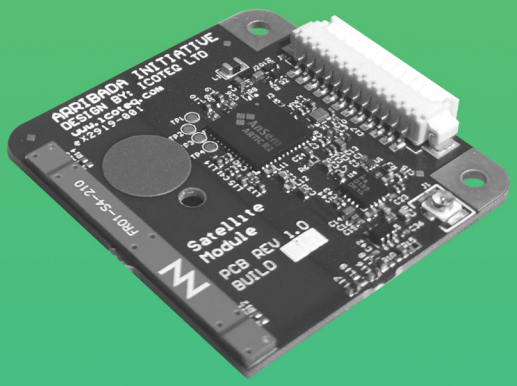
ARRIBADA initiative

Arribada Horizon



Arribada Horizon ARTIC R2 Developer's Kit User Manual & Quickstart Guide

Version 1.0.2 | April 2020

HORIZON-DEV-ARTIC-R2

https://arribada.org | hello@arribada.org

SHUTTLEWORTH FOUNDATION

1. Introduction

This document provides a brief overview of how to setup, install and provision an Horizon tracker device.

These instructions are to be used with:

- v1 nRF52840 bootloader
- v1 nRF52840 application firmware
- v5 configuration file format version
- v2.0.0 python tools

These instructions should not be assumed to be correct for versions other than those state above.

2. Prerequisites

2.1. Hardware

- 1 x Horizon board rev 3.0
- (Optional) 1 x SARA-U270 cellular module rev 1.0
- (Optional) 1 x ARTIC R2 satellite module rev 1.0
- 1 x PC running Ubuntu Linux (recommend Ubuntu 20.04 LTS) & 1 x Micro A USB cable
- 1 x J-Link JTAG programmer or equivalent (required for programming nRF52840 device)

2.2. Software

The following release packages should be downloaded and installed by cloning the following git repository: https://github.com/arribada

- Python tools software package | arribada_tools-2.0.0.zip
- nRF52840 bootloader file | https://github.com/arribada/horizon-v9-firmware/blob/master/horizon_bootloader_18a6102.hex
- nRF52840 firmware image DFU file | https://github.com/arribada/horizon-v9-firmware

Ubuntu 20.04 LTS is the recommended Linux operating system to use.

Run the following commands to install necessary software and check the version numbers are correct:

```
$ sudo apt-get update

$ sudo apt-get install -y git python3 python3.7 python3-pip libglib2.0-dev libyaml-dev usbutils python3-dev

$ sudo pip3 install python-dateutil==2.8.0

$ git clone https://github.com/arribada/horizon.git

$ cd horizon

$ sudo python3 setup.py install
```

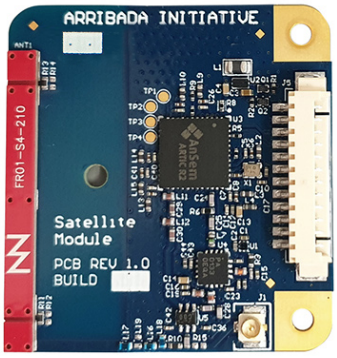
3. Hardware Preparation

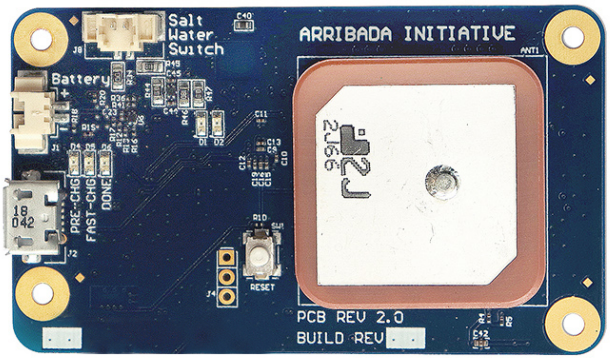
3.0.0.1. Horizon board rev 3.0

With the board(s) held in the following orientation:

ARTIC R2 Developer's Kit Contents:

1 x Horizon board
1 x ARTIC R2 module (daughterboard)
1 x Picoblade cable assembly
1 x MicroUSB cable





The main Horizon board accepts either an ARTIC R2 module, or a Cellular SARA-U270 module.

Use the included Picoblade cable (Developer's Kit) assembly to connect a module to the Horizon board.

4.2V min to 5.4V max input supply voltage (single use primary cells).

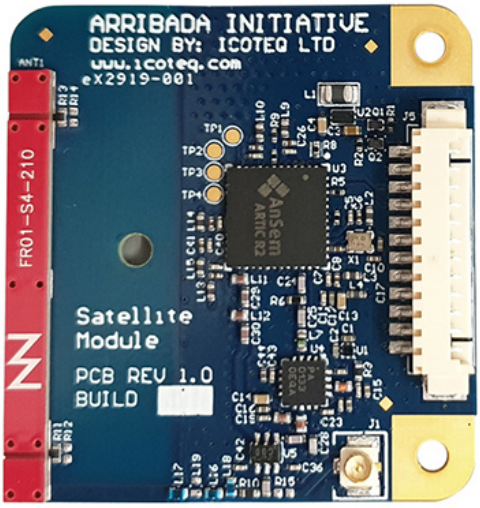
3.4 to 4.2V max input supply voltage (lithium rechargeable cell).

Connectors / preparation;

- Attach a Salt Water Switch via a cable to the Salt Water Switch picoblade connector (top edge connector)
- Attach the USB micro B cable between a Linux PC and the micro USB connector (left edge, lower connector)
- Attach a Lithium-ion or Lithium polymer battery to the JST connector (left edge, upper connector)

3.0.0.1. Connecting Horizon modules

There are two compatible modules currently available. A SARA-U270 Cellular module and an ARTIC RS Argos Satellite module. Both modules are connected to the GPS board via the picoblade connector.

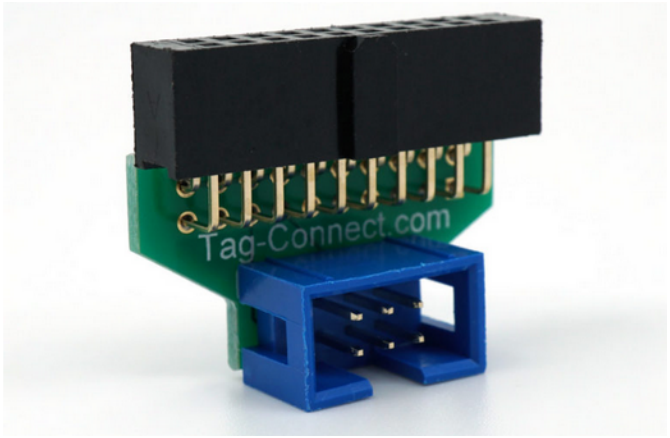


- Attach the antenna via a cable to the U.FL antenna connector (left edge connector)
- Attach the Horizon board to the module via the picoblade connector (edge)

- On the cellular module, attach the battery via a JST header and cable to the module and insert a SIM card into the sim card holder. Ensure that you have an active data plan and APN in preparation for configuration.

Firmware programming

First time secure bootloader programming must be done with a JTAG compatible device. Developer Kit's are shipped pre-programmed. Programming is only necessary to bring up new boards, or to upgrade or downgrade devices. As the tool nrjprog is used a SEGGER J-link is required.

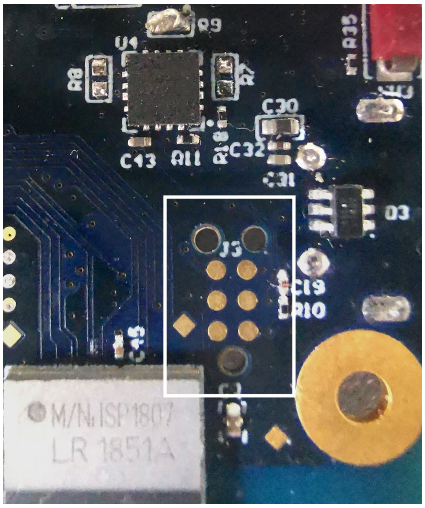


Probe cable selection

The following debug probe and connector are compatible with the Developer Kit.

1 x TC2030-IDC-NL
1 x ARM20-CTX 20-Pin to TC2030-IDC Adapter for Cortex

The J-Link debug probe should be pushed on to the 6-pin debug header as indicated in the white box to the right.



To program the board, once the J-Link debug probe has been connected to the target board as show above, then run the following commands from the arribada_nrf52840_tracker root directory:

```
$ make -C ports/nrf52840/bootloader erase
$ make -C ports/nrf52840/bootloader flash_softdevice
$ make -C ports/nrf52840/bootloader flash
```

To confirm the board is correctly programmed power it and check that the LED indicator is cycling between red, green and blue

3.0.1. Application programming via secure bootloader

```
$ pip install nrfutil

$ sudo nrfutil dfu usb-serial -pkg PACKAGE.zip -p /dev/ttyACM0
```

4. User interface

4.1. Push buttons

The following switches and push buttons are provided:

- DFU mode push button
- Hard reset push button

The DFU button operates as follows:

Tracker state	Operation	Action
Power cycle or hard reset	Hold DFU button down while resetting / power cycling	Enter secure boot DFU mode
Operational	Hold DFU button for 1 second and release	GPS test
	Hold DFU button for 3 seconds and release	Cellular test
	Hold DFU button for 5 seconds and release	Satellite test

Upon pressing the RESET button, with the USB cable connected to a PC, the status LED shall indicate the provisioning state accordingly as outlined in LED indicators.

Refer to the section Test modes for more information on the test modes.

4.2. LED indicators

The Horizon board is fitted with a single RGB status LED. The LED is used to indicate the system state during and immediately after start-up.

LED State	Description	Notes
Flashing Red	Fatal device communication error during boot-up Refer to UART debug output trace for more information	
Solid Red	Missing configuration file or log file ie provisioning incomplete	
Green	Configuration file checks passed	LED will stay on for 5 seconds
Flashing White	GPS test fix	
Solid White	GPS test fix locked and fixed for minimum required time period.	LED will stay on for 5 seconds
Flashing Yellow	Test cellular data connection in progress	
Solid Yellow	Test cellular data connection made and IoT message success	LED will stay on for 5 seconds
Flashing Blue	Test message to satellite sending in progress	
Solid Blue	Test satellite message sent	LED will stay on for 5 seconds
Off	Operational	
Red - Green - Blue Alternating	Secure bootloader DFU mode	

5. Provisioning and deployment guide

It is recommended the provisioning steps be carried out in the order described in this guide.

> By default the configuration tools will talk to the first enumerated USB tracker device on the bus. It is possible to override this using the USB_INDEX environment variable e.g., USB_INDEX=1 will talk to the 2nd enumerated device on the bus, etc. If USB_INDEX is not set

then USB_INDEX=0 is assumed.

5.1. Cellular AWS user account setup

The first-time AWS account configuration is required to be done only once. The main purpose is to create an **Admin** user group and an **IoTConsumer** user group with appropriate levels of access privileges. The Admin user group shall be assigned full administrator privileges whereas the IoTConsumer user group shall be assigned basic read-only priveleges that permits the downloading of data files form the IoT service.

The steps in this guide assume you have already created an AWS account with a root user (which is normally linked to a designated e-mail address) and you have access to the AWS console at https://console.aws.amazon.com.

5.1.1. Create admin group

Follow the steps below:

1. Open the IAM management console at https://console.aws.amazon.com/iam
2. Select **Groups** from the menu on the left hand side
3. Click the **Create New Group** button at the top of the screen
4. When prompted for a group name enter **Admin** and then click the **Next Step** button at the bottom
5. On the **Attach Policy** screen, enter the keyword "admin" into the filter and select the checkbox next to the **AdministratorAccess** policy. Click the **Next Step** button at the bottom
6. Review the settings you have just entered on the next screen and then click the **Create Group** button at the bottom.

5.1.2. Create IoT consumer group

We will repeat the same process as before with some slight alterations:

1. Open the IAM management console at https://console.aws.amazon.com/iam
2. Select **Groups** from the menu on the left hand side
3. Click the **Create New Group** button at the top of the screen
4. When prompted for a group name enter **IoTConsumer** and then click the **Next Step** button at the bottom
5. On the **Attach Policy** screen, enter the keyword "iotanalytics" into the filter and select the checkbox next to the **AWSIoTAnalyticsReadOnlyAccess** policy. Click the **Next Step** button at the bottom
6. Review the settings you have just entered on the next screen and then click the **Create Group** button at the bottom.

5.1.3. Create user accounts

For each user account you wish to create, execute the steps below:

1. Open the IAM management console at https://console.aws.amazon.com/iam
2. Select **Users** from the menu on the left hand side
3. Click the **Add User** button at the top of the screen
4. Assign a unique user name for the user and enter this into the **User name** box
5. Select the check box against **Programmatic access** only for the **Access type** then click the **Next: Permissions** button at the bottom
6. Select either the **Admin** or **IoTConsumer** group by clicking on the appropriate check box on the left then click the **Next: Tags** button at the bottom
    - Select the group you wish the user to be a member based on their role
7. Skip this screen by pressing the **Next: Review** button at the bottom
8. Review the settings and then click the **Create user** button at the bottom
9. On the next page click the **Download .csv** button to download the autogenerated access key ID and Access secret key
    - This information is downloaded to the file `accessKeys.csv` and is required for login access when using the AWS command-line tools

5.2. Configure user security credentials

Execute the following commands from the command line:

```
$ aws configure

AWS Access Key ID []: <enter your AWS Access Key ID here>
AWS Secret Access Key []: <enter your AWS Secret Access Key here>
Default region name []: <enter region name eg us-west-2>
Default output format [None]: <leave blank>
```

Note that the above process merely creates a file called `~/.aws/credentials`. This is a shared credentials file that is used by all the AWS command-line tools to authenticate remote user access.

> The above steps rely upon the AWS CLI tool being installed. Please make sure all dependency software is installed as outlined in the Plastic Tracker Quick Start Guide#Prerequisites section.

5.3. Installation of IoT configuration into AWS account

The following steps may only be carried out by a user that is a member of the **Admin** group and need only be carried out once:

```
$ cd arribada_tools-x.y.z
$ cd certificates
  # Replace UNIQUENAME below with a unique name identifier. This is
required as all AWS instances must have a unique identifier
$ aws_config --install --namespace UNIQUENAME

2019-03-28 15:32:47,748 aws_config INFO Generating root CA certificate for
arribada...
2019-03-28 15:32:47,886 aws_config INFO Installing arribada onto AWS...
2019-03-28 15:32:47,886 aws INFO register_root_ca
2019-03-28 15:32:47,906 credentials INFO Found credentials in shared
credentials file: ~/.aws/credentials
2019-03-28 15:32:49,324 aws INFO create_iot_policy
2019-03-28 15:32:50,237 aws INFO create_iot_thing_group
2019-03-28 15:32:51,141 aws INFO create_s3
2019-03-28 15:32:53,290 aws INFO deploy_sam_packages
2019-03-28 15:33:01 Building resource
'ArribadaPushShadowIoTAnalyticsFunction'
2019-03-28 15:33:01 Running PythonPipBuilder:ResolveDependencies
2019-03-28 15:33:06 Running PythonPipBuilder:CopySource

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml
```

```
Commands you can use next
=========================
[*] Invoke Function: sam local invoke
[*] Package: sam package --s3-bucket <yourbucket>

Uploading to 43382c729305e71a5c4fa118fe669aa5  1036 / 1036.0  (100.00%)
Successfully packaged artifacts and wrote output template to file
/tmp/tmp.yaml.
Execute the following command to deploy the packaged template
aws cloudformation deploy --template-file /tmp/tmp.yaml --stack-name <YOUR
STACK NAME>

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - PushShadowIOT
2019-03-28 15:34:01 Building resource
'ArribadaPushLoggingIoTAnalyticsFunction'
2019-03-28 15:34:01 Running PythonPipBuilder:ResolveDependencies
2019-03-28 15:34:03 Running PythonPipBuilder:CopySource

Build Succeeded

Built Artifacts  : .aws-sam/build
Built Template   : .aws-sam/build/template.yaml

Commands you can use next
=========================
[*] Invoke Function: sam local invoke
[*] Package: sam package --s3-bucket <yourbucket>

Uploading to 9609bf1b3f7d1e52c6b7de5b40d4eeb7  3933 / 3933.0  (100.00%)
Successfully packaged artifacts and wrote output template to file
/tmp/tmp.yaml.
Execute the following command to deploy the packaged template
aws cloudformation deploy --template-file /tmp/tmp.yaml --stack-name <YOUR
STACK NAME>

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - PushLoggingIOT
2019-03-28 15:34:53,832 aws INFO create_iot_pipelines
2019-03-28 15:35:14,954 aws INFO create_iot_rules
2019-03-28 15:35:21,723 aws_config INFO Saving certificate and keys for
arribada...
```

```
2019-03-28 15:35:21,725 aws_config INFO Saving unique certificate ID
7917f42e527c1ff22ef8a47d7749f78f154fe8831acab3eed73bba16e483b755 from AWS
for arribada...
```

Before executing the above steps please ensure that the AWS `sam` tool is installed and on your `PATH` as described in the Plastic Tracker Quick Start Guide#Prerequisites section

Upon completion, the following files should be visible in the certificates directory:

```
$ ls -la
total 32
drwxrwxr-x  2 liamw liamw 4096 Mar 28 11:35 .
drwxrwxr-x 11 liamw liamw 4096 Mar 28 08:47 ..
-rw-rw-r--  1 liamw liamw 1103 Mar 28 11:35 arribada.cert
-rw-rw-r--  1 liamw liamw   64 Mar 28 11:35 arribada.cid
-rw-rw-r--  1 liamw liamw 1704 Mar 28 11:35 arribada.key
-rw-rw-r--  1 liamw liamw  451 Mar 28 11:35 arribada.pubkey
-rw-rw-r--  1 liamw liamw  744 Mar 28 08:47 README
-rw-rw-r--  1 liamw liamw 1758 Mar 28 08:47
VeriSign-Class-3-Public-Primary-Certification-Authority-G5.pem
```

The file `arribada.cert` is an automatically generated file. It is the root CA certificate that has now been installed for your AWS account and is used as the root CA for generating all subsequent device certificates and keys by `aws_config`. This CA certificate is used by the AWS server to authenticate that any uploaded device certificates stem from a trusted root.

The file `VeriSign-Class-3-Public-Primary-Certification-Authority-G5.pem` is supplied as part of the arribada-tools-x.y.z package. It is the primary CA used by all Amazon AWS services and can be used to authenticate the AWS servers.

These files should not be deleted. If you accidentally delete these files, you will not be able to provision any tracker devices without first doing an `--uninstall` and then repeat the above `--install` step.

5.4. Provisioning of device certificate and device key into AWS

The following steps may only be carried out by a user that is a member of the **Admin** group and should be carried out for each tracker board:

```
$ aws_config --register_thing yourThingName

2019-03-28 15:26:31,265 aws_config INFO Creating certificate and keys for
yourThingName
2019-03-28 15:26:31,312 aws_config INFO Registering yourThingName into AWS
2019-03-28 15:26:31,591 credentials INFO Found credentials in shared
credentials file: ~/.aws/credentials
2019-03-28 15:26:34,962 aws_config INFO Saving certificate and key files
for yourThingName
2019-03-28 15:26:34,963 aws_config INFO Creating default device shadow
yourThingName
{
    "metadata": {
        "desired": {
            "device_status": {
                "last_log_file_read_pos": {
                    "timestamp": 1553786796
                }
            }
        }
    },
    "state": {
        "desired": {
            "device_status": {
                "last_log_file_read_pos": 0
            }
        }
    },
    "timestamp": 1553786796,
    "version": 1
}
```

Note that *yourThingName* should be unique for each tracker board being registered. Upon completion, the following new files should be visible in the certificates directory:

```
$ ls -la yourThingName.*
-rw-rw-r-- 1 liamw liamw 1135 Mar 28 15:26 yourThingName.cert
-rw-rw-r-- 1 liamw liamw 1704 Mar 28 15:26 yourThingName.key
-rw-rw-r-- 1 liamw liamw  451 Mar 28 15:26 yourThingName.pubkey
```

The file `yourThingName.cert` is an automatically generated file. It is the device certificate that has now been installed into your AWS account against the new thing called *yourThingName*. Associated with this device certificate is the device's private key called `yourThingName.key`. Both these files must now be registered into the plastic tracker's cellular modem as outlined in the following section.

5.5. Provisioning of per device certificate and keys into tracker

The following steps should be carried out for each tracker board:

```
$ sudo cellular_config --root_ca
VeriSign-Class-3-Public-Primary-Certification-Authority-G5.pem --cert
yourThingName.cert --key yourThingName.key

<Add example output here>
```

5.6. JSON configuration file preparation

A `yourThingName.json` file should be created for each tracker board that has been registered into AWS. Each file should be largely identical with the exception of device-specific parameters that should be set as follows.

For the IOT configuration, the following fields are needed for the device-specific IOT cellular AWS configuration:

```
{
 "version": 1, # Recommended to be set to 1 for first file version
 "iot": {
  "cellular": {
            "aws": {
                    "arn": "xxxxxxxxx.iot.us-west-2.amazonaws.com"
                    "thingName": "yourThingName"
            }
  }
 }
}
```

Note that the value for the `"arn"` field can be obtain by running the following command:

```
$ aws_config --get_iot_endpoint

2019-03-28 19:43:22,359 credentials INFO Found credentials in shared
credentials file: ~/.aws/credentials
a8fb7n41z7p2n.iot.us-west-2.amazonaws.com
```

The ARN is specific to each AWS account and does not change unless you switch to different AWS account's login credentials.

5.7. Check Horizon board status

To obtain system status information, enter the following command

```
$ sudo tracker_config --status
{ "cfg_version": 5, "fw_version": 1, "unique_device_identifier": xxxxxxx, "cellular_module_present":
true, "satellite_module_present": false }
```

The firmware version and configuration version must match the version requirements for the quick start guide as outlined in the introduction. If these versions do not match, then it is advised not to proceed further.

5.8. Flash memory erase

It is recommended that flash memory is erased in its entirety if the board is being programmed for this first time. The following commands will erase the entire flash chip and then reset the CPU:

```
$ sudo tracker_config --reset FLASH
$ sudo tracker_config --reset CPU
```

5.9. Erase existing configuration

This command will erase all existing configuration settings held in RAM before proceeding with provisioning:

```
$ sudo tracker_config --erase
```

Now confirm the currently active configuration stored in RAM is "empty":

```
$ sudo tracker_config --read current_config.json
$ cat current_config.json

{
    "version": 0,     # TBD: will this be the default version when the system is not programmed?
    "logging": {
        "fileSize": 0,
        "fileType": "LINEAR",
        "startEndSyncEnable": false
    },
    "rtc": {
        "dateTime": "Mon Jan  1 00:23:40 2018"
    }
}
```

The status LED shall be solid RED to indicate that the configuration is not yet valid. The provisioning process can now be carried out as normal as outlined in the following sections.

5.10. Program the GPS almanac

> The uploading of a GPS almanac is not mandatory and can be skipped if this is not needed.

The GPS almanac data file can be downloaded via HTTP from the u-blox AssistNow offline servers. Refer to the following guide for more

information on registering for this service:

https://www.u-blox.com/sites/default/files/products/documents/MultiGNSSAssistNow_QuickStart_%28UBX-14003139%29.pdf

Upon registration to the service, a token shall be issued that permits access to the HTTP server. Typing the following URL format into a web browser should obtain a new `mgaoffline.ubx` file:

http://offline-live1.services.u-blox.com/GetOfflineData.ashx?token=insert_your_token_here&gnss=gps,glo&period=4

This assumes an almanac for a 4 week period with data for both the GNSS and GLONASS satellite systems.

Note that it is necessary to add your own token to the URL otherwise this will result in a `Bad Request` error from the AssistNow offline server.

To apply the almanac file to the tracker board, enter the following command:

```
$ sudo gps_almanac --file mgaoffline.ubx
```

5.11. Program the GPS module general configuration settings

It is recommended to use the pre-generated GPS ASCII configuration file supplied found inside the tracker_config tools folder. However, it is possible to edit the configuration settings, for example, using the u-center tool.

To apply the GPS ASCII configuration to the Horizon board, enter the following command:

```
$ sudo gps_ascii_config --file ublox_gnss_configuration.dat
```

The GNSS settings are configured broadly as follows:

- Hot start GNSS
- GPS reporting once per second
- Optimised for maritime tracking applications
- All GNSS satellite systems enabled
- Reporting of TTFF and POS messages only

5.12. Program the ARTIC R2 satellite firmware

The ARTIC firmware is packaged into 3 seperate files; `ARTIC.XMEM ARTIC.YMEM ARTIC.PMEM`. These need to be combined for use on the tracker board. To do so use the following line where `--path` points to the folder that contains the aforementioned files:

```
$ sudo artic_mems_to_firmware_file --path path/to/artic/firmware/ --output Artic_firmware.bin
```

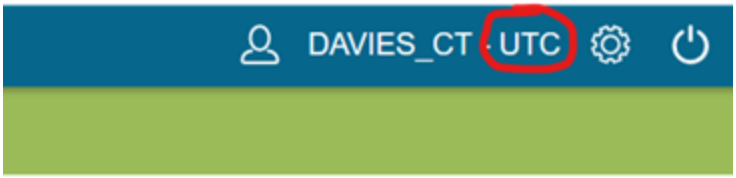
Once this file has been generated, transfer it to the tracker board with the following command:

```
$ sudo tracker_config --firmware_type ARTIC --firmware Artic_firmware.bin
```

5.13. Collect the satellite orbital information for the configuration file

To collect the latest satellite orbital information visit the Argos website at: https://argos-system.cls.fr

After signing in you **MUST** ensure that your time is set to UTC. This can be confirmed by checking the top right of the webpage as seen below



Once this is set, navigate to the "Satellite pass prediction" page, ensure "NK" and "NN" are **NOT** ticked under "Satellite choice". This is because these two satellites do not support the Argos 3 messages that we are using.

Now click "Download satellite AOP" and save the file to disk.

To convert this data to the configuration JSON format use the following line:

```
$ sudo artic_bulletin_to_json --file AOP_2019_08_19_10_53_06.txt

{"bulletin": [{"secondsSinceEpoch": 1566170997, "params": [7195.621, 98.569, 322.886, -25.341,
101.3601, 0.0], "satelliteCode": "MA"}, {"secondsSinceEpoch": 1566169463, "params": [7195.651,
98.6984, 336.302, -25.34, 101.3604, 0.0], "satelliteCode": "MB"}, {"secondsSinceEpoch": 1566167445,
"params": [7195.586, 98.7004, 345.036, -25.34, 101.3591, 0.0], "satelliteCode": "MC"},
{"secondsSinceEpoch": 1566170008, "params": [7226.502, 99.181, 268.188, -25.5, 102.0106, -1.98],
"satelliteCode": "19"}, {"secondsSinceEpoch": 1566167176, "params": [7160.249, 98.5406, 112.607,
-25.154, 100.6149, 0.0], "satelliteCode": "SR"}]}
```

5.14. Program the JSON configuration file

The system configuration file is described using JSON. Refer to the section JSON configuration format for more information.

To apply the JSON configuration file to the tracker board, enter the following commands:

```
$ sudo tracker_config --write yourThingName.json --setdatetime "`date`"
```

This will additionally set the RTC date and time to the local time of your PC.

5.15. Create an empty log file

```
$ sudo tracker_config --erase_log --create_log LINEAR
```

Upon completion of the above steps the status LED shall turn GREEN to indicate that provisioning has been successful.

5.16. Test modes

A number of test modes can be activated prior to deployment in order to check that the hardware and overall system configuration are functioning correctly. It is recommended to use the --test_mode option to enable these tests prior to unplugging the USB cable.

Below is an example command for enabling the GPS and cellular test modes:

```
$ tracker_config --test_mode GPS,CELLULAR
```

Note that the test modes will not become active until the USB cable is unplugged. The progress of the test modes can be tracked using the status LED.

5.16.1. Maximizing GPS TTFF performance

In order to maximize the TTFF performance of the GPS module, it is recommended that a complete ephemeris is obtained after the first GPS fix is made. This generally requires a certain period of time to elapse (with continual GPS lock) that is sufficient for the GPS device to obtain the ephemeris data from all satellites in view.

This can be accomplish using GPS test mode. For example, setting the `testFixHoldTime` parameter to `60` would prevent the tracker from shutting down until a period of 60 seconds of continuous GPS lock have elapsed during GPS test mode.

In order to aid the end user, the LED indicator can be used to ascertain the status of the GPS test mode process:

- LED flashing green - waiting for GPS test fix or GPS test fix lost during hold period
- LED solid green - fix found, waiting for hold period to elapse
- LED off - hold period elapsed with continual GPS lock

5.16.2. Checking IOT cellular connectivity

In order to ensure that the cellular connectivity is working properly, it is recommended to activate cellular test mode on start up. This will execute a complete status and logging update cycle.

5.16.3. Sending a satellite test message

While it can not be guaranteed that satellites will be in view of the Horizon board, it is still possible to transmit a single satellite test message using the satellite test mode. This can at least confirm that the hardware is functional.

6. Post-provisioning guide

6.1. Test modes

The test modes can also be activated using the DFU push button or via a bluetooth connection using the `--test_mode` command after provisioning over USB has completed.

6.2. GPS

The tracker shall now start logging periodically in accordance with the tracker JSON configuration file settings. GPS fixes shall be stored to the log file. The most recent GPS fix shall also be stored in RAM along with a timestamp of when the GPS fix was made.

6.3. IOT connectivity and data management

The system will attempt to make an IOT connection in accordance with the JSON configuration file settings. During each IOT connection cycle the following information shall be provided:

- Device status update
- Logging data backlog

The IoT infrastructure has been configured to organise the received data into three separate data sets which aggregate data across all deployed devices as follows:

- Device status updates - provides the last know status of each device whenever it connects. This includes:
  - Last known GPS location and timestamp of when that GPS fix was made
  - Battery level when the device last connected
  - Log file read position
- GPS location updates - provides a time series record of all GPS fixes since deployment (not just the last known location)
  - GPS location including GPS accuracy information
  - Time To First Fix (where available)
- Battery charge and/or voltage updates - provides a time series record of all battery updates, not just the last known battery level

All data sets are organized into records continaining the *thingName* that originated the data, a *timestamp* of when the data was generated and a set of fields that represent the data record itself. More information about the structure of each data set can be found in the appendices section AWS data set file format.

6.4. Extracting data using AWS tools

The `aws_config` tool provides a number of methods of obtaining data about the devices that have been registered into an AWS account instance.

6.4.1. Obtain a list of registered devices

This command may only be run by a user in the **Admin** group:

```
$ aws_config --list_things

[u'yourThingName1', u'yourThingName', u'yourThingName2']
```

6.4.2. Obtain the current device shadow status record

This command may only be run by a user in the **Admin** group:

```
$ aws_config --get_shadow thingName
{
    "metadata": {
        "desired": {
            "device_status": {
                "last_log_file_read_pos": {
                    "timestamp": 1553857919
                }
            }
        }
    },
    "state": {
        "delta": {
            "device_status": {
                "last_log_file_read_pos": 0
            }
        },
        "desired": {
            "device_status": {
                "last_log_file_read_pos": 0
            }
        }
    },
    "timestamp": 1553857975,
    "version": 2
}
```

Note that the device shadow record indicates fields that have changed under `state.delta` as well as the current value of all fields that been set under `state.desired`.

6.4.3. Obtain a list of implemented data sets

This command may only be run by a user in the **Admin** or **IOTConsumer** group:

```
$ aws_config --list_datasets
2019-03-29 11:23:34,217 credentials INFO Found credentials in shared
credentials file: ~/.aws/credentials
[u'arribada_battery_charge', u'arribada_device_status',
u'arribada_gps_location']
```

6.4.4. Download data sets

This command may only be run by a user in the **Admin** or **IOTConsumer** group:

```
$ aws_config --download_dataset arribada_gps_location
2019-03-29 11:14:48,107 aws_config INFO Retrieving data set meta data for
arribada_gps_location
2019-03-29 11:14:48,129 credentials INFO Found credentials in shared
credentials file: ~/.aws/credentials
2019-03-29 11:14:49,050 aws_config INFO Downloading data set to file
ffa31d0f-15db-4c59-9a85-c8e1a1d368b2.csv...
```

Each data set is configured to automatically update every **15 minutes** in the AWS infrastructure ie there is an effective 15 minute lag on any data set updates from tag devices. This is shortest time interval permitted for automatic updates. However, it is possible to manually force an immediate update of a data set using the `--update_dataset datasetName` option, although this may still take 1-2 minutes before a new data set is available to be downloaded.

Each data set is retrieved as a `.csv` file containing the full history of all updates to that data set. The first line inside the `.csv` file denotes the field names for that data set.

Note that all data set fields are represented as strings with their values in double-quotes; this is a quirk of the AWS IoT data set functionality but stripping of double-quotes can be easily done as part of data post-processing.

The data set file name is automatically generated by AWS infrastructure and changes whenever the data set contents change. When a data set has not changed the `.csv` file name remains the same ie the `--download_dataset` option will overwrite the previous local data set file.

6.5. AWS firmware update

This command may only be run by a user in the **Admin** group:

```
$ aws_config --firmware_update yourThingName --firmware_version
versionNumber --file DFUZipFileName
2019-03-29 11:28:03,703 aws_config INFO Using firmware file
nrf52840_xxaa.bin from DFU zip archive
2019-03-29 11:28:03,706 aws_config INFO Uploading firmware file
nrf52840_xxaa.bin to AWS...
2019-03-29 11:28:03,731 credentials INFO Found credentials in shared
credentials file: ~/.aws/credentials
2019-03-29 11:28:05,344 aws_config INFO Notifying device shadow update for
yourThingName...
{
    "metadata": {
        "desired": {
            "device_update": {
                "firmware_update": {
                    "url": {
                        "domain": {
                            "timestamp": 1553858886
                        },
                        "path": {
                            "timestamp": 1553858886
                        },
                        "port": {
                            "timestamp": 1553858886
                        }
                    },
```

```
                    "version": {
                        "timestamp": 1553858886
                    }
                }
            }
        }
    },
    "state": {
        "desired": {
            "device_update": {
                "firmware_update": {
                    "url": {
                        "domain": "arribada.s3.amazonaws.com",
                        "path":
"/f99f1395-dbef-4032-aebc-d01d21b55cf6?AWSAccessKeyId=AKIAIPZT2EDQFBIJHOWA&
Expires=1585394885&Signature=%2F%2Fabi%2BYo8%2FGd69yfB3L2Pk3MEGs%3D",
                        "port": 443
                    },
                    "version": 2
                }
            }
        }
    },
    "timestamp": 1553858886,
    "version": 3
```

```
}
```

6.6. AWS configuration update

This command may only be run by a user in the **Admin** group:

```
$ aws_config --configuration_update yourThingName --file yourThingName.json

2019-03-29 11:29:11,540 aws_config INFO Uploading configuration data length
1267 to AWS...
2019-03-29 11:29:11,566 credentials INFO Found credentials in shared
credentials file: ~/.aws/credentials
2019-03-29 11:29:12,886 aws_config INFO Notifying device shadow update for
yourThingName...
{
    "metadata": {
        "desired": {
            "device_update": {
                "configuration_update": {
                    "url": {
                        "domain": {
                            "timestamp": 1553858954
                        },
                        "path": {
                            "timestamp": 1553858954
                        },
                        "port": {
                            "timestamp": 1553858954
                        }
                    },
                    "version": {
                        "timestamp": 1553858954
                    }
                }
            }
        }
    },
    "state": {
        "desired": {
            "device_update": {
                "configuration_update": {
                    "url": {
                        "domain": "arribada.s3.amazonaws.com",
                        "path":
"/dd2eaae2-b564-415c-bd14-67a7e03a9b7b?AWSAccessKeyId=AKIAIPZT2EDQFBIJHOWA&
Expires=1585394952&Signature=eYfbXSIui5XNQ4i0L%2BpbxNOuXrw%3D",
                        "port": 443
```

```
                    },
                    "version": 1
                }
            }
        }
    },
    "timestamp": 1553858954,
    "version": 4
```

```
}
```

6.7. Obtaining the log file after a tracker board is recovered

To obtain the log file connect a USB cable from the Horizon board to a PC. Check that the LED turns solid red. Then execute the following commands:

```
$ sudo tracker_config --read_log log.bin
$ log_parse --file log.bin > log.json
```

The log file is first downloaded in raw binary format and then converted to JSON records by the `log_parse` tool. An example log file output is shown below:

```
{"GPSPosition": {"accuracyHorizontal": 12.767, "longitude": -2.1183083, "height": -0.003, "iTOW":
401079000, "latitude": 51.376391899999994, "accuracyVertical": 1.558}}
{"DateTime": {"seconds": 3, "month": 1, "hours": 1, "year": 2018, "minutes": 37, "day": 1}}
{"Pressure": {"pressure": -9.317}}
{"DateTime": {"seconds": 3, "month": 1, "hours": 1, "year": 2018, "minutes": 37, "day": 1}}
{"GPSPosition": {"accuracyHorizontal": 12.77, "longitude": -2.1183246, "height": -0.001, "iTOW":
401080000, "latitude": 51.3764012, "accuracyVertical": 1.558}}
{"DateTime": {"seconds": 4, "month": 1, "hours": 1, "year": 2018, "minutes": 37, "day": 1}}
{"GPSPosition": {"accuracyHorizontal": 12.957, "longitude": -2.118356, "height": 0.003, "iTOW":
401081000, "latitude": 51.3764021, "accuracyVertical": 1.558}}
{"DateTime": {"seconds": 5, "month": 1, "hours": 1, "year": 2018, "minutes": 37, "day": 1}}
{"GPSPosition": {"accuracyHorizontal": 13.201, "longitude": -2.1183889, "height": 0.009, "iTOW":
401082000, "latitude": 51.3764057, "accuracyVertical": 1.558}}
```

7. Appendices

7.1. Advanced combining of commands

To save time, it is possible to combine multiple commands together. For example:

```
$ sudo tracker_config --erase_log --create_log LINEAR --setdatetime "`date`" --write
configuration_v4.json --read verif.json --read_log log.bin
```

The above command will execute the steps in the following order:

- Read the current log file
- Erase the log file
- Create a new log file
- Write the new configuration file
- Set the date and time
- Read back the configuration file to `verif.json`

7.2. Bluetooth connectivity

7.2.1. Tracker configuration settings

It is first necessary to perform provisioning via USB in order to enable bluetooth connectivity. Refer to the JSON configuration section for the overall bluetooth configuration parameters. However, the following settings are typically needed:

- `bluetooth.triggerControl = [ "SCHEDULED" ]` will use `SCHEDULED` activation of the BLE backend
- `bluetooth.scheduledInterval = 0` will activate bluetooth advertising continuously
- `bluetooth.scheduledDuration = 0` will activate bluetooth advertising continuously

Note that more a energy conscious setting for `scheduledInterval` and `scheduledDuration` may be applied but there are performance

trade-offs to consider:

- The scanning duration from the PC may need to be longer to discover the device during its active advertising period
- Even if a device is successfully scanned it may subsequently be deactivated before a BLE connection attempt is made

Note that the `scheduledInterval` and `scheduledDuration` only apply when no BLE connection is established. Once a BLE connection is made, a device is no longer discoverable until the BLE connection is dropped.

Note that bluetooth connectivity is not possible whilst the USB peripheral stack has been enumerated with a host PC i.e., a USB connection always takes precedence. However, it is possible to activate a bluetooth connection when either an external USB charger is used (i.e., not connected to a PC) or with no USB cable attached (i.e., the system is running from the battery only).

7.2.2. Device addressing

The Horizon device addressing adopts the random static addressing scheme. This allows the device address to be set to any 46-bit value with the 2 MSBs forcibly set to 1. Note that the public addressing scheme requires device addresses to be assigned by the IEEE and so is not used by the current firmware.

The device address may optionally be user-configured for each tracker device via the `bluetooth.deviceAddress` setting. If no device address configuration is supplied then the default internal random static address of the nRF52 device is used instead.

7.2.3. Host PC preparation for scanning and connectivity

Firstly, confirm your bluetooth low energy adapter is available and enabled. This is required to be done every time your Linux PC is booted.

This is best achieved using `hcitool` and `hciconfig` as follows:

```
# Confirm BLE device is present and enabled

$ hcitool dev

Devices:
 hci0 F4:B7:E2:4B:4D:6C

$ hciconfig hci0 up

# Find source code distribution of arribada_tool python package

$ cd arribada-tools-x.y.z/tests
$ export HCI_DEV=0
$ sudo ./setup_ble.sh
```

Note that if you wish to use a different device to `hci0` then ensure that you substitute the correct device name in the above commands and ensure `HCI_DEV` is set to the correct device number.

7.2.4. Scanning for tracker devices

The `ble_scan` tool performs a bluetooth scan of available tracker devices in the vicinity. The system looks for any devices with the device name `Arribada_Tracker`. The following (example) output is produced by the tool:

```
Device df:3e:d9:3c:a0:ae (random), RSSI=-62 dB
Flags = 06
Complete Local Name = Arribada_Tracker
```

Each device found shall have a unique (random) device address of the form `xx:xx:xx:xx:xx:xx`. The device address will either be the Nordic factory programmed device address (if `bluetooth.deviceAddress` is not set) or the value that was programmed via `bluetooth.deviceAddress`.

The `ble_scan` tool also accepts a `--timeout` parameter which defines the number of seconds to continue scanning before reporting all discovered devices. It may be necessary to extend the timeout parameter when the advertising interval has been set to a longer duration. The default timeout is 1 second.

7.2.5. Provisioning using bluetooth

It is possible to perform provisioning and also download the log file over a bluetooth low energy connection. The procedure is identical to the USB

case with the exception that the `--ble_addr` option must now be added to the `tracker_config` tool. The device address discovered by `ble_scan` should be used with the `--ble_addr` parameter e.g., `--ble_addr df:3e:d9:3c:a0:ae`.

Also refer to the section on the `ble_auto` tool for automated batch processing of multiple tracker devices.

7.2.6. Link performance

7.2.6.1. Log file download

The table below indicates typical performance over a range of RSSI operating values for downloading a log file:

RSSI	Bytes	Time	Speed (bits per second)
-35	957433	60.6	126393.795379538
-50	957217	61.861	123789.398813469
-65	957245	63.352	120879.530243718
-72	957329	65.106	117633.274966977
-78	957357	65.464	116993.400953196
-82	957385	109.791	69760.5450355676
-90	Fail		

A typical value of RSSI is -30 dBm with < 1 cm separation between antennas. The operational link range is 10 - 15 meters based on using 1M PHY mode and 0 dBm transmit power.

7.2.6.2. GPS device programming over bluetooth

The GPS almanac loading requires 107 seconds and the GPS ASCII configuration requires 16 seconds.

7.3. Bluetooth `ble_auto` automation tool

The `ble_auto` tool is a command-line batch processing tool allowing the automation of provisioning use-cases as follows:

```
ble_auto  [ --debug ] [ --connection_retry <number_of_connection_attempts> ] [ --connection_timeout
<timeout> ] [ --scan_timeout <timeout> ] [ --black_list <name_of_black_list_file> ] [ --white_list
<name_of_white_list_file> ] [ --firmware_update_main <new_fw_file> ] [ --firmware_update_ble
<new_fw_file> ] [ --log_erase ] [ --log_create ] [ --log_skip_download ] [ --config <new_config_file>
] [ --gps_config <name_of_config_file> ] [ --gps_almanac <name_of_almanac_file> ] [ --datetime ] [
--reset ]

--debug: set the logging level to DEBUG.  Default will be INFO.
--connection_retry: set the number of connection retries.  Default is 3 attempts.
--connection_timeout: set the timeout in seconds for establishing a connection with a device.  Default
is 10.
--scan_timeout: set the scan timeout in seconds for finding new devices.  Default is 15.
--black_list: the file shall contain a newline separated list of device IDs of the form
XX:XX:XX:XX:XX:XX.  Device IDs listed in this file shall not be serviced by the tool.
--white_list: the file shall contain a newline separated list of device IDs of the form
XX:XX:XX:XX:XX:XX.  Only Device IDs listed in this file shall be serviced by the tool.
--log_skip_download: each serviced device shall not have its log file downloaded
--log_erase: each serviced device shall have its log file erased, but only if it was successfully
downloaded or --log_skip_download was specified
--log_create: each serviced device shall have a new log file created (should be used in combination
with --log_erase)
--config: each serviced device shall be programmed with a new JSON configuration file
--datetime: synchronize the tracker's RTC to the local time on the PC
--reset: the processor shall be hard reset upon completion of all other batch processing tasks
```

Each command-line option effectively represents a different task to be carried out during the servicing of discovered tracker devices. Multiple tasks can be run sequentially when servicing each device. The following command-line example will:

- Output all information logging messages to the log file called `log_file`
- Obtain status information, such as battery level, log file size, last known GPS position (always done by default and can not be disabled)
- Download the log file of each device discovered (if one exists) and convert it to JSON (always done by default)
- Apply a new JSON configuration file called `config_new.json`
- Resynchronize the date and time to the local PC

```
$ sudo ble_auto --config config_new.json --gps_almanac mgaoffline.ubx --datetime 2>&1 | tee log_file
```

For each device found, its log file shall be downloaded unless the `--log_skip_download` option is specified. The log file(s) create shall be of the form `ble_auto_ddmmyyyy_hhmmss_xxxxxxxxxxxx.bin` and `ble_auto_ddmmyyyy_hhmmss_xxxxxxxxxxxx.json` where `ddmmyyyy` is the current date, `hhmmss` is the current time and `xxxxxxxxxxxx` is the bluetooth device address in hexadecimal digits.

7.3.1. Example output

```
$ sudo ble_auto --scan_timeout 1 --log_erase --log_create --reset
2018-10-31 15:08:36,459 ble_auto INFO Scanning for new devices...
2018-10-31 15:08:37,467 ble_auto INFO Discovered device=c8:cb:86:a9:04:e2 rssi=-57
2018-10-31 15:08:37,467 ble_auto INFO Discovered device=d5:8d:8c:d3:bb:32 rssi=-59
2018-10-31 15:08:37,468 ble_auto INFO Discovered device=ee:fa:05:ba:86:8b rssi=-58
2018-10-31 15:08:37,468 ble_auto INFO Discovered device=f2:4c:32:12:f1:78 rssi=-55
2018-10-31 15:08:37,468 ble_auto INFO Connecting to device=c8:cb:86:a9:04:e2
2018-10-31 15:08:37,634 ble_auto INFO Processing tasks for device=c8:cb:86:a9:04:e2
2018-10-31 15:08:37,634 ble_auto INFO Reading status from device=c8:cb:86:a9:04:e2
2018-10-31 15:08:38,760 ble_auto INFO Status for device=c8:cb:86:a9:04:e2: id=00:00:00:00:00:00:00:00
status={'cfg_version': 4, 'ble_fw_version': 65704, 'fw_version': 4} battery={'charging_level': 2,
'charging_ind': False} log_file={u'fileSize': 1768} last_pos={u'accuracyHorizontal': 32.289,
u'seconds': 48, u'longitude': -2.3582229999999997, u'height': 0.0, u'hours': 0, u'iTOW': 313733000,
u'year': 2018, u'latitude': 51.3797328, u'accuracyVertical': 1.557, u'month': 1, u'minutes': 8,
u'day': 1}
2018-10-31 15:08:38,760 ble_auto INFO Downloading log file from device=c8:cb:86:a9:04:e2
2018-10-31 15:08:38,919 ble_auto INFO Erase log file from device=c8:cb:86:a9:04:e2
2018-10-31 15:08:39,336 ble_auto INFO Creating log file on device=c8:cb:86:a9:04:e2
2018-10-31 15:08:39,351 ble_auto INFO Resetting device=c8:cb:86:a9:04:e2
2018-10-31 15:08:39,790 ble_auto INFO Disconnecting from device=c8:cb:86:a9:04:e2
2018-10-31 15:08:39,790 ble_auto INFO Connecting to device=d5:8d:8c:d3:bb:32
2018-10-31 15:08:40,056 ble_auto INFO Processing tasks for device=d5:8d:8c:d3:bb:32
2018-10-31 15:08:40,057 ble_auto INFO Reading status from device=d5:8d:8c:d3:bb:32
2018-10-31 15:08:40,567 ble_auto INFO Status for device=d5:8d:8c:d3:bb:32: id=00:00:00:00:00:00:00:00
status={'cfg_version': 4, 'ble_fw_version': 65704, 'fw_version': 4} battery={'charging_level': 97,
'charging_ind': False} log_file={u'fileSize': 1615} last_pos={u'accuracyHorizontal': 512.313,
u'seconds': 54, u'longitude': -2.3584281, u'height': 0.0, u'hours': 5, u'iTOW': 313737000, u'year':
2018, u'latitude': 51.3795149, u'accuracyVertical': 2.036, u'month': 1, u'minutes': 8, u'day': 1}
2018-10-31 15:08:40,568 ble_auto INFO Downloading log file from device=d5:8d:8c:d3:bb:32
2018-10-31 15:08:40,695 ble_auto INFO Erase log file from device=d5:8d:8c:d3:bb:32
2018-10-31 15:08:41,106 ble_auto INFO Creating log file on device=d5:8d:8c:d3:bb:32
2018-10-31 15:08:41,121 ble_auto INFO Resetting device=d5:8d:8c:d3:bb:32
2018-10-31 15:08:41,560 ble_auto INFO Disconnecting from device=d5:8d:8c:d3:bb:32
2018-10-31 15:08:41,560 ble_auto INFO Connecting to device=ee:fa:05:ba:86:8b
2018-10-31 15:08:41,774 ble_auto INFO Processing tasks for device=ee:fa:05:ba:86:8b
2018-10-31 15:08:41,775 ble_auto INFO Reading status from device=ee:fa:05:ba:86:8b
2018-10-31 15:08:42,832 ble_auto INFO Status for device=ee:fa:05:ba:86:8b: id=00:00:00:00:00:00:00:00
status={'cfg_version': 4, 'ble_fw_version': 65704, 'fw_version': 4} battery={'charging_level': 4,
'charging_ind': False} log_file={u'fileSize': 1492} last_pos={u'accuracyHorizontal': 1587.193,
u'seconds': 52, u'longitude': -2.358267, u'height': -0.046, u'hours': 0, u'iTOW': 313738000, u'year':
2018, u'latitude': 51.3798501, u'accuracyVertical': 699.264, u'month': 1, u'minutes': 14, u'day': 1}
2018-10-31 15:08:42,832 ble_auto INFO Downloading log file from device=ee:fa:05:ba:86:8b
2018-10-31 15:08:42,946 ble_auto INFO Erase log file from device=ee:fa:05:ba:86:8b
2018-10-31 15:08:43,431 ble_auto INFO Creating log file on device=ee:fa:05:ba:86:8b
2018-10-31 15:08:43,446 ble_auto INFO Resetting device=ee:fa:05:ba:86:8b
2018-10-31 15:08:43,885 ble_auto INFO Disconnecting from device=ee:fa:05:ba:86:8b
2018-10-31 15:08:43,885 ble_auto INFO Connecting to device=f2:4c:32:12:f1:78
2018-10-31 15:08:44,069 ble_auto INFO Processing tasks for device=f2:4c:32:12:f1:78
2018-10-31 15:08:44,070 ble_auto INFO Reading status from device=f2:4c:32:12:f1:78
2018-10-31 15:08:44,542 ble_auto INFO Status for device=f2:4c:32:12:f1:78: id=00:00:00:00:00:00:00:00
status={'cfg_version': 4, 'ble_fw_version': 65704, 'fw_version': 4} battery={'charging_level': 60,
'charging_ind': False} log_file={u'fileSize': 2167} last_pos={u'accuracyHorizontal': 67.019,
u'seconds': 41, u'longitude': -2.3581231, u'height': 0.009, u'hours': 3, u'iTOW': 313741000, u'year':
2018, u'latitude': 51.3795406, u'accuracyVertical': 1.754, u'month': 1, u'minutes': 16, u'day': 1}
2018-10-31 15:08:44,542 ble_auto INFO Downloading log file from device=f2:4c:32:12:f1:78
2018-10-31 15:08:44,707 ble_auto INFO Erase log file from device=f2:4c:32:12:f1:78
2018-10-31 15:08:45,149 ble_auto INFO Creating log file on device=f2:4c:32:12:f1:78
2018-10-31 15:08:45,164 ble_auto INFO Resetting device=f2:4c:32:12:f1:78
2018-10-31 15:08:45,602 ble_auto INFO Disconnecting from device=f2:4c:32:12:f1:78
2018-10-31 15:08:45,602 ble_auto INFO Scanning for new devices...
2018-10-31 15:08:46,655 ble_auto INFO No new devices discovered
2018-10-31 15:08:46,655 ble_auto INFO Post-processing devices...
2018-10-31 15:08:46,655 ble_auto INFO Converting log file binary to JSON for device=c8:cb:86:a9:04:e2
2018-10-31 15:08:46,668 ble_auto INFO Converting log file binary to JSON for device=d5:8d:8c:d3:bb:32
2018-10-31 15:08:46,677 ble_auto INFO Converting log file binary to JSON for device=ee:fa:05:ba:86:8b
2018-10-31 15:08:46,684 ble_auto INFO Converting log file binary to JSON for device=f2:4c:32:12:f1:78
```

7.4. JSON logging format

The `log_parse` tool shall generate log output using the following JSON format:

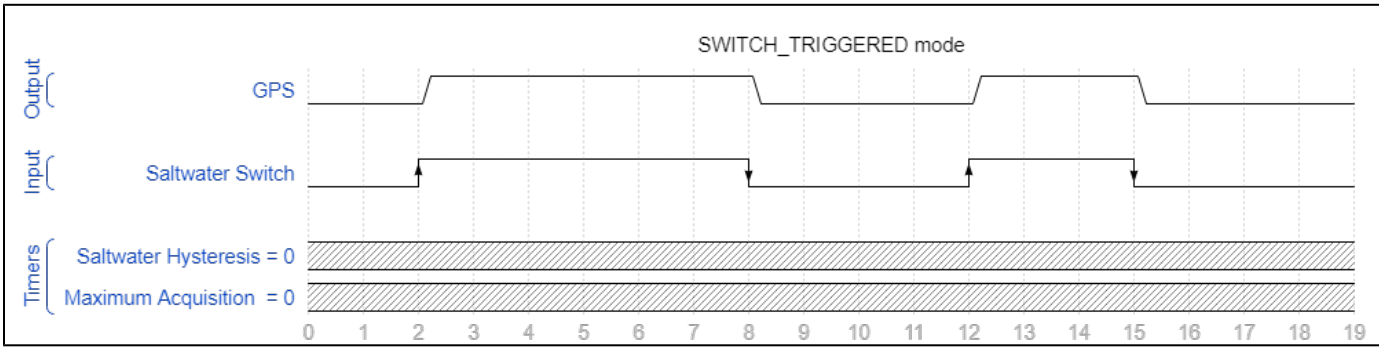
```
{
 "DateTime": {
  "seconds": number,
  "month": number,
  "hours": number,
  "year": number,
  "minutes": number,
  "day": number
 },
 "Timestamp": {
  "timestamp": number
 },
 "TimeToFirstFix": {
  "ttff": number    # Time in milliseconds to acquire first GPS fix
 },
 "GPSPosition": {
        "accuracyHorizontal": number, # Estimated horizontal accuracy in metres
        "accuracyVertical": number,   # Estimated vertical accuracy in metres
  "latitude": number,  # Latitude degrees
  "iTOW": number,   # iTOW time in seconds
  "longitude": number, # Longitude degrees
  "height": number  # Height in metres
 },
 "Surfaced": {   # Salt water switch "surfaced" detection event
 },
 "Submerged": {          # Salt water switch "submerged" detection event
 },
 "BatteryCharge": {
  "charge": number # Charge level indicator 0..100
 },
 "BatteryVoltage": {
  "voltage": number # Battery voltage reading from ADC
 },
 "BluetoothEnable": {
  "cause": number
 },
 "BluetoothDisable": {
  "cause": number
 },
 "BluetoothConnected": {
 },
 "BluetoothDisconnected": {
  "cause": number
 },
 "GPSOn": {    # Logged whenever the GPS device is woken up
 },
 "GPSOff": {    # Logged whenever the GPS device is shutdown
 },
 "Startup": {
  "cause": string  # Hex string of RCC.CSR register value on ARM processor
 },
 "SoftWatchdog": {
  "watchdogAddress": string # Hex address of PC before the soft WDOG triggered
 },
```

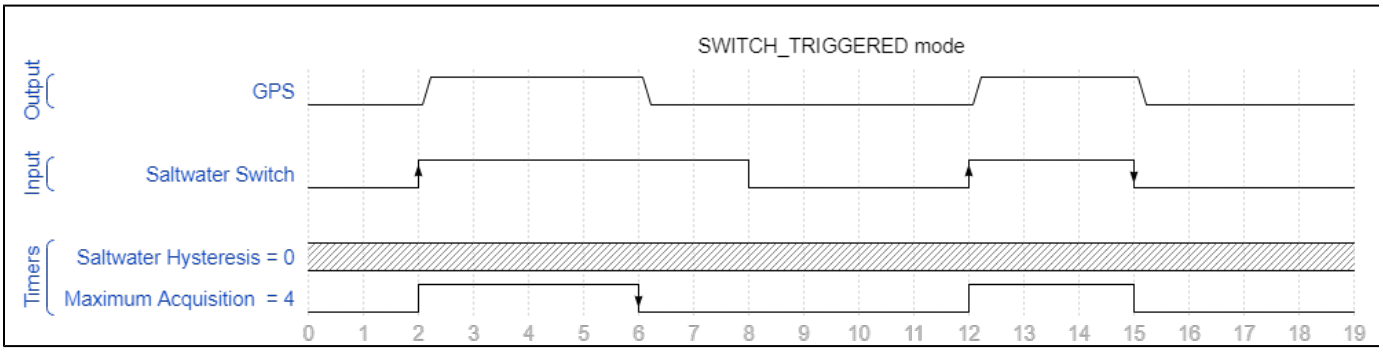
```
 "IOTConfigUpdate": {   # Notification an IOT configuration file update was received
  "version": number,
  "file_length": number
 },
 "IOTFirmwareUpdate": {   # Notification an IOT firmware file update was received
  "version": number,
  "file_length": number
 },
 "IOTErrorCode": {       # Notification of error code when an IOT error occurs
  "error_code": number
 },
 "IOTStatus": {
  "status": string,  # Indication of current IOT connectivity status
 }
}
```

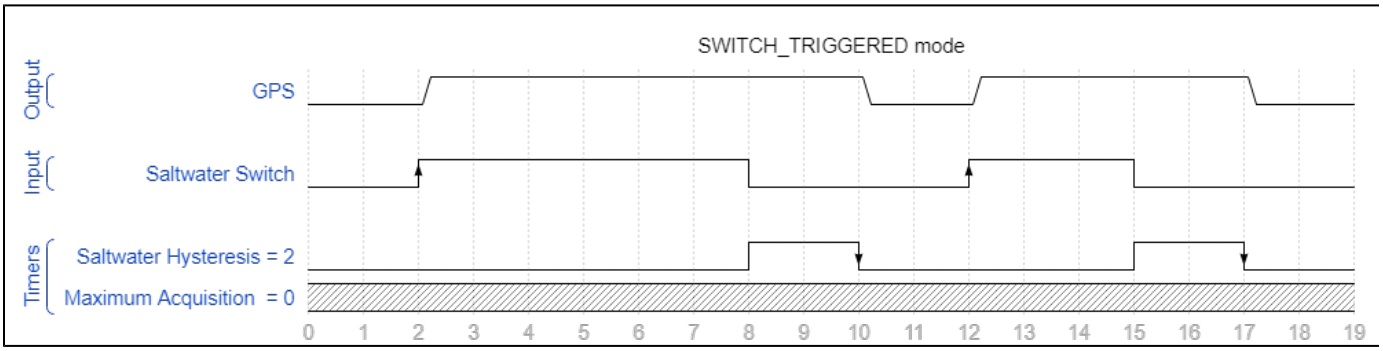
7.5. GPS Trigger Modes

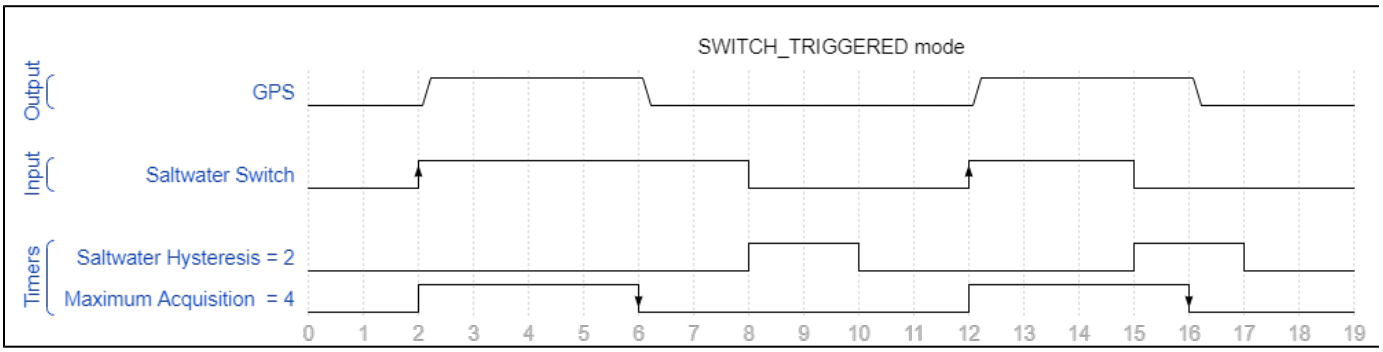
The following diagrams describe the different GPS trigger options. Any arrows on rising edges is the cause for the GPS to turn on, any arrow on a falling edge is the cause for the GPS to turn off

7.5.1. SWITCH_TRIGGERED

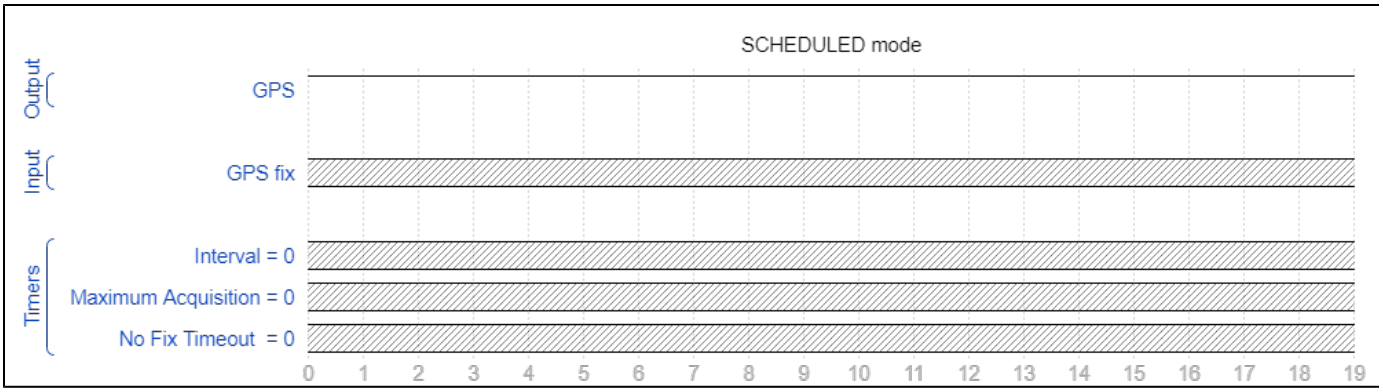


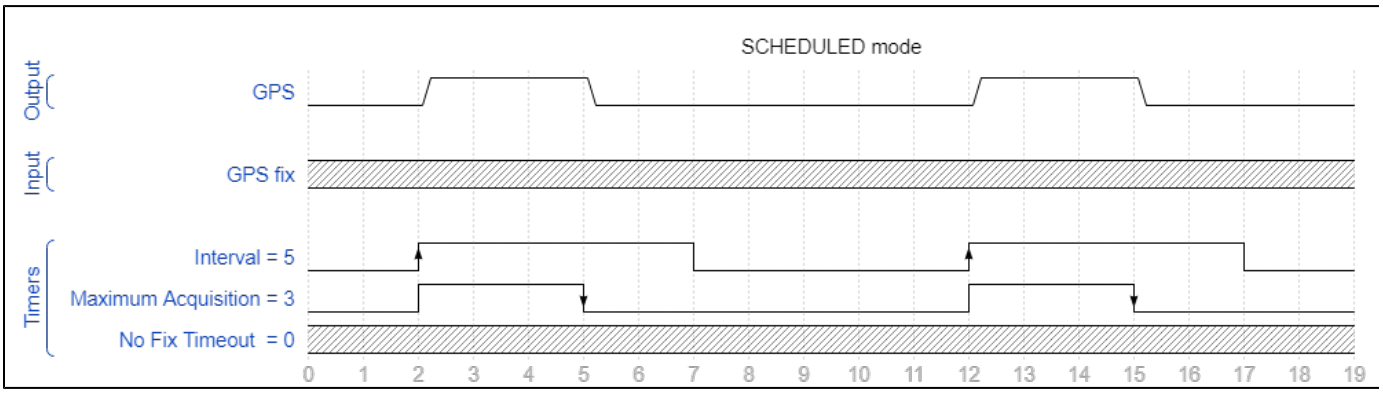


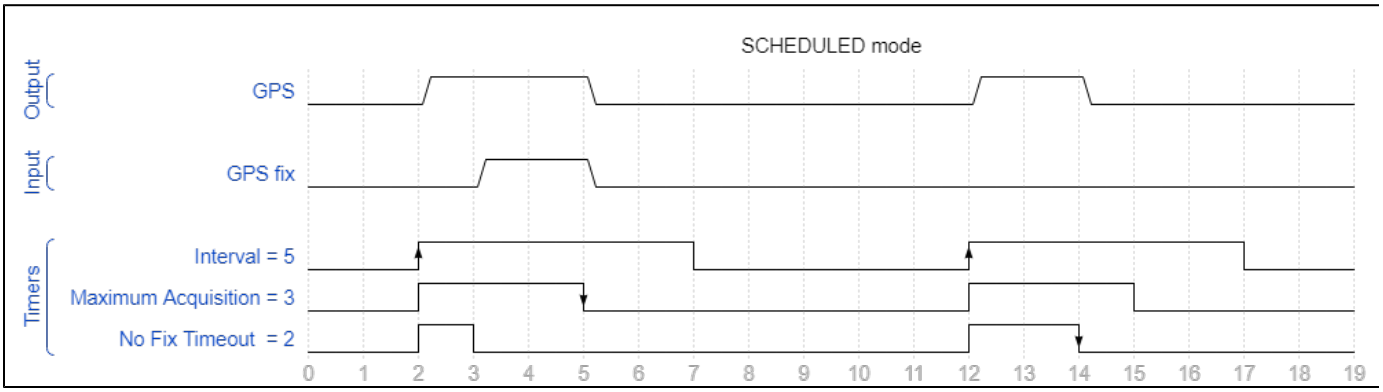




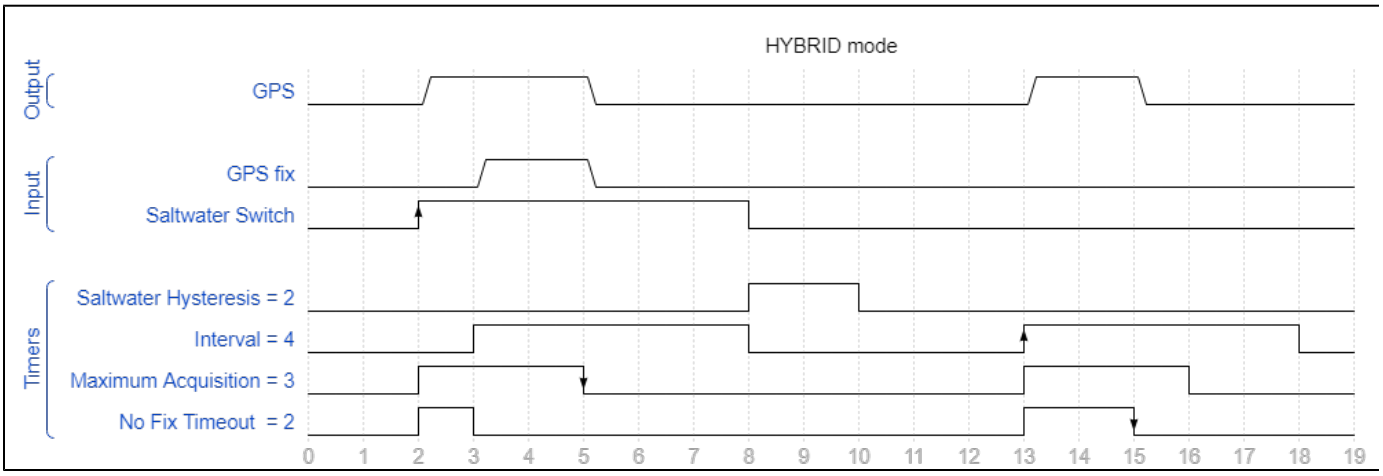
7.5.2. SCHEDULED


SCHEDULED mode
Output
GPS
Input
GPS fix
Timers
Interval = 0
Maximum Acquisition = 0
No Fix Timeout = 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19


SCHEDULED mode
Output
GPS
Input
GPS fix
Timers
Interval = 5
Maximum Acquisition = 3
No Fix Timeout = 0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19


SCHEDULED mode
Output
GPS
Input
GPS fix
Timers
Interval = 5
Maximum Acquisition = 3
No Fix Timeout = 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

7.5.3. HYBRID


HYBRID mode
Output
GPS
Input
GPS fix
Saltwater Switch
Timers
Saltwater Hysteresis = 2
Interval = 4
Maximum Acquisition = 3
No Fix Timeout = 2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

7.6. JSON configuration format

The raw JSON configuration file format is shown below for the v5 configuration file format.

```
{
    "version": number,
 "system": {
  "deviceName": string
 },
 "gps": {
  "lastKnownPosition": {   # Last known GPS location recorded with date and time (read-only)
   "accuracyHorizontal": number,
   "accuracyVertical": number,
   "height": number,
   "iTOW": number,
   "latitude": number,
   "longitude": number,
   "day": number,
   "month": number,
   "year": number,
   "hours": number,
   "minutes": number,
   "seconds": number,
  },
  "logPositionEnable": boolean, # GPS position logging enable
  "logTTFFEnable": boolean,  # GPS TTFF logging enable
  "logDebugEnable": boolean,  # GPS debug event logging
  "maximumAcquisitionTime": number, # Maximum GPS acqusition on-time (0=>indefinite)
  "scheduledAcquisitionInterval": number, # GPS scheduling period (0=>continuous)
  "scheduledAcquisitionNoFixTimeout": number, # GPS no-fix timeout period (0=>indefinite)
  "testFixHoldTime": number,      # Hold period after test GPS fix before shutting down (0=>disabled)
        "maxFixes": number,             # Maximum number of fixes to allow in the acquisition period
(0=>unlimited, 1=>1 fix, etc)
                                  # NOTE: This field has no effect when processing the test fix
hold time.
  "mode": string                  # One of { "SWITCH_TRIGGERED", "SCHEDULED", "HYBRID", "ONE_SHOT" }
 },
 "rtc": {
  "syncToGPS": boolean,
  "dateTime": string         # Should accept any reasonable time string e.g., "Wed, 21 Feb 2018
16:17:13 GMT"
 },
 "logging": {
  "enable": boolean,      # Global logging enable/disable
  "bytesWritten": number,           # This attribute is read only
  "fileSize": number,               # This attribute is read only
  "fileType": string,               # This attribute is read only (LINEAR or CIRCULAR)
  "groupSensorReadingsEnable": boolean, # Try to group log entries with a single timestamp entry
  "syncFramingEnable": boolean,     # This attribute is read only
  "hrtEnable": boolean,     # Reserved for future use
  "dateTimeEnable": boolean,        # Should accept any reasonable time string e.g., "Wed, 21 Feb 2018
16:17:13 GMT"
 },
 "saltwaterSwitch": {
  "logEnable": boolean,   # Log enable/disable of saltwater switch submerged and surfaced event
transitions
  "hysteresisPeriod": number  # Required settling period in seconds for debouncing switch closed
events (0=>no debouncing)
 },
 "bluetooth": {
  "deviceAddress": string,        # Format shall be "xx:xx:xx:xx:xx:xx" and the upper 2 most MSBs must
be set to 1.
  "triggerControl": [ string ],   # Permitted options are "REED_SWITCH", "SCHEDULED", "GEOFENCE",
"ONE_SHOT"
  "scheduledInterval": number, # Scheduling period in seconds; 0=>CONTINUOUS
  "scheduledDuration": number, # Scheduled duration in seconds; 0=>INDEFINITE
  "advertisingInterval": number, # Expressed in units of 0.625 ms
  "connectionInterval": number,  # Expressed in units of 1.25 ms
```

```
  "connectionInactivityTimeout": number, # Force connection to drop after inactivity period in seconds
  "phyMode": string,    # One of { "1_MBPS", "2_MBPS" }
        "advertisingTags": [ string ],  # Permitted options are "LAST_GPS_TIME", "BATTERY_LEVEL", etc
  "logEnable": boolean   # Enable/disable logging of BLUETOOTH_xxx log messages
 },
 "accelerometer": {
  "logEnable": boolean,   # Enable accelerometer logging
  "config": number,    # Reserved for future use
  "highThreshold": number,  # Vector magnitude sum high threshold (when TRIGGERED)
  "sampleRate:" number,   # Sample rate (when PERIODIC)
  "mode": string,                # One of { "PERIODIC", "TRIGGERED" }
  "scheduledAcquisitionInterval": number, # The repetition period in seconds at which acquisitions are
started for PERIODIC mode, 0=>continuously
  "maximumAcquisitionTime": number  # The period in seconds over which acquisitions are made for
PERIODIC mode, 0=>indefinitely
 },
 "pressure_sensor": {
  "logEnable": boolean,   # Enable pressure sensor logging
  "sampleRate": number,   # Sampling rate for PERIODIC or TRIGGERED modes
  "highThreshold": number,  # High threshold (TRIGGERED_BETWEEN or TRIGGERED_ABOVE)
  "lowThreshold": number   # Low threshold (TRIGGERED_BELOW, TRIGGERED_BETWEEN)
  "mode": string,                # One of { "PERIODIC", "TRIGGERED" }
  "scheduledAcquisitionInterval": number, # The repetition period in seconds at which acquisitions are
started for PERIODIC mode, 0=>continuously
  "maximumAcquisitionTime": number  # The period in seconds over which acquisitions are made for
PERIODIC mode, 0=>indefinitely
 },
    "iot": {
        "enable": boolean,             # Enable/disable IoT function
        "logEnable": boolean,          # Info/debug logging enable for all IoT functions
        "minBatteryThreshold": number,  # Don't activate IoT unless min. battery level threshold is
met
        "cellular": {
            "enable": boolean,             # Enable/disable the cellular module
            "connectionPriority": number,   # 0=>highest, 10=>lowest; for handling radio access
contentions (default 0)
            "connectionMode": string,       # "2G, "3G", "AUTO"
            "logFilter": [ string ],        # List of logging tag names for sending of outgoing
logging data
            "statusFilter": [ string ],     # List of status fields to send in a status update
            "maxBackoffInterval": number,   # If no cellular coverage is found, the system will use an
exponential back-off
                                            # timer before performing a cell search again.  The
exponential back-off timer
                                            # will be capped to this value (seconds).  This timer
overrides the "maxInterval"
                                            # timer.
            "gpsScheduleIntervalOnMaxBackoff": number,  # The IoT function can modify the GPS
scheduling interval
                                                        # if the cellular max backoff interval has
occurred (seconds)
                                                        # This allows a more power efficient mode to
be activated by
                                                        # increasing the GPS schedule interval when
cellular coverage is lost
            "minInterval": number,          # Don't try to make a connection if any other radio
technology
                                            # has transmitted within the prior interval (seconds).
            "maxInterval": number,          # Max. allowed interval in seconds between updates
                                            # This overrides "minUpdates" and forces a connection but
only once the "maxBackoffInterval"
                                            # timer has deactivated.
            "minUpdates": number,           # Min. number of (eg GPS) updates to receive before trying
to connect
                                            # 1=>at least 1 update, 2=>at least 2 updates, etc
                                            # 0=>not permitted
            "checkFirmwareUpdates": boolean,        # Allow device to check for firmware updates via
IoT cloud
            "checkConfigurationUpdates": boolean,   # Allow device to check for configuration updates
via IoT cloud
```

```
            "aws": {
                    "arn": string,                  # The value of AWS $arn ie same as the HTTPS
domain name
                    "port": number,                 # HTTPS port number for AWS
                    "thingName": string,            # Unique thingName assigned to this device in AWS
                                                    # If not specified use "system.deviceName"
                    "loggingTopicPath": string,     # eg "/topics/#/logging" where # represents the
thingName and shall be substituted
                    "deviceShadowPath": string,     # eg "/things/#/shadow" where # represents the
thingName and shall be substituted
            }
        },
        "satellite": {
            "enable": boolean,                  # Satellite module enable/disable
            "connectionPriority": number,       # 0=>highest, 10=>lowest; for handling radio access
contentions
            "statusFilter": [ string ],         # List of status fields to pack into each transmit
message
            "minUpdates": number,               # Min. number of updates before transmitting
                                                # 1=>at least 1 update, 2=>at least 2 updates, etc
                                                # 0=>not permitted
            "minInterval": number,              # Don't try to send a message if any other radio
technology
                                                # has transmitted within the prior interval (seconds).
            "maxInterval": number,              # Max. allowed interval in seconds between updates
                                                # This overrides "minUpdates" setting and forces a
transmission
                                                # as soon as the next satellite window arrives
            "randomizedTxWindow": number        # Number of seconds +/- about predicted transmission
window
                                                # to randomise eg 30 means randomise within +/- 30
seconds
            "artic": {
                "deviceIdentifier": string,     # Artic specific configuration for device identifier
(typically 7 hex bytes)
                "bulletin": [ string ]          # List of bulletin string entries, one per satellite
            }
```

```
        }
    }
}
```

Note that the above JSON configuration record format is subject to change in future releases.

7.7. Tool command-line arguments

7.7.1. aws_config

```
usage: aws_config [-h] [--version] [--debug] [--cert_path CERT_PATH]
                  [--namespace NAMESPACE] [--install] [--uninstall]
[--upgrade]
                  [--region REGION] [--get_iot_endpoint]
                  [--register_thing THING_NAME]
                  [--unregister_thing THING_NAME] [--list_things]
                  [--get_shadow THING_NAME] [--set_shadow THING_NAME]
                  [--delete_shadow THING_NAME] [--create_shadow THING_NAME]
                  [--update_status THING_NAME]
                  [--send_logging THING_NAME]
                  [--update_dataset DATASET_NAME]
                  [--download_dataset DATASET_NAME] [--list_datasets]
                  [--data DATA] [--firmware_update THING_NAME]
                  [--firmware_version FIRMWARE_VERSION]
                  [--configuration_update THING_NAME] [--file FILE]

optional arguments:
  -h, --help                            Show this help message and exit
  --version                             Display application version number
  --debug                               Turn on debug trace level
  --cert_path CERT_PATH                 Path for where all certificates and
keys shall be stored (default is ./certificates)
  --namespace NAMESPACE                 Namespace for creating IoT objects
(default is arribada)
  --install                             (admin) Installation setup of IoT
infrastructure (need only be done once)
  --uninstall                           (admin) Uninstall a previously
installed IoT infrastructure (WARNING: will delete all data!!!)
  --upgrade                             (admin) Upgrade an existing IoT
infrastructure without deleting any data or registered things
  --region REGION                       AWS region name (default is
us-west-2)
  --get_iot_endpoint                    (admin) Display the domain name of
the AWS IoT endpoint
  --register_thing THING_NAME           (admin) Create certificate and key
and register new thing as THING_NAME
  --unregister_thing THING_NAME         (admin) Unregister thing called
```

```
THING_NAME
  --list_things                          (admin) Return a list of registered
things by their THING_NAME
  --get_shadow THING_NAME                (admin) Obtain the JSON shadow
record for THING_NAME
  --set_shadow THING_NAME                (admin) Set the JSON shadow record
for THING_NAME; JSON is passed in --data option.
  --delete_shadow THING_NAME             (admin) Delete the JSON shadow
record for THING_NAME
  --create_shadow THING_NAME             (admin) Create a previously deleted
default JSON shadow record for THING_NAME
  --update_status THING_NAME             (admin) Provide JSON for
"device_status" field for THING_NAME; JSON is passed in --data option.
  --send_logging THING_NAME              (admin) Send a logging .bin file to
THING_NAME; file is passed in --file option.
  --download_dataset DATASET_NAME        (any) Download a .csv file for the
given dataset name on the AWS IoTAnalytics server
  --update_dataset DATASET_NAME          (admin) Force an update of the
given dataset name on the AWS IoTAnalytics server
  --list_datasets                        (any) Obtain a list of available
data sets on the AWS IoTAnalytics server
  --data DATA                           Use @filename to pass JSON from a
file, otherwise the data is passed raw
  --firmware_update THING_NAME           (admin) Perform firmware update on
THING_NAME; DFU zip is passed as --file with --firmware_version
  --firmware_version FIRMWARE_VERSION    (admin) Mandatory when performing
--firmware_update
  --configuration_update THING_NAME      (admin) Perform configuration
```

```
update on THING_NAME
  --file FILE                              Provides a file name to support
other operations
```

7.7.2. tracker_config

```
usage: tracker_config [-h] [--ble_addr BLUETOOTH_ADDR] [--debug]
                      [--write WRITE] [--read READ] [--read_log READ_LOG]
                      [--battery] [--status] [--erase]
                      [--erase_log] [--create_log CREATE_LOG] [--reset
RESET]
                      [--test_mode TEST_MODE] [--protect] [--unprotect]
                      [--setdatetime SETDATETIME] [--getdatetime]
                      [--firmware_type FIRMWARE_TYPE] [--firmware FIRMWARE]
                      [--version]

optional arguments:
  -h, --help                    show this help message and exit
  --ble_addr BLUETOOTH_ADDR     use bluetooth as communications backend
connecting to device xx:xx:xx:xx:xx:xx
  --debug                       enable debug trace
  --write FILENAME              write the JSON configuration file to RAM
storage on the tracker
  --read FILENAME               read the current RAM storage on the tracker
the specified JSON file
  --read_log FILENAME           read the current log file (binary format)
to the specified filename
  --battery                     read the current battery level and display
  --status                      read the current device status and display
  --erase                       erase the current RAM configuration to
defaults
  --erase_log                   erase the current log file in flash memory
  --create_log LOG_TYPE         create a log file of the specified type
(LINEAR, CIRCULAR)
  --reset RESET_TYPE            reset the system with the specified type
(CPU, FLASH)
  --test_mode TEST_MODE         apply test mode flash as a comma separated
list of items (allowed: GPS, CELLLAR, SATELLITE)
  --protect                     protect the current configuration file from
changes
  --unprotect                   unprotect the current configuration file
from changes
  --setdatetime DATETIME        set the current RTC date and time
  --getdatetime                 get the current RTC data and time and
display
  --firmware_type FIRMWARE_TYPE upload a new firmware file of the specified
type (allowed: ARTIC)
  --firmware FILENAME           firmware update filename
  --version                     display this application's version number
```

7.7.3. cellular_config

```
usage: cellular_config [-h] [--serial SERIAL] [--baud BAUD]
                       [--ble_addr BLUETOOTH_ADDR] --root_ca ROOT_CA --cert
                       CERT --key KEY [--debug] [--verify]

--serial               local serial port for bridging via local serial port
(eg /dev/ttyUSB0)
--baud                 set the serial port communications baud rate when
using local serial port (default 115200)
--ble_addr             bluetooth device address for bridging via BLE (ie
xx:xx:xx:xx:xx:xx)
--root_ca filename     (mandatory) filename of the CA certificate to store
in secure area on the device
--cert filename        (mandatory) filename of the device certificate to
store in secure area on the device
--key filename         (mandatory) filename of the device private key to
store in secure area on the device
--debug                enable debug level trace (for AT command debugging)
--verify      check if all the device
```

7.7.4. gps_almanac

```
usage: gps_almanac [-h] [--serial SERIAL] [--baud BAUD]
                   [--ble_addr BLUETOOTH_ADDR] --file FILE [--debug]

--serial               local serial port for bridging via local serial port
(eg /dev/ttyUSB0)
--baud                 set the serial port communications baud rate when
using local serial port (default 115200)
--ble_addr             bluetooth device address for bridging via BLE (ie
xx:xx:xx:xx:xx:xx)
--file FILENAME        the mgaoffline.ubx filename to send to the u-blox
M8N device
```

7.7.5. gps_ascii_config

```
usage: gps_ascii_config [-h] [--serial SERIAL] [--baud BAUD]
                        [--ble_addr BLUETOOTH_ADDR] --file FILE [--debug]

--serial               local serial port for bridging via local serial port
(eg /dev/ttyUSB0)
--baud                 set the serial port communications baud rate when
using local serial port (default 115200)
--ble_addr             bluetooth device address for bridging via BLE (ie
xx:xx:xx:xx:xx:xx)
--file FILENAME        the ublox_gnss_configuration.dat filename to send to
the u-blox M8N device
```

7.8. AWS data set file formats

All data set files shall be provided in `.csv` format. The first line of each file shall contain the field names and subsequent lines of the `.csv` file shall contain the data records for a given data set.

Note that data sets shall grow incrementally over time i.e., historical data is not removed even if the historical data has already been downloaded.

7.8.1. Device status data set

Field Name	Presence	Type	Notes	Status Filter Enable Tag
"__dt"	Mandatory	String	Delta time since previous logged item	-
"thing_name"	Mandatory	String	The AWS registered thing name associated with the data record	-
"timestamp"	Mandatory	Integer	Number of seconds since the epoch	-
"last_log_file_read_pos"	Optional	Integer	The last known read position from the log file stored on the tracker device. The information persists across multiple connections and is updated only by the tracker device.	"LAST_LOG_READ_POS"
"last_gps_fix_longitude"	Optional	Float	GPS longitude expressed as floating point.	"LAST_GPS_LOCATION"
"last_gps_fix_latitude"	Optional	Float	GPS latitude expressed as floating point.	"LAST_GPS_LOCATION"
"last_gps_fix_time"	Optional	Integer	Number of seconds since the epoch	"LAST_GPS_LOCATION"
"last_cellular_connection"	Optional	Integer	Number of seconds since the epoch	"LAST_CELLULAR_CONNECT"
"last_sat_tx"	Optional	Integer	Number of seconds since the epoch	"LAST_SAT_TX"
"next_sat_tx"	Optional	Integer	Number of seconds since the epoch	"NEXT_SAT_TX"
"battery_level"	Optional	Integer	Battery charge level 0..100	"BATTERY_LEVEL"
"battery_voltage"	Optional	Integer	Battery voltage ADC reading.	"BATTERY_VOLTAGE"
"config_version"	Optional	Integer	Current configuration file version used by tracker device	"CONFIG_VERSION"
"fw_version"	Optional	Integer	Current firmware file version used by tracker device	"FW_VERSION"

7.8.2. GPS location data set

Field Name	Presence	Type	Notes	Logging Filter Enable Tag
"__dt"	Mandatory	String	Delta time since previous logged item	-
"thing_name"	Mandatory	String	The AWS registered thing name associated with the data record	-
"timestamp"	Mandatory	Integer	Number of seconds since the epoch	Derived from either "TIMESTAMP" or "DATETIME". In either case it shall be converted to a "timestamp" value by the AWS lambda function processing.
"longitude"	Mandatory	Float	GPS longitude co-ordinate	"GPS"
"latitude"	Mandatory	Float	GPS latitude co-ordinate	"GPS"
"height"	Mandatory	Float	GPS height in metres	"GPS"
"h_acc"	Mandatory	Float	GPS horizontal accuracy in metres	"GPS"
"v_acc"	Mandatory	Float	GPS vertical accuracy in metres	"GPS"
"ttff"	Optional	Integer	Time To First Fix in seconds	"GPS"

7.8.3. Battery data set

Field Name	Presence	Type	Notes	Logging Filter Enable Tag
"__dt"	Mandatory	String	Delta time since previous logged item	-
"thing_name"	Mandatory	String	The AWS registered thing name associated with the data record	-
"timestamp"	Mandatory	Integer	Number of seconds since the epoch	-
"battery_level"	Optional	Integer		"BATTERY_LEVEL"
"battery_voltage"	Optional	Integer		"BATTERY_VOLTAGE"

7.9. AWS device shadow JSON record format

```
{
 "device_update": {
  "configuration_update": { "url": { "domain": string, "port": number,
"path": string }, "version": number },
  "firmware_update": { "url": { "domain": string, "port": number, "path":
string }, "version": number },
 },
 "device_status": {
  "last_gps_location": { "longitude": float, "latitude": float,
"timestamp": number },
  "last_cellular_connected_timestamp": number, # UTC time of when tracker
connected as timestamp
  "last_sat_tx_timestamp": number, # Last satellite transmission (prepass)
timestamp
  "next_sat_tx_timestamp": number, # Computation of next prepass window
timestamp
  "battery_level": number,   # Last battery level 0..100
  "battery_voltage": number,   # Last battery voltage (from ADC)
  "configuration_version": number, # Last configuration version 0..N
  "firmware_version": number,   # Last firmware version 0..N
  "last_log_file_read_pos": number, # Log file read seek position offset
 }
}
```

Note that the `"device_update"` field is manipulated by the `aws_config` tool via the `--configuraton_update` and `--firmware_update` options. It should not be set directly.

The `"device_status"` field may be set using the `--update_status` option to the `aws_config` tool. It shall also be updated whenever the tracker board connects to the IOT service.

Note that when a tracker board is first provisioned, only the `last_log_file_read_pos` parameter shall be present with an initial value of `0`.