:
  G01:
    srp_apriori_model:
      no_model:   false
      cannon_ball_model:   false
      simple_boxwing_model:   false
      full_boxwing_model:     true
```

Listing 44: yaml example for override

Manual conventions and Tips

Converting latex to markdown pandoc

To display code listings use the package `https://www.overleaf.com/learn/latex/Code_listing`listings.

Acknowledgements

In this section we wish to acknowledge the use of and heritage of some of the source code that we have used to help develop the Ginan.

Eclipse Routine. The routines to calculate the eclipsing times for GPS satellites were based off the original routines written by Jan Kouba, they have since been heavily modified.

JPL Planetary Ephemerides. We are using the Jet Propulison (JPL) Planetary and Lunar Ephemerides processing program (ftp://ssd.jpl.nasa.gov/pub/eph/planets/fortran/), in particular the routines:

- CONST.f
- FSIZER3.f
- INTERP.f
- PLEPH.f
- SPLIT.f

We have modified the following subroutines:

- asc2eph.f90
- STATE.f90

so that there is no longer a dependency on a binary file produced in the original JPL form.

Standards of Fundamental Astronomy (SOFA) routines. We have used a number of routines obtained from SOFA, http://www.iausofa.org/:

- anp.for
- bi00.for
- bpn2xy.for
- bpn2xy.for
- c2ixys.for

- c2tcio.for
- cal2jd.for
- cp.for
- cr.for
- era00.for
- fad03.for
- fae03.for
- faf03.for
- faju03.for
- fal03.for
- falp03.for
- fama03.for
- fame03.for
- fane03.for
- faom03.for
- fapa03.for
- fasa03.for
- faur03.for
- fave03.for
- gmst00.for
- gmst06.for
- gmst_iers.f03
- ir.for
- jd2cal.for
- jdcalf.for
- numat.for
- nut00a.for
- obl80.for

- pn00a.for
- pn00.for
- pnm00a.for
- pnm06a.for
- pom00.for
- pr00.for
- rx.for
- rxr.for
- ry.for
- rz.for
- s00.for
- s06.for
- sp00.for
- taiutc.for
- tide_pole_oc.f90
- tide_pole_se.f90
- time_GPS.f90
- time_TAI.f90
- time_TT.f90
- time_TT_sec.f90
- time_UTC.f90
- tr.for
- xy06.for
- xys00a.for
- xys06a.for

International Earth Rotation Service (IERS) routines. The following routines we originally sourced fromthe IERS:

- interp_iers.f
- CNMTX.F

- FUNDARG.F
- LAGINT.f
- ORTHO_EOP.F
- PMSDNUT2.F
- RG_ZONT2.F
- UTLIBR.F
- IERS_CMP_2015.F

Listings