energy-balance-1-software

Elle Smith

Sep 03, 2021

CONTENTS:

This software is intended to be used for logging data from an NCAS AMOF energy balance tower via an Campbell Scientific CR3000 logger using python. The software is intended to be used for soil and radiation data products, and where data is referred to, it is data relating to these data products. It may also work with with CR1000 or other Campbell loggers but this has not been tested.

There are various components as explained briefly below:

- The software provides capabilities for streaming data from the Campbell logger, downloading data from specific dates and producing a csv file containing this data for each day.
- There is functionality to put the data from these daily csv files into a MySQL database, to allow for visualization e.g. with Grafana.
- There is a script to create daily or monthly netCDF files that conform to the NCAS-GENERAL Data Standard from these daily csv files. As part of this process, quality control is carried out on the variables, and quality control variables are added to the netCDF datasets.
- It is also possible to create csv files for soil and radiation that have had quality control applied, to allow you to view the data manually or plot via the plotting script provided.
- A plotting script is provided to plot specified csv columns against time, provided a timestamp exists in the csv file.

With this software, the process of collecting data, applying quality control, plotting and creating netCDF files should be much more straightforward.

Below, images of the sensors in use:

CHAPTER

ONE

QUICK GUIDE

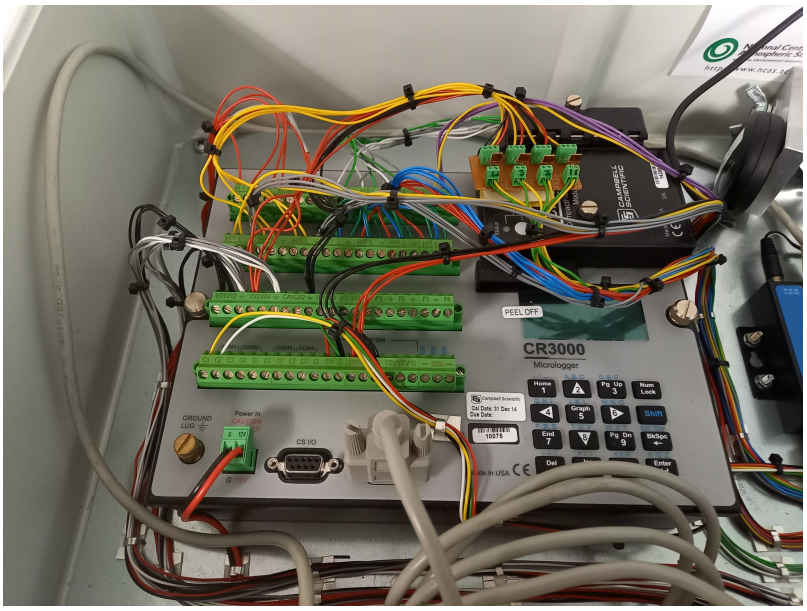
The full documentation can be found at https://ncas-energy-balance-1-software.readthedocs.io/en/latest/

- This repository contains scripts for logging data from an NCAS AMOF energy balance via an Campbell Scientific CR3000 logger using python.
- The logger should be configured to generate data tables for the data it collects.
- Information about getting started with the logger can be found on the Campbell Scientific website (https://www.campbellsci.co.uk/cr3000). See in particular the 'Getting Started Guide' and the Videos & Tutorials.
- The names of these data tables will be required to use this software and get the data from the logger.
- This software replaces the need to use Microsoft Windows software for acquiring the data.

Before attempting to install this software, ensure you have Python 3.7 and git installed. (What is Git: https://github.com/git-guides/) (Installing Git: https://github.com/git-guides/install-git)

The software is intended to be used for soil and radiation data products, and where data is referred to, it is data relating to these data products. It may also work with with CR1000 or other Campbell loggers but this has not been tested.

The CR3000 logger:



There are various components as explained briefly below:

- The software provides capabilities for streaming data from the Campbell logger, downloading data from specific dates and producing a csv file containing this data for each day.

- There is functionality to put the data from these daily csv files into a MySQL database, to allow for visualization e.g. with Grafana.
- There is a script to create daily or monthly netCDF files that conform to the NCAS-GENERAL Data Standard from these daily csv files. As part of this process quality control is carried out on the variables, and quality control variables are added to the netCDF datasets.
- It is also possible to create csv files for soil and radiation that have had quality control applied, to allow you to view the data manually or plot via the plotting script provided.
- A plotting script is provided to plot specified csv columns against time, provided a timestamp exists in the csv file.

With this software the process of collecting data, applying quality control, plotting and creating netCDF files should be much more straightforward.

To see which scripts are available and what they do, go to scripts.

1.1 Install

All scripts can be downloaded from the Github repo. In order to do this, follow the steps below. An explanation of using `git clone` can be found here: https://github.com/git-guides/git-clone

```
$ git clone https://github.com/ncasuk/ncas-energy-balance-1-software.git
$ cd ncas-energy-balance-1-software
```

In order to use the scripts, create a virtual environment and install the package within the ncas-energy-balance-1-software repository: The package is called `energy_balance` and contains the scripts and all you will need to use them.

```
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install .
```

If you would like to be able to edit the scripts, change the final line to

```
$ pip install -e .
```

Creating a virtual environment allows you to manage the installation requirements. You will need to activate the virtual environment before using the scripts (`source venv/bin/activate`) after opening a new terminal window.

If running script 7 - plot_csv.py, you will need to install matplotlib. On Raspberry Pi, run

```
$ sudo apt-get install python3-matplotlib
```

Otherwise,

```
$ pip install matplotlib
```

It is likely that you will use a Raspberry Pi to connect to the logger using a Moxa NPort in Real COM mode. Therefore you will run the scripts on the Raspberry Pi.

- If using Real COM mode the Real COM driver must be installed (see https://www.moxa.com/en/products/industrial-edge-connectivity/serial-device-servers/general-device-servers/nport-5100-series#resources) along with raspberrypi-kernel-headers (run `sudo apt install raspberrypi-kernel-headers`)

- You will then need to map the target IP to the Real COM port - find the readme explaining how to do this at the path: `/usr/lib/npreal2`.
- You may then have to change the group and permission of the port:

```
$ sudo chown root:dialout /dev/ttyr0
$ sudo chmod 660 /dev/ttyr0
```

Sometimes there can be a problem installing numpy on Raspberry Pi. The troubleshooting page for this is: https://numpy.org/devdocs/user/troubleshooting-importerror.html

The command:

```
$ sudo apt-get install libatlas-base-dev
```

usually works.

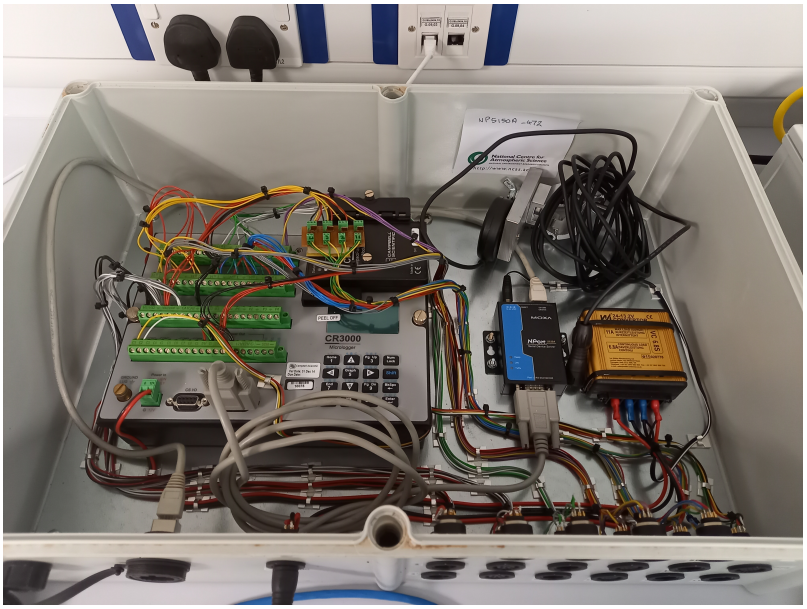
If you get the error `error: invalid command 'bdist_wheel'`, run the below command.

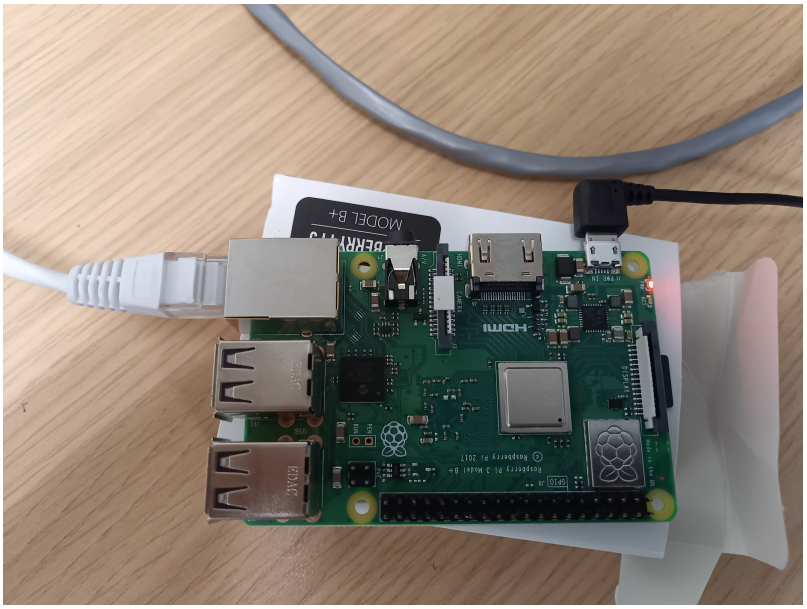
```
$ pip install wheel
```

You also may encounter problems installing netCDF4 on Raspberry Pi. The below steps should be used in the case where you get the error `ValueError: did not find HDF5 headers`.

```
$ sudo apt-get install libhdf5-dev
$ sudo apt-get install libnetcdf-dev
```

Below, images of the logger set up with a Moxa Nport and the Raspberry Pi.





1.2 Downloading Data

To start downloading data from the logger, ensure you have activated your virtual environment and set up any config changes (see what to change and how: config). It is likely you will need to change `logger_url`, `logger_csv_path` and `logger_tables` in your config to get started.

You can either download data from a specific date range:

```
$ cd energy_balance/scripts
$ python download_data_by_date.py -s 2021-07-21 -e 2021-07-23
```

Or download all data from today:

```
$ cd energy_balance/scripts
$ python download_data.py
```

These commands will pull data from the tables you have specified from your logger and save it in csv files at the path you have provided.

1.3 Errors

Sometimes the scripts that retrieve data from the logger may raise an error such as:

```
NoDeviceException
```

If this occurs, please run the script again. The connection to the logger sometimes fails if there are too many queries sent.

CHAPTER

TWO

SCRIPTS

The scripts make use of the package PyCampbellCR1000 (on an updated fork) - this works for CR3000 loggers as well. The documentation for this package can be found here: https://pycampbellcr1000.readthedocs.io/en/latest/

To see the API and source code for these scripts, go to api.

Various settings used in these scripts can be set/changed in the config file: ncas-energy-balance-1-software/energy_balance/etc/config.ini. This includes input/output file paths and settings for netcdf global attributes. This is explained on the config page.

Note: To change the default config settings, provide your own config file by setting the environment variable CONFIG as the file path to your file. e.g.

```
$ export CONFIG='path/to/my/config.ini'
```

Note:

To find your datalogger URL you must know:

- The path to the port and the baud rate of the port. (URL = 'serial:path:baudrate' e.g. 'serial:/dev/ttyUSB0:115200')
- For serial connections, You can also provide the byte size, parity and stop bits if required, this is assumed to be '8N1' if not provided. (e.g. 'serial:/dev/ttyUSB0:115200:8N1')
- It may also be possible to connect via tcp (URL = 'tcp:host-ip:port'), however this is experimental and serial is recommended.

Use the `-h` option on any script to see the command line arguments available.

1. download_data.py:

- Created to be set up as a cron job every 5 minutes (or another time interval). This downloads data from tables on the logger and saves to a daily csv file. Doing this provides a stream of data and saving as daily files allows netCDF files to easily be created.
- If the command line argument `-t` is present, the script will update the logger time at midnight, to prevent the logger time drifting. So if set up to run on an interval with a cron job (that will run at midnight), the logger clock will be reset daily.
- The files are made in the directory specified in the config file, under `logger_csv_path`, under another directory named after the table e.g. `<logger_csv_path>/SoilMoisture/SoilMoisture_2021-07-21.csv`
- The datalogger URL must be set in the config file e.g. serial:/dev/ttyUSB0:115200 or tcp:host-ip:port (see above note explaining this.)

- Edit `logger_tables` in the config file to change the tables downloaded. The default tables are Housekeeping, GPS_datetime, SoilTemperature, SoilMoisture, SoilHeatFlux and Radiation, these are AMOF specific.

To run once:

```
$ cd energy_balance/scripts
$ python download_data.py
```

This will produce an output if successful, example given below, showing how many records were found (repeated for each table):

```
Your download is starting.
Packet 0 with 13 records
Packet 1 with 14 records
Packet 2 with 14 records
---------------------------
41 new records were found
```

To set up a cron job:

```
$ crontab -e
```

Add a command, such as the one below, to this file:

```
*/5 * * * * . /home/pi/ncas-energy-balance-1-software/venv/bin/activate && /home/pi/ncas-
↪energy-balance-1-software/energy_balance/scripts/download_data.py >> /home/pi/campbell_
↪data/data-download-cron.log 2>&1
```

The section `. /home/pi/ncas-energy-balance-1-software/venv/bin/activate` activates the virtual environment. This sets this script running every 5 minutes. The first file path needs to point to your virtual environment and the second to the location of the script. The final file path points to the location at which to write a log file. This can be excluded if this is not required.

The command:

```
$ crontab -l
```

will list cron jobs you have set up.

By using `download_data.py -t`, instead of just `download_data.py`, the logger time will be updated at midnight.

2. download_data_by_date.py:

- Intended to be used to bulk download data over a range of days.
- Useful if system has been turned off/ was down etc.
- This downloads data from tables on the logger and saves to a daily csv file.
- The files are made in the directory specified in the config file, under `logger_csv_path`, under another directory named after the table e.g. `<logger_csv_path>/SoilMoisture/SoilMoisture_2021-07-21.csv`
- Can be used in conjunction with the `download_data.py` script. For example, if the `download_data.py` script has stopped working over a period time, the `download_data_by_date.py` script can be used to fill in these missing days, and will fill partially complete daily files as well.
- The datalogger URL must be set in the config file e.g. serial:/dev/ttyUSB0:115200 or tcp:host-ip:port
- The start and end dates of the days to download should be provided on the command line (in the format YYYY-MM-DD). A start date is required but an end date is not. If an end date is not provided, data is downloaded only for the day provided as the start date.

- If a file for a day has partial data, this script will download the rest of the data for that day, following on from the latest entry in that file.
- Edit `logger_tables` in the config file to change the tables downloaded. The default tables are Housekeeping, GPS_datetime, SoilTemperature, SoilMoisture, SoilHeatFlux and Radiation.

To run:

The below command will download data for 21/07/2021, 22/07/2021 and 23/07/2021 and create a csv file for each day.

```
$ cd energy_balance/scripts
$ python download_data_by_date.py -s 2021-07-21 -e 2021-07-23
```

This next command will download data only for 21/07/2021.

```
$ python download_data_by_date.py -s 2021-07-21
```

You will see a statement saying `Data downloaded for ...` once this is complete.

3. add_to_mysql.py:

- This script will load the csv data for today's files, created by the `download_data` script, into MySQL tables, providing the tables have already been created in the database.
- For information on setting up MySQL on a Raspberry Pi, see https://pimylifeup.com/raspberry-pi-mysql/
- For information on creating tables in MySQL, see https://dev.mysql.com/doc/refman/8.0/en/creating-tables.html
- These updating tables could then be used as a source for visualizing the data, for example with Grafana. This would mean the plots could be kept up to date and allow you to see the data in real time.
- This could be set up as cron job along with the `download_data` script, to keep the tables up to date. See explanation below.
- Edit `logger_tables` and `mysql_tables` in the config file to change the table names to those of your table names from the logger and the corresponding tables you have created in MySQL.
- The default values used for the MySQL tables are housekeeping, gps, soil_temp, soil_moisture, soil_heat_flux and radiation. The defaults used for the logger tables are Housekeeping, GPS_datetime, SoilTemperature, SoilMoisture, SoilHeatFlux and Radiation.
- The top level directory containing the csv files is taken from the config file (under `logger_csv_path`), assumed to be the same as that used to create the files. (i.e. the same as that used for the `download_data.py` script)
- The username, password and database name should also be provided as command line arguments. See below:

```
$ cd energy_balance/scripts
$ python add_to_mysql.py -u <username> -p <password> -d <database>
```

This will output `Inserted data into MySQL tables` if successful.

Setting up as a cron job:

If the download data script is set up every 5 minutes, this script could be set up to run on a 5 minute interval but 3 minutes after the download data script. The jobs in the crontab file would like this:

```
*/5 * * * * . /home/pi/ncas-energy-balance-1-software/venv/bin/activate && /home/pi/ncas-
↪energy-balance-1-software/energy_balance/scripts/download_data.py >> /home/pi/campbell_
↪data/data-download-cron.log 2>&1
3-59/5 * * * * . /home/pi/ncas-energy-balance-1-software/venv/bin/activate && /home/pi/
↪campbell_data/mysql_insert/add_to_mysql.py -u<username> -p<password> -d<database-name>␣
↪>> /home/pi/campbell_data/cron_output/mysql-cron.log 2>&1
```

For extra security, the username and password for the database could be passed in from a text file, preventing them appearing in any logs. This can be done by using the path to the text file as below:

```
3-59/5 * * * * . /home/pi/ncas-energy-balance-1-software/venv/bin/activate && /home/pi/
↪campbell_data/mysql_insert/add_to_mysql.py `cat /home/pi/campbell_data/mysql_insert/
↪args.txt` >> /home/pi/campbell_data/cron_output/mysql-cron.log 2>&1
```

where `/home/pi/campbell_data/mysql_insert/args.txt` contains `-u<username> -p<password> -d<database-name>`

This means that the download data script would run at 00:00, 00:05, 00:10, 00:15 and every 5 minutes after. The mySQL script would run at 00:03, 00:08, 00:13 and every 5 minutes after.

4. create_files.py:

- This script can be used to make netCDF files, that conform to the NCAS-GENERAL Data Standard, for soil and radiation data products. Quality control is carried out during this step, and quality control variables are included in the netCDF file.
- Further details of the values used for quality control by these scripts can be found at: qc
- The quality control level used to calculate valid min/max values is the value set in the config file under `qc_flag_level`.
- Information on how the netCDF file should be built can be found at https://sites.google.com/ncas.ac.uk/ncasobservations/home/data-project/ncas-data-standards/ncas-amof/. Example files can also be found here.
- For this to work, ensure settings in the config file are filled in correctly, e.g. column names, input files, input date format
- Some of the quality control settings can be adjusted in the config file. e.g. the max/min temperature expected for Soil Temperature and the lower and upper bounds for the cleaning time of the radiation sensors. It would be sensible to discuss these settings with the instrument scientist.
- The script takes some command line arguments to specify options for the creation of the files.
- The files are created at the `netcdf_path` specified in the config file.

```
usage: create_files.py [-h] -s START_DATE [-e END_DATE] -f {daily,monthly}
                       -d {soil,radiation}

optional arguments:
-h, --help            show this help message and exit
-s START_DATE, --start-date START_DATE
                        The start date to create netCDF files for. e.g.
                        '2021-07-30' when creating daily files, '2021-07' when
                        creating monthly files.
-e END_DATE, --end-date END_DATE
                        The end date to create netCDF files for. e.g.
                        '2021-07-30' when creating daily files, '2021-07' when
                        creating monthly files. This is inclusive.
-f {daily,monthly}, --frequency {daily,monthly}
                        The frequency for creating the netCDF files, options
                        are daily or monthly.
-d {soil,radiation}, --data-product {soil,radiation}
                        The data product to create files for.
```

A start date is required, but an end date is not. If an end date is not provided, files are only created for the given start date. An example of usage is below.

To create a monthly netCDF file for June 2021, July 2021 and August 2021 for soil:

```
$ cd energy_balance/scripts
$ python create_files.py -s 2021-06 -e 2021-08 -f monthly -d soil
```

The file created for June 2021 would be ncas-energy-balance-1_<platform>_202106_soil_v<version>.nc, where platform and version are set in the config file.

To create a monthly netCDF file for soil for July 2021 only:

```
$ cd energy_balance/scripts
$ python create_files.py -s 2021-07 -f monthly -d soil
```

The file created would be called ncas-energy-balance-1_<platform>_202107_soil_v<version>.nc, where platform and version are set in the config file.

To create daily netCDF files for each day between 20th July 2021 and 27th July 2021 for radiation:

```
$ cd energy_balance/scripts
$ python create_files.py -s 2021-07-20 -e 2021-07-27 -f daily -d radiation
```

A file would be created for each day, e.g. for 20th July 2021: ncas-energy-balance-1_<platform>_20210720_radiation_v<versi
nc, where platform and version are set in the config file.

5. calculate_valid_min_max.py:

- This script allows you to recalculate the valid min/max variables after manually changing the values of a quality control flag variable.
- For example, the qc flag variable for soil_temperature is qc_flag_soil_temperature. If values of the qc flag variable are changed, it may change the valid minimum/maximum.
- The quality control level used remains the value set in the config file under qc_flag_level.
- To update the valid max/min values, use this script as below:

```
$ cd energy_balance/scripts
$ python calculate_valid_min_max.py -v soil_temperature -qc qc_flag_soil_temperature -fp␣
↪/path/to/ncas-energy-balance-1_lab_20210730_soil_v0.1.nc
```

Once complete, you will see a message, e.g.

```
Recalculated valid min and valid max for soil_temperature, using qc_flag_soil_
↪temperature as a mask, with qc flag value of 1
```

In general, the usage is:

```
usage: calculate_valid_min_max.py [-h] [-v VAR_NAME] [-qc QC_VAR_NAME] -fp FILE_PATH

optional arguments:
    -h, --help              show this help message and exit
    -v VAR_NAME, --var-name VAR_NAME
                            The name of the variable to update the min/max on.
                            e.g. 'soil_temperature'
    -qc QC_VAR_NAME, --qc-var-name QC_VAR_NAME
                            The name of the quality control variable to use as a
                            mask for retrieving valid values. e.g.
                            'qc_flag_soil_temperature'
```

(continues on next page)

(continued from previous page)

```
  -fp FILE_PATH, --file-path FILE_PATH
                        The path to netCDF file on which to recalculate the
                        min/max e.g. /path/to/my/file.nc
```

6. create_qc_csvs.py:

- This script will generate csvs for soil/radiation data that have been quality controlled according the level of quality control specified in the config file. These can then be plotted to see how changing the quality control changes the plot.
- This will only apply automatic quality control as discussed in qc and will not take into account any manual changes done on the netCDF file.
- Only columns used as variables in the netCDF files will be included. In the soil files these are: soil temperature, soil water potential, soil heat flux. In the radiation files: downwelling longwave radiation in air, upwelling longwave radiation in air, downwelling shortwave radiation in air, upwelling shortwave radiation in air and radiometer body temperature.
- The name of the file created will be `<data_product>_qc_<date>.csv` e.g. `soil_qc_20210730.csv`.
- The files are made in the directory specified in the config file, under `qc_csv_path`.
- The quality control carried out flags data outside operational bounds, suspect data and data taken when sensors are being cleaned. To do this a quality control matrix is created, assigning each value a quality control flag. These are numbers from 0 to 255.
  - 0 is not used.
  - 1 means the data is 'good' i.e. it is within operational and expected bounds and hasn't raised any suspicion.
  - Further values 2, 3, 4 etc. are assigned specific definitions e.g. 2 could mean the data is outside the operational bounds, 3 could mean there is a timestamp error.
  - Further details of the values used for quality control by these scripts can be found at: qc
- The flag level to use can be set in the config file under `qc_flag_level`. Setting the level as 1, means only 'good' data is provided. This can be increased to include data from other qc flags, as described by the variables in the NetCDF files. (The level chosen will include data from that level and below.)
- Some of the quality control settings can be adjusted in the config file. e.g. the max/min temperature expected for Soil Temperature and the lower and upper bounds for the cleaning time of the radiation sensors. It would be sensible to discuss these settings with the instrument scientist.
- These csvs can be plotted using script #6 below.

```
usage: create_qc_csvs.py [-h] -s START_DATE [-e END_DATE] -f {daily,monthly}
                         -d {soil,radiation}

optional arguments:
-h, --help            show this help message and exit
-s START_DATE, --start-date START_DATE
                        The start date to create files for. e.g.
                        '2021-07-30' when creating daily files, '2021-07' when
                        creating monthly files.
-e END_DATE, --end-date END_DATE
                        The end date to create files for. e.g.
                        '2021-07-30' when creating daily files, '2021-07' when
                        creating monthly files. This is inclusive.
-f {daily,monthly}, --frequency {daily,monthly}
```

(continues on next page)

(continued from previous page)

```
                        The frequency for creating the csv files, options
                        are daily or monthly.
-d {soil,radiation}, --data-product {soil,radiation}
                        The data product to create files for.
```

```
$ cd energy_balance/scripts
$ python create_qc_csvs.py -s 2021-07-30 -f daily -d radiation
```

An example of how the data could look before and after the quality control, in csv format, is shown below:

BEFORE:

```
Datetime,WP_kPa_1,T107_1,shf_1
2021-07-30 00:00:00,101.2294921875,21.69464111328125,0.4606184959411621,
2021-07-30 00:05:00,67.27587890625,21.682518005371094,8.577472686767578
2021-07-30 00:10:00,55.2167313385,21.796310424804688,2.078993320465088
2021-07-30 00:15:00,86.1962890625,21.664581298828125,0.1369409263134002
```

AFTER:

```
Datetime,WP_kPa_1,T107_1,shf_1
2021-07-30 00:00:00,,21.69464111328125,0.4606184959411621,
2021-07-30 00:05:00,67.27587890625,21.682518005371094,8.577472686767578
2021-07-30 00:10:00,55.2167313385,21.796310424804688,2.078993320465088
2021-07-30 00:15:00,,21.664581298828125,0.1369409263134002
```

The 2 soil water potential values (column WP_kPa_1) over 80kPa hav been masked out, as this is one of the quality control settings.

7. plot_csv.py:

- This script can be used to generate quick plots from csv files, provided the file contains a date/time column, using matplotlib. It will plot the csv columns you specify against datetime.
- The name of the datetime column must be specified in the config file, under `datetime_header`.
- This will allow you take a quick look at any data, and could be used to look at how the plot changes when data is masked from the quality control.
- The command line options allow you to specify the datetimes to plot between and which columns of the csv to plot.
- If a start and/or end date are not provided, these will default to the start/end times in the csv.

```
usage: plot_csv.py [-h] [-s START] [-e END] -f FILE -c COLUMNS

optional arguments:
-h, --help            show this help message and exit
-s START, --start START
                        The start date/time for the plot in 'YYYY-MM-dd
                        HH:MM:SS' format. e.g. '2021-07-10 04:00:00'.
-e END, --end END     The end date/time for the plot in 'YYYY-MM-dd
                        HH:MM:SS' format. e.g. '2021-07-10 16:00:00'.
-fp FILE_PATH, --file-path FILE_PATH
                        The path to the csv file to plot. e.g. /path/to/file.csv
-c COLUMNS, --columns COLUMNS
```

(continues on next page)

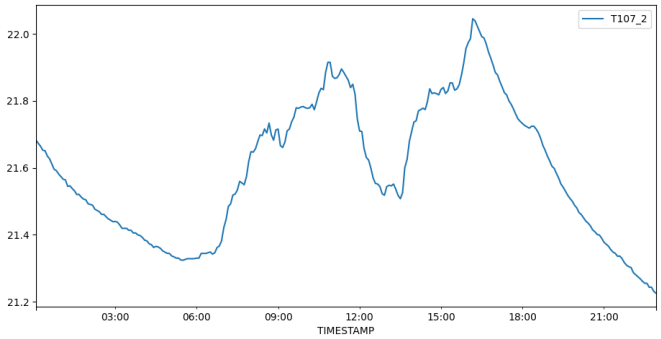
(continued from previous page)

```
                        The columns from the csv to plot against datetime,
                        provide as comma separated list if more than one e.g. 'IR01Dn,
↪IR01Up'.
```

Note that datetimes should be provided in quotations to allow them to be parsed correctly.

```
$ cd energy_balance/scripts
$ python plot_csv.py -s '2021-07-10 04:00' -e '2021-07-10 16:00' -fp /path/to/my/file.
↪csv -c shf_1,shf_2,shf_3
```

An example plot, of temperature from sensor 2, is show below:



The command for this was:

```
$ python plot_csv.py -s '2021-07-30 00:00:00' -e '2021-07-30 23:59:00' -fp path/to/
↪SoilTemperature_2021-07-30.csv -c T107_2
```

CHAPTER

THREE

CONFIG

This explains the various options that can be set when using the scripts. To see the config file: https://github.com/ncasuk/ncas-energy-balance-1-software/blob/master/energy_balance/etc/config.ini

To override these settings, provide your own config file by setting the environment variable CONFIG as the file path to your file. e.g.

```
$ export CONFIG='path/to/my/config.ini'
```

3.1 Specifying types

It is possible to specify the type of the entries in the configuration file, for example if you want a value to be a list when the file is parsed.

This is managed through a `[config_data_types]` section at the top of the INI file which has the following options:

```
[config_data_types]
# only used by installed package
lists =
dicts =
ints =
floats =
boolean =
# use the below when creating your own file
extra_lists =
extra_dicts =
extra_ints =
extra_floats =
extra_booleans =
```

Simply adding the name of the value you want to format afer = will render the correct format. e.g. `ints = index_length processing_level qc_flag_level` will set `index_length`, `processing_level` and `qc_flag_level` as ints.

3.2 Settings

These settings apply across the scripts:

```
[common]
# url for logger E.g. tcp:iphost:port or serial:/dev/ttyUSB0:19200:8N1 or serial:/
↪COM1:19200:8N1
logger_url = serial:/dev/ttyUSB0:115200
# latitude of station
latitude_value = 13.1
# longitude of station
longitude_value = 25.5
# name of date/time column in files/on logger
datetime_header = Datetime
# path to output/read logger csv files from
logger_csv_path = ~/ncas-energy-balance-1-software
# where to output netcdf files
netcdf_path = ~/ncas-energy-balance-1-software
# path to output qc csv files
qc_csv_path = ~/ncas-energy-balance-1-software
# masking will include values that have been quality controlled to this level or less (e.
↪g. <= 1)
qc_flag_level = 1
# fill value to use in netcdf files
fill_value = -1e+20
# names of the data tables in the logger to process in scripts
logger_tables = Housekeeping GPS_datetime SoilTemperature SoilMoisture SoilHeatFlux␣
↪Radiation
# names of the tables you have created in mysql (must map to logger tables)
mysql_tables = housekeeping gps soil_temp soil_moisture soil_heat_flux radiation
```

These settings are specific for the soil data product:

```
[soil]
# number of sensors
index_length = 3
# input file path to directory containing soil csv files to turn into netcdf files
input_file_path = ~/ncas-energy-balance-1-software
# double % to escape as it is a special character
# date format as it is on the input files
input_date_format = %%Y-%%m-%%d
# location and format of files - use {date} where the date exists in the file name
# if files are found in directories under the main 'input file path directory', include␣
↪that here
soil_moisture_file = SoilMoisture/SoilMoisture_{date}.csv
soil_temperature_file = SoilTemperature/SoilTemperature_{date}.csv
soil_heat_flux_file = SoilHeatFlux/SoilHeatFlux_{date}.csv
# columns to extract: what the columns are called in the input csv file to allow them to␣
↪be extracted
# give the below in index order i.e. the first listed will be given index 1
soil_moisture_headers = WP_kPa_1 WP_kPa_2 WP_kPa_3
soil_temperature_headers = T107_1 T107_2 T107_3
soil_heat_flux_headers = shf_1 shf_2 shf_3
```

(continues on next page)

(continued from previous page)

```
# the max/min expected soil temp in degrees C. Any temperature outside this will be␣
↪flagged as suspect data
max_expected_temp = 28
min_expected_temp = 18
```

These settings are specific for the radiation data product:

```
[radiation]
# input file path to directory containing radiation csv files to turn into netcdf files
input_file_path = ~/ncas-energy-balance-1-software
# double % to escape as it is a special character
# date format as it is on the input files
input_date_format = %%Y-%%m-%%d
# location and format of file
# if file is found in directories under the main 'input file path directory', include␣
↪that here
radiation_file = Radiation/Radiation_{date}.csv
# columns to extract: what the columns are called in the input csv file to allow them to␣
↪be extracted
# give the below in index order i.e. the first listed will be given index 1
# longwave downwelling
lwdn_header = IR01Dn
# longwave upwelling
lwup_header = IR01Up
# shortwave downwelling
swdn_header = SR01Dn
# shortwave upwelling
swup_header = SR01Up
# radiometer body temperature (in kelvin)
body_temp_header = NR01TK
# the time range to qc as 'sensor being cleaned'
# give in hh:mm:ss
cleaning_time_lower = 05:55:00
cleaning_time_upper = 06:05:00
```

These settings correspond to the global attributes on the netCDF files produced. Anything set here will be set as a global attribute:

```
[global]
Conventions = CF-1.6, NCAS-AMF-2.0.0
source = NCAS Energy Balance Station unit 1
instrument_manufacturer = Campbell Scientific
instrument_model = CR3000
instrument_serial_number =
instrument_software = EB1_logger.cr5
instrument_software_version = v1
creator_name = Eleanor Smith
creator_email = eleanor.smith@stfc.ac.uk
creator_url = https://orcid.org/0000-0002-6448-5778
institution = Centre for Environmental Data Analysis (CEDA)
processing_software_url = https://github.com/ncasuk/ncas-energy-balance-1-software
processing_software_version = v0.1
```

(continues on next page)

(continued from previous page)

```
calibration_sensitivity =
calibration_certification_date =
calibration_certification_url =
sampling_interval = 5 minute
averaging_interval = 5 minute
product_version = 0.1
processing_level = 1
project = energy balance placement
project_principal_invesitgator = Eleanor Smith
project_principal_invesitgator_email = eleanor.smith@stfc.ac.uk
project_principal_invesitgator_url = https://orcid.org/0000-0002-6448-5778
licence = Data usage licence - UK Government Open Licence agreement: http://www.
↪nationalarchives.gov.uk/doc/open-government-licence
acknowledment = Acknowledgement of NCAS as the data provider is required whenever and␣
↪wherever these data are used
platform = lab
platform_type = stationary_platform
deployment_mode = land
title = Measurements from the NCAS energy balance station.
featureType = timeSeries
geospatial_bounds =
platform_altitude =
location_keywords =
amf_vocabularies_release = https://github.com/ncasuk/AMF_CVs/tree/v2.0.0
history =
comment =
```

CHAPTER

FOUR

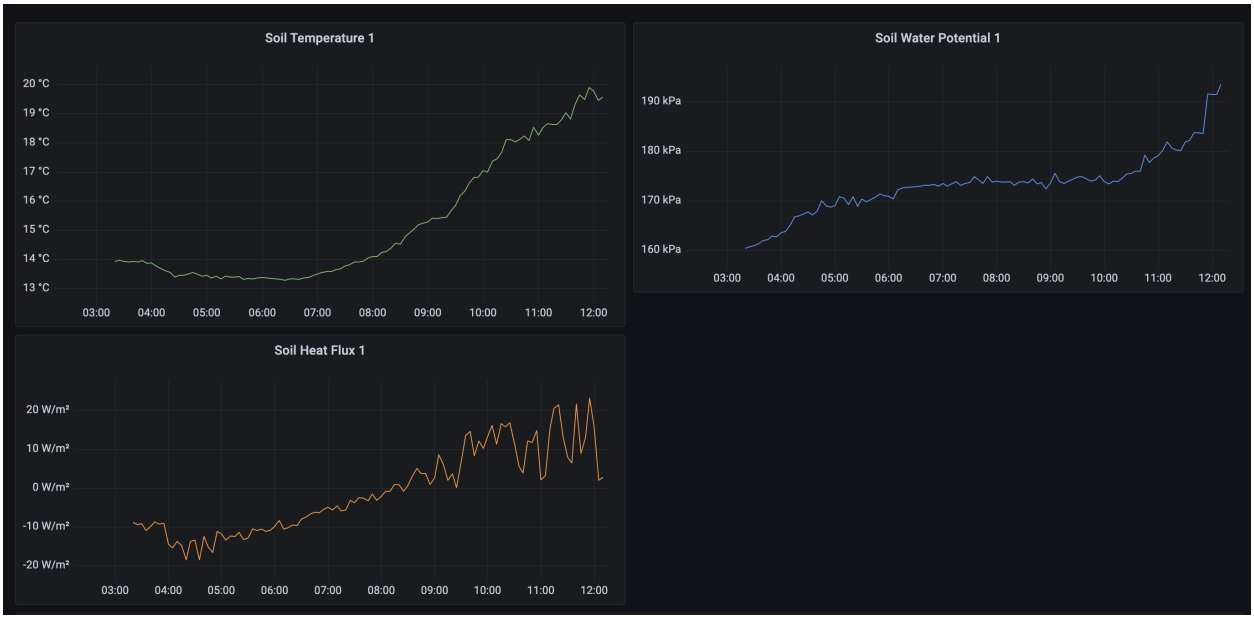
GRAFANA

Grafana can be used to visualize data and can be used in conjunction with the download_data.py and add_to_mysql.py scripts (script numbers 1 and 3) to create plots that are kept up to date.

The MySQL database that you create when using the add_to_mysql.py script can be set up as a data source for Grafana.

With the data download and add to MySQL scripts running in 5 minute intervals with cron jobs, as discussed on the scripts page, the data will be kept up to date.

For an example, see the image below:



This shows the data for the Soil Temperature, Soil Water Potential and Soil Heat Flux for sensor channel 1. This allows you to see changes in the data over time and notice any anomalies. You can set the time range to be much smaller or larger than that shown in the image and can zoom in on data to see it in more detail.

These plots are time series plots, but there are many more types of plot available such as bar chart, gauge, pie chart and others.

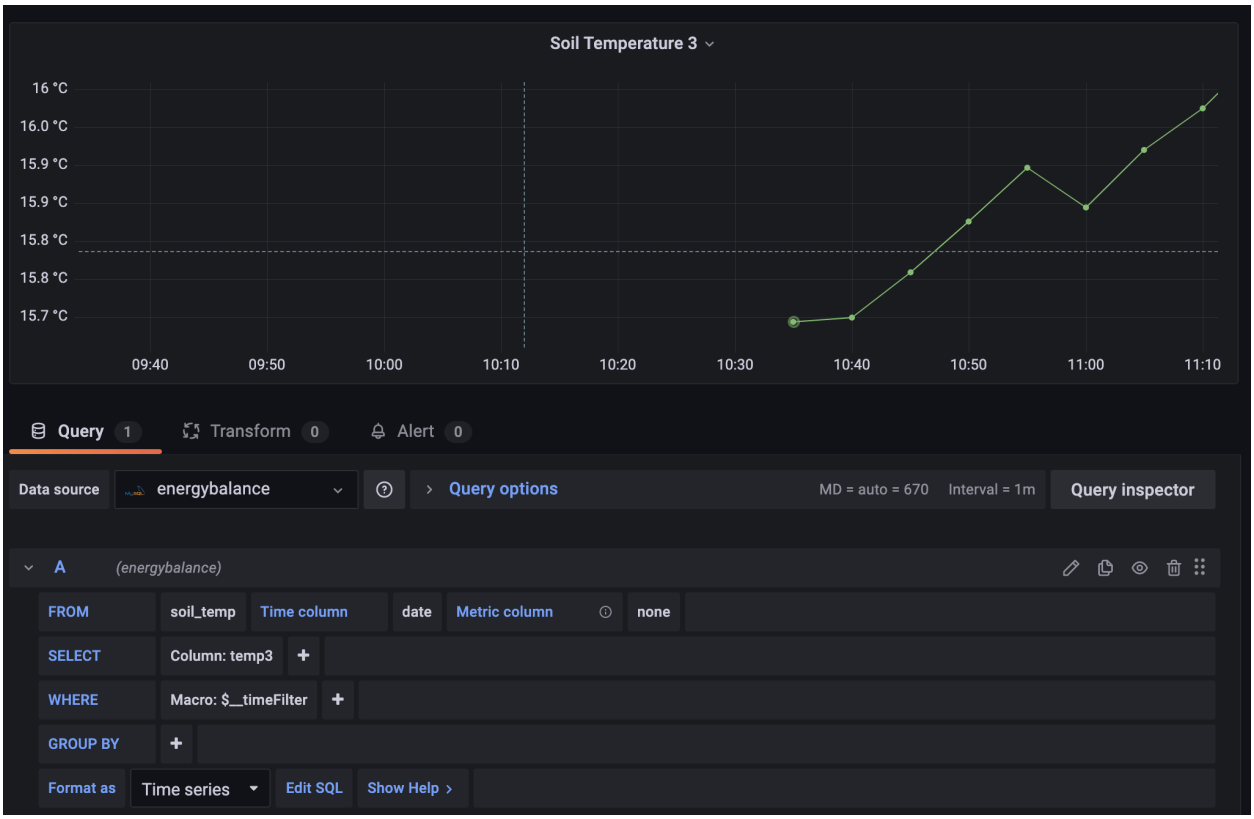
There are other configurable options on Grafana. For example, it is possible to set up alerts if the data exceeds or goes below a certain value. You can also share a snapshot of the dashboard you create with someone else.

4.1 Install

- See instructions at https://grafana.com/tutorials/install-grafana-on-raspberry-pi/ for setting up Grafana on a Raspberry Pi.
- To set up MySQL as a data source see https://grafana.com/docs/grafana/v7.5/datasources/mysql/

4.2 Create a plot

- To create plots you must create a new dashboard, then use the 'add panel' button in the top right corner.
- Then, under `Query`, you can select the data source required (my database is called `energybalance`), the table from the database to use (my table is called `soil_temp`), and set the time column and the column to plot. (In the image below, the time column is called `date` and the column I am plotting is called `temp3`.)



- There are options to change the name of the plot, line colours, axis names etc. on the right hand side.

CHAPTER

FIVE

QUALITY CONTROL

This explains the quality control values and definitions used when creating quality control csvs and netCDF files (which include quality control variables once created) For more information on how these netCDF files have been built see https://sites.google.com/ncas.ac.uk/ncasobservations/home/data-project/ncas-data-standards/ncas-amof/. The control quality control variables are included as they are 'Data Product Specific Variables'.

5.1 soil

For the soil data product, 3 quality control variables are created. Below each, the values and meanings used are listed. These are also included in the netCDF file created. These are:

1. qc_flag_soil_heat_flux

   0: not used 1: good data 2: bad data - value outside operational range (-30 to 70 C) (e.g. if the soil temperature is outside this range, the corresponding value of soil heat flux is flagged) 3: suspect data 4: timestamp error

   - flag 2 is applied automatically, all other data is marked as 1.

2. qc_flag_soil_temperature

   0: not used 1: good data 2: bad data - value outside operational range (-35 to 50 C) 3: suspect data (includes temperature data that is higher/ lower than expected - these expected values can be set in the config file - speak to instrument scientist to find out what this should be) 4: timestamp error

   - flags 2 and 3 (3 for outside range of expected temperature) are applied automatically, all other data is marked as 1. More data can be manually marked as suspect if needed.

3. qc_flag_soil_water_potential

   0: not used 1: good data 2: bad data - soil water potential over 80kPa (contact between soil and sensor usually lost at this point) 3: bad data - value outside operational range (0 to 200 kPa) 4: suspect data 5: timestamp error

   - flags 2 and 3 are applied automatically, all other data is marked as 1.

5.2 radiation

For the radiation data product, 6 quality control variables are created.

1. qc_flag_upwelling_shortwave

   0: not used 1: good data 2: no data 3: bad data sw radiation < 0 4: bad data sw radiation > 2000 W m-2 5: suspect data 6: timestamp error

   - flags 2, 3 and 4 are applied automatically, all other data is marked as 1.

2. qc_flag_downwelling_shortwave

   0: not used 1: good data 2: no data 3: bad data sw radiation < 0 4: bad data sw radiation > 2000 W m-2 5: suspect data 6: timestamp error

   - flags 2, 3 and 4 are applied automatically, all other data is marked as 1.

3. qc_flag_upwelling_longwave

   0: not used 1: good data 2: no data 3: bad data sw radiation < 0 4: bad data sw radiation > 1000 W m-2 5: suspect data 6: timestamp error

   - flags 2, 3 and 4 are applied automatically, all other data is marked as 1.

4. qc_flag_downwelling_longwave

   0: not used 1: good data 2: no data 3: bad data sw radiation < 0 4: bad data sw radiation > 1000 W m-2 5: suspect data 6: timestamp error

   - flags 2, 3 and 4 are applied automatically, all other data is marked as 1.

5. qc_flag_body_temperature

   0: not used 1: good data 2: bad data body temperature outside operational range -40 to 80C 3: suspect data 4: timestamp error

   - flag 2 is applied automatically, all other data is marked as 1. This flag is applied to all variables, as they are all affected if the body temperature is outside operational bounds.

6. qc_flag_cleaning

   0: not used 1: good data 2: bad data sensor being cleaned (the times for this should be set in the config file - speak to instrument scientist to find out what this should be) 3: suspect data 4: timestamp error

   - flag 2 is applied automatically, all other data is marked as 1. This flag is applied to all variables, as they are all affected when the sensor is being cleaned.

- The script to create netCDF files or that which makes the qc csvs, automatically applies flags as mentioned under each variable. Any other flags are there to be added manually after inspection of the data.
- Manually adding flags may result in the valid max/min values of variables needing to be changed in the netCDF file - these are calculated only from 'good data'. There is a script available to recalculate these values. (Script 5 - calculate_valid_min_max.py)
- Suspect data has been included to cover other scenarios not covered by the other flags e.g. data is higher/lower than expected, the data has changed significantly since the last reading.

CHAPTER

SIX

API

6.1 Scripts

1. download_data.py:

2. download_data_by_date.py:

energy_balance.scripts.download_data_by_date.**get_data**(*url*, *start_date*, *end_date*, *dir_path*)

Extract data from the campbell data logger for each specified table and save to a daily csv file between the date ranges specified. Default tables are: Housekeeping, GPS_datetime, SoilTemperature, SoilMoisture, SoilHeatFlux and Radiation

Parameters

- **url** – (str) URL for connection with logger in format 'tcp:iphost:port' or 'serial:/dev/ttyUSB0:19200:8N1'
- **start_date** – (datetime.datetime) The start date from which to collect data
- **end_date** – (datetime.datetime) The end date after which to stop collecting data. (the end date will be included in the data.)
- **dir_path** – (str) The path to the top level directory in which to create the csv files and folders.

Returns None

energy_balance.scripts.download_data_by_date.**get_data_from_range**(*device*, *table*, *csv_path*, *start*, *end*, *header*)

Gets range of data specified by start and end dates and saves to csv at the path specified.

Parameters

- **device** – (pycampbellcr1000.CR1000 object) URL for connection with logger in format 'tcp:iphost:port' or 'serial:/dev/ttyUSB0:19200:8N1'
- **table** – (str) The name of the table on the logger from which the data is being extracted.
- **csv_path** – (str) The path to the csv file to back fill with todays data.
- **start** – (datetime.datetime) The start datetime from which to collect data
- **end** – (datetime.datetime) The end datetime after which to stop collecting data. (end will be included in the data.)

Returns None

3. add_to_mysql.py:

energy_balance.scripts.add_to_mysql.**insert_into_tables**(*user*, *password*, *database*, *dir_path*)
Gets data from the csv files found in the specified directory path and inserts it into MySQL tables. The MySQL tables must have been created proir to running this. The default names used to map the logger tables to MySQL tables are: {'Housekeeping': 'housekeeping', 'GPS_datetime': 'gps', 'SoilTemperature': 'soil_temp', 'SoilMoisture': 'soil_moisture', 'SoilHeatFlux': 'soil_heat_flux', 'Radiation': 'radiation'}

Parameters

- **user** – (str) The username for connecting to MySQL.
- **password** – (str) The password for connecting to MySQL.
- **database** – (str) The names of the database in which the tables exist.
- **dir_path** – (str) The path to the top level directory in which the csv files and folders were created.

Returns None

4. create_files.py:

energy_balance.scripts.create_files.**create_files**(*start_date*, *end_date*, *frequency*, *data_product*)
Create netcdf files for the specified data product in the time range provided. If no data product is provided, netcdf files are created for soil and radiation.

Parameters

- **start_date** – (datetime.datetime) The start date for which to create the files.
- **end_date** – (datetime.datetime) The end date for which to create the files.
- **frequency** – (str) The frequency for files - daily or monthly.
- **data_product** – (str) The data product to create the netcdf files for e.g. radiation or soil

Returns None

energy_balance.scripts.create_files.**create_radiation_files**(*date*, *frequency*)
Create radiation netcdf.

Parameters

- **date** – (datetime.datetime) The date for which to create the file.
- **frequency** – (str) The frequency for files - daily or monthly.

Returns None

energy_balance.scripts.create_files.**create_soil_files**(*date*, *frequency*)
Create soil netcdf.

Parameters

- **date** – (datetime.datetime) The date for which to create the file.
- **frequency** – (str) The frequency for files - daily or monthly.

Returns None

energy_balance.scripts.create_files.**get_create_file**(*data_product*)
Get the function for creating files for the specified data product.

5. calculate_valid_min_max.py:

energy_balance.scripts.calculate_valid_min_max.**calculate_valid_min_max**(*fpath*, *var_name*, *qc_var_name*, *qc_value*)

Re calculate valid min and valid max for a variable, given the quality control variable and maximum desired quality control value. Useful after quality control variable has been changed manually.

Parameters

- **fpath** – (str) Path to netCDF file on which to calculate the min/max
- **var_name** – (str) The name of the variable to update the min/max on.
- **qc_var_name** – (str) The name of the quality control variable to use as a mask for retrieving valid values.
- **qc_value** – (int) Max value of qc to use i.e. 1 will calculate min/max on only 'good data', 2 will calculate it on good data and data marked with a flag of 2.

6. create_qc_csvs.py:

energy_balance.scripts.create_qc_csvs.**create_files**(*start_date*, *end_date*, *frequency*, *data_product*, *fpath*)

Create masked csvs for the specified data product in the time range provided.

Parameters

- **start_date** – (datetime.datetime) The start date for which to create the files.
- **end_date** – (datetime.datetime) The end date for which to create the files.
- **frequency** – (str) The frequency for files - daily or monthly.
- **data_product** – (str) The data product to create the csvs for e.g. radiation or soil
- **fpath** – (str) The directory path at which to create the output file.

Returns None

energy_balance.scripts.create_qc_csvs.**create_radiation_files**(*date*, *frequency*, *path*)

Create radiation masked csv.

Parameters

- **date** – (datetime.datetime) The date for which to create the file.
- **frequency** – (str) The frequency for files - daily or monthly.

Returns None

energy_balance.scripts.create_qc_csvs.**create_soil_files**(*date*, *frequency*, *path*)

Create soil masked csv.

Parameters

- **date** – (datetime.datetime) The date for which to create the file.
- **frequency** – (str) The frequency for files - daily or monthly.

Returns None

energy_balance.scripts.create_qc_csvs.**get_create_file**(*data_product*)

Get the function for creating files for the specified data product.

energy_balance.scripts.create_qc_csvs.**prepare_date**(*date*, *frequency*)

Convert datetimes to strings, dependening on frequency. If monthly: format returned will be %Y%m If daily: format returned will be %Y%m%d

Parameters

- **date** – (datetime.datetime) The date for which the file is being created.
- **frequency** – (str) The frequency for files - daily or monthly.

Returns (str) The date converted to string format.

7. plot_csv.py:

energy_balance.scripts.plot_csv.**plot**(*start*, *end*, *columns*, *fpath*)
Plot the columns from the csv specified.

Parameters

- **start** – (str) The start date/time from which to plot.
- **end** – (str) The end date/time for the plot.
- **columns** – (list) List of columns to plot.
- **fpath** – (str) File path of csv file.

Returns None

energy_balance.scripts.plot_csv.**validate_time**(*time*)
Validate that time is in the correct format (Y-M-d H:M:S)

Parameters **time** – (str) The time string to validate

Return time (str) The input time, if validated. Otherwise an exception is raised.

6.2 Quality control

class energy_balance.netcdf.quality_control.**QualityControl**(*date*, *frequency*)
Bases: object

Base class used for apply quality control to data in pandas data frames. Creates a quality control dataframe and a masked dataframe (the initial data with a quality control mask applied) from input csv files. The input files and various options are taken from a config file.

Constant values are taken from the config file, excluding 'headers' which must be set in each specific implementation.

Parameters

- **date** – (datetime.datetime) The date to do the QC for. If frequency is monthly, only the year and month will be taken into account.
- **frequency** – (str) 'daily' or 'monthly'. Determines whether one days worth of data, or one months worth is taken from the csv files to create the dataframes.

apply_qc(*conditions*, *choices*, *col*)
Generic method to apply QC to a column in a dataframe, new column is created in QC dataframe.

Parameters

- **conditions** – (list) The conditions at which a QC flag should be applied. e.g. [np.isnan(self._df[col]), self._df[col] < -35, self._df[col] > 50]
- **choices** – (list) The QC flag to be applied, corresponds to conditions. e.g. [2, 2, 2]
- **col** – (str) The name of the column to apply QC to e.g. 'WP_kPa_1'

create_dataframes()
Class specific implementation to create pandas dataframe from input csv and empty QC dataframe other than column names. Sets self._df and self._qc

create_masked_csv(*file_path*)
Create a csv file from the masked dataframe.

Parameters **file_path** – (str) The path at which to create the csv file e.g. /path/to/my/file.csv

create_masked_df(*qc_flag*)
Create masked pandas dataframe based on self._qc and the qc flag requested. Sets self._df_masked.

Parameters **qc_flag** – (int) Max value of qc to show i.e. 1 will show only 'good data', 2 will show good data and data marked with a flag of 2.

property df
Returns the original dataframe created from the input csv files. All headers set in each class implementaiton of self.headers are included.

property df_masked
Returns the original dataframe masked following QC.

dt_header = 'Datetime'

execute_qc()
Create the dataframes, apply the QC and create the masked dataframe.

headers = 'UNDEFINED'

prepare_date(*input_date_format*)
Prepares the input date format so it matches with the frequency requested.

Parameters **input_date_format** – (str) The format in which the date is provided in the input csv files.

Returns (str) The date now converted to string format.

property qc
Returns the QC dataframe created based on conditions and choices set in the qc_variables method.

qc_flag_level = 1

qc_variables()
Class specific implementation to apply QC to all columns.

class energy_balance.netcdf.soil_quality_control.**SoilQualityControl**(*date*, *frequency*)
Bases: energy_balance.netcdf.quality_control.QualityControl

create_dataframes()
SoilQualityControl specific implementation to create pandas dataframe from input csvs and empty QC dataframe other than column names. Sets self._df and self._qc

qc_variables()
SoilQualityControl specific implementation to set QC conditions and flags and record in QC dataframe.

class energy_balance.netcdf.radiation_quality_control.**RadiationQualityControl**(*date*, *frequency*)
Bases: energy_balance.netcdf.quality_control.QualityControl

apply_cleaning_and_temp_masks()
Apply cleaning QC and body temperature QC to all columns in the dataframe.

`create_dataframes()`
RadiationQualityControl specific implementation to create pandas dataframe from input csvs and empty QC dataframe other than column names. Sets self._df and self._qc

`create_masked_df(qc_flag)`
RadiationQualityControl specific implementation to create the masked dataframe.

`qc_variables()`
RadiationQualityControl specific implementation to set QC conditions and flags and record in QC dataframe.

6.3 NetCDF

`class energy_balance.netcdf.base_netcdf.BaseNetCDF(df, qc, date, frequency)`
Bases: `object`

Base class used for creating netCDF files. Creates all the common variables found in netCDF files under the NCAS-GENERAL Data Standard. Sets all the required global attributes.

Constant values are taken from the config file, excluding 'headers' and 'data_product' which must be set in each specific implementation.

Parameters

- **df** – A pandas dataframe containing all columns required to create the netCDF file.
- **qc** – A pandas dataframe with the same columns as df, but containing the quality control values instead. (i.e. 1, 2, 3 etc.)
- **date** – (datetime.datetime) The date to create the netCDF file for. If frequency is monthly, only the year and month will be taken into account.
- **frequency** – (str) 'daily' or 'monthly'. Determines whether the file will use data from one day or for one month.

`convert_date_to_string(date, frequency)`
Generate a date string for the file name based on the date provided and the frequency required.

Parameters

- **date** – (datetime.datetime) The date to convert to string.
- **frequency** – (str) The frequency at which to have the date string.

Returns (str) The date now converted to string format.

`static convert_times(times)`
Convert times from strings to total seconds since 1970-01-01T00:00:00.

Parameters **times** – (sequence) Times to convert to total seconds since 1970-01-01T00:00:00.

Returns (list) The times converted to total seconds since 1970-01-01T00:00:00.

`create_lat_variable()`
Create the common latitude variable.

`create_lon_variable()`
Create the common longitude variable.

`create_netcdf()`
Method to create the netCDF dataset

create_qc_variable(*name*, *header*, *dimensions*, ***kwargs*)
Generic method to create a qc variable on the dataset.

Parameters

- **name** – (str) The name of the variable to be created.
- **header** – (str) The name of the column in the df pandas dataframe to use to populate the data of this variable.
- **dimensions** – (tuple) The dimensions of the variable to be created e.g. ('time',) or ('time', 'index')
- **kwargs** – (dict) Dictionary of attributes {'attr_name': 'attr_value'} to set on the variable e.g. {'standard_name': 'soil_temperature'}

create_specific_dimensions()
Class specific implementation to create dimensions specific to that data product.

create_specific_variables()
Class specific implementation to create variables specific to that data product, including any qc variables.

create_time_related_variable(*name*, *data_type*, *values*, *long_name*)
Generic method to create variables day of year, day, year, month, hour, second, minute.

Parameters

- **name** – (str) The name of the variable to be created.
- **data_type** – The data type of the variable to be created e.g. numpy.float32
- **values** – (sequence) The values to set for this variable.
- **long_name** – (str) The long name of this variable.

create_time_variable()
Create the common time variable.

create_variable(*name*, *data_type*, *dims*, *header*, ***kwargs*)
Generic method to create a variable in the netCDF4 dataset.

Parameters

- **name** – (str) The name of the variable to be created.
- **data_type** – The data type of the variable to be created e.g. numpy.float32
- **dims** – (tuple) The dimensions of the variable to be created e.g. ('time',) or ('time', 'index')
- **header** – (str) The name of the column in the pandas dataframe to use to populate the data of this variable.
- **kwargs** – (dict) Dictionary of attributes {'attr_name': 'attr_value'} to set on the variable e.g. {'standard_name': 'soil_temperature'}

data_product = 'UNDEFINED'

dt_header = 'Datetime'

fill_value = -1e+20

get_masked_data(*mask_value*)
Create masked pandas dataframe based on self.qc and the qc flag requested. Sets self.df_masked.

Parameters **mask_value** – (int) Max value of qc to show i.e. 1 will show only 'good data', 2 will show good data and data marked with a flag of 2.

headers = 'UNDEFINED'

qc_flag_level = 1

set_global_attributes()
Sets the global attributes in the dataset based on those listed in the config file.

static times_as_datetimes(*times*)
Convert times from strings to datetimes.

Parameters **times** – (sequence) Times to convert to datetimes in format Y-m-d H:M:S.

Returns (list) The times converted to datetimes.

class energy_balance.netcdf.soil_netcdf.**SoilNetCDF**(*df*, *qc*, *date*, *frequency*)
Bases: energy_balance.netcdf.base_netcdf.BaseNetCDF

Class for creating soil netcdf files. Creates soil specific dimensions and variables

static convert_temps_to_kelvin(*temps*)
Convert temperatures from degrees celsius to Kelvin.

Parameters **temps** – (sequence) Temperatures to convert (in degrees C).

Returns (list) The temperatures converted to Kelvin.

create_qc_variable(*name*, *headers*, ***kwargs*)
SoilNetCDF specific implementation to account for index dimension.

Parameters

- **name** – (str) The name of the variable to be created.
- **header** – (list) The names of the columns in the df pandas dataframe to use to populate the data of this variable.
- **kwargs** – (dict) Dictionary of attributes {'attr_name': 'attr_value'} to set on the variable e.g. {'standard_name': 'soil_temperature'}

create_soil_heat_flux_variable()
Create soil heat flux variable on the netCDF dataset.

create_soil_moisture_variable()
Create soil water potential variable on the netCDF dataset.

create_soil_temp_variable()
Create soil temperature variable on the netCDF dataset.

create_specific_dimensions()
SoilNetCDF specific implementation to create index dimension.

create_specific_variables()
SoilNetCDF specific implementation to create all soil specific variables.

create_variable(*name*, *data_type*, *headers*, ***kwargs*)
SoilNetCDF specific implementation to account for index dimension.

Parameters

- **name** – (str) The name of the variable to be created.
- **data_type** – The data type of the variable to be created e.g. numpy.float32
- **headers** – (list) The name of the columns in the pandas dataframe to use to populate the data of this variable.

- **kwargs** – (dict) Dictionary of attributes {'attr_name': 'attr_value'} to set on the variable e.g. {'standard_name': 'soil_temperature'}

class energy_balance.netcdf.radiation_netcdf.**RadiationNetCDF**(*df*, *qc*, *date*, *frequency*)

Bases: energy_balance.netcdf.base_netcdf.BaseNetCDF

create_radiation_variables()

Create all radiation variables.

create_specific_dimensions()

Create any radiation specific dimensions - there are none.

create_specific_variables()

RadiationNetCDF specific implementation to create all radiation specific variables, including qc variables.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search