ECMWF COPERNICUS REPORT
Copernicus Climate Change Service

Copernicus Climate Change Service Global Land and Marine Observations Database

Fourth version of Technical Service document

Issued by: NUIM / Peter Thorne

Date: 27/01/2021

Ref: C3S_311a_Lot2.3.4.2-2020_202101_Fourth version of Technical_Service_Document.v1

Official reference number service contract: 2017/C3S_311a_Lot2_NUIM/SC2

This document has been produced in the context of the Copernicus Climate Change Service (C3S).

The activities leading to these results have been contracted by the European Centre for Medium-Range Weather Forecasts, operator of C3S on behalf of the European Union (Delegation Agreement signed on 11/11/2014). All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose.

The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission

Contributors

NATIONAL UNIVERSITY OF IRELAND MAYNOOTH (NUIM)

1. Peter Thorne
2. Simon Noone
3. Corinne Voces

NATIONAL OCEANOGRAPHY CENTRE (NOC)

1. David Berry
2. Elizabeth Kent
3. Irene Perez Gonzalez

SCIENCE AND TECHNOLOGY FACILITIES COUNCIL (STFC)

1. Ag Stephens
2. William Tucker

MET OFFICE

1. Robert Dunn

Table of Contents

Executive Summary

The C3S311a Lot 2 (Global Land and Marine Observations Database) service is concerned with the provision of globally available land and marine surface meteorological records. The service includes inventorying of and brokering access to data sources, their harmonization (via conversion to a Common Data Model, merging and quality assurance) and their provision via the Copernicus Climate Change Service Data Store (CDS).

This technical service document describes all relevant aspects of service provision sufficient for ECMWF or a third-party to take over the operation of the service at any point during or upon completion of the service contract. As such, the document contains all information necessary to understand, run and update the set of fundamental data holdings using the methodologies developed and deployed. The document does not constitute a user guide. Instead accompanying user guides are provided for land and marine domains on the respective CDS catalogue entry documentation tabs and any accompanying annexes and materials referenced therein.

This document is a living document which shall be subject to regular revision to reflect the status of the service at any given point in time. Where service capabilities are foreseen, but not yet developed and/or implemented a placeholder title and description of what may be added is provided. Feedback on the adequacy and completeness of the document is welcomed at any time. Feedback should be provided to c3s_311a_lot2_technical_feedback@surfacetemperatures.org which shall distribute the feedback to the core author team.

Former versions are archived and available upon request. The version history is given below:

Version	Release date	Release notes
1.0	31/08/17	Initial version consisting primarily of structural document outline proposed.
2.0	14/12/17	Reflects the initial test release processing.
3.0	12/04/19	Reflects the beta data release.
4.0	24/03/20	Update for first data release.
5.0	27/01/21	Update for second data release

1. Introduction

This service runs on a number of servers hosted on the JASMIN compute platform in the UK (www.jasmin.ac.uk) managed by the STFC Centre for Environmental Data Analysis (CEDA; www.ceda.ac.uk). The overview of this service is provided in Section 2 of this document.

In terms of overall processing, the service aims to:

1. Ingest and archive data from sources in their native format
2. Undertake an inventory of these sources
3. Convert to a common data model
4. Merge the sources accounting for inter-source duplication
5. Undertake quality assurance
6. Provide access to the harmonised holdings
7. Provide externally-facing web-services to provide access to the data/metadata

As such, the service requires a mixture of data management and data processing steps. The data processing software is written in a mix of Fortran, R, Python, SQL and Linux Shell scripts. Although aspects of the service are automated (timely updates, data push) there are other aspects that are by their nature either semi-automated or manual in nature. This technical service documentation covers all of these aspects.

The remainder of this document is structured as follows:

- Section 2 provides an architectural overview of the services and servers.
- Section 3 defines technical specifications and configuration necessary to set-up and run the system operationally.
- Section 4 describes the steps necessary to undertake periodic period-of-record updates and re-issues.
- Section 5 provides contact details for relevant Service personnel.
- Section 6 provides links to additional necessary documentation.

This operational system provides for the possibility to append timely updates and push updated products for provision via the CDS (C3S Data Store). Section 3 presents technical information about setting up and managing these components of the service. All remaining processing steps associated with the service require a degree of manual intervention to be performed. These steps are described in Section 4.

2. System overview and architecture

Although the system architecture has been decided on, its implementation is still under development. Further details remain to be agreed.

2.1 Data deposition server

The data deposition srver facility presents a web-interface through which data providers can connect with the project and upload new datasets for potential inclusion. Data can be pushed to JASMIN by a range of transfer protocols before being manually assessed by the project team. Data can be safely checked and inventoried in this area to ensure against viruses, malware and data corruption prior to ingestion into JASMIN servers. The data deposition server is publicly available at https://datadeposit.climate.copernicus.eu/home/

2.1.1 The purpose of the data deposition server

The service as a whole needs to receive a range of data feeds that can be categorised as either:

- **Internally managed**: project staff arrange and manage data transfers to JASMIN.
- **Externally managed**: other individuals/groups manage data transfers to JASMIN.

Internally managed data feeds can make use of the standard data transfer protocols on to the JASMIN platform because individuals working on them are entitled to login access to the platform. However, externally managed data feeds may be set up by individuals working outside JASMIN. This is desirable because (a) the service does not wish to invite all data providers onto its platform and (b) we want to provide a simple interface to data providers in order to maximise the amount of data that can be brought into the database with minimal effort from staff directly involved with the service. This includes ingest of any data rescued under the C3S 311a Lot1 activity. In some cases, externally managed feeds will eventually by migrated to be managed internally.

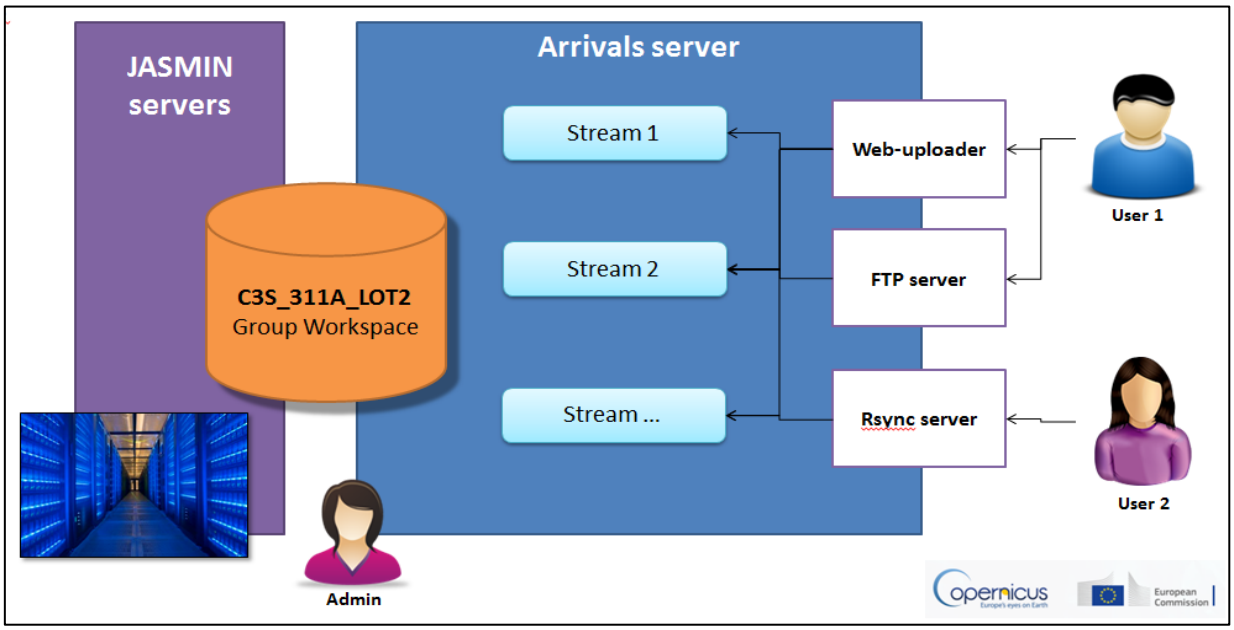
2.1.2 Overview of the data deposition server

The data deposition server enables externally managed data feeds (or *collections*) to be set up without any direct login access to JASMIN. This means that a member of the service team can act as an administrator in setting up accounts on the server so that external users can push their data in both automated and manual ways. The data deposition server provides three different transfer protocols to meet the needs of different users:

1. **Web-upload tool**: a graphical interface that allows upload (and some manipulation) of individual files.
2. **FTP server**: a traditional FTP server that users can either script access to or interact with manually. [Not yet open to public access]
3. **Rsync server**: a more sophisticated transfer protocol/tool that enables recursive and scripted transfer operations along with higher performance. [Not yet open to public access]

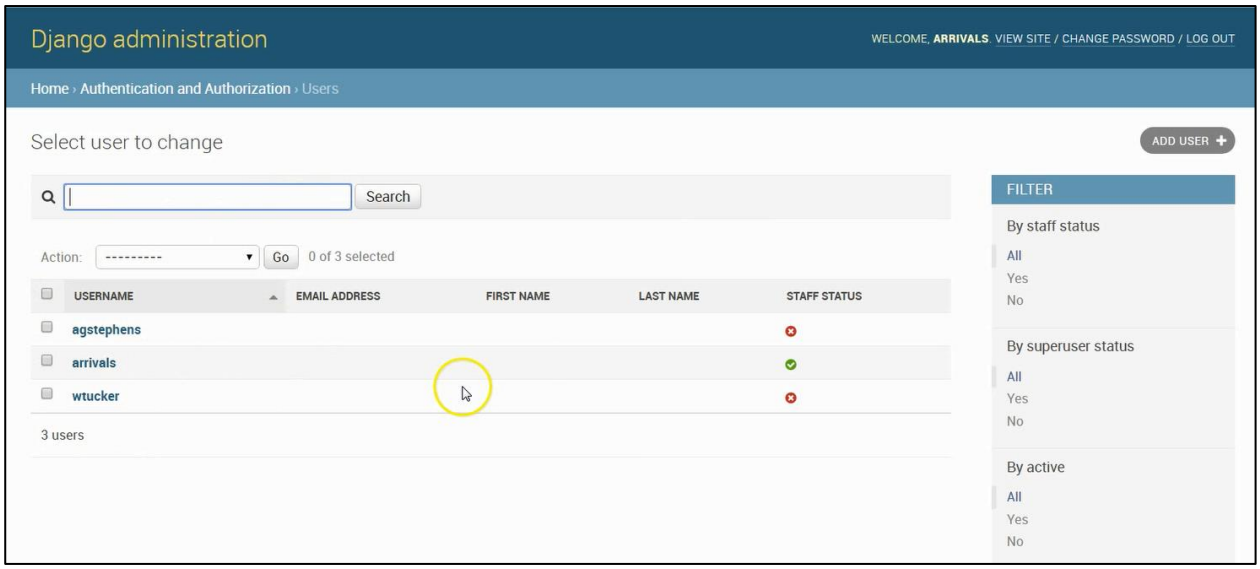
Figure 1 shows a schematic of the data deposition server and its relationship with both the JASMIN platform and the external users.

Figure 1. Data deposition server workflow



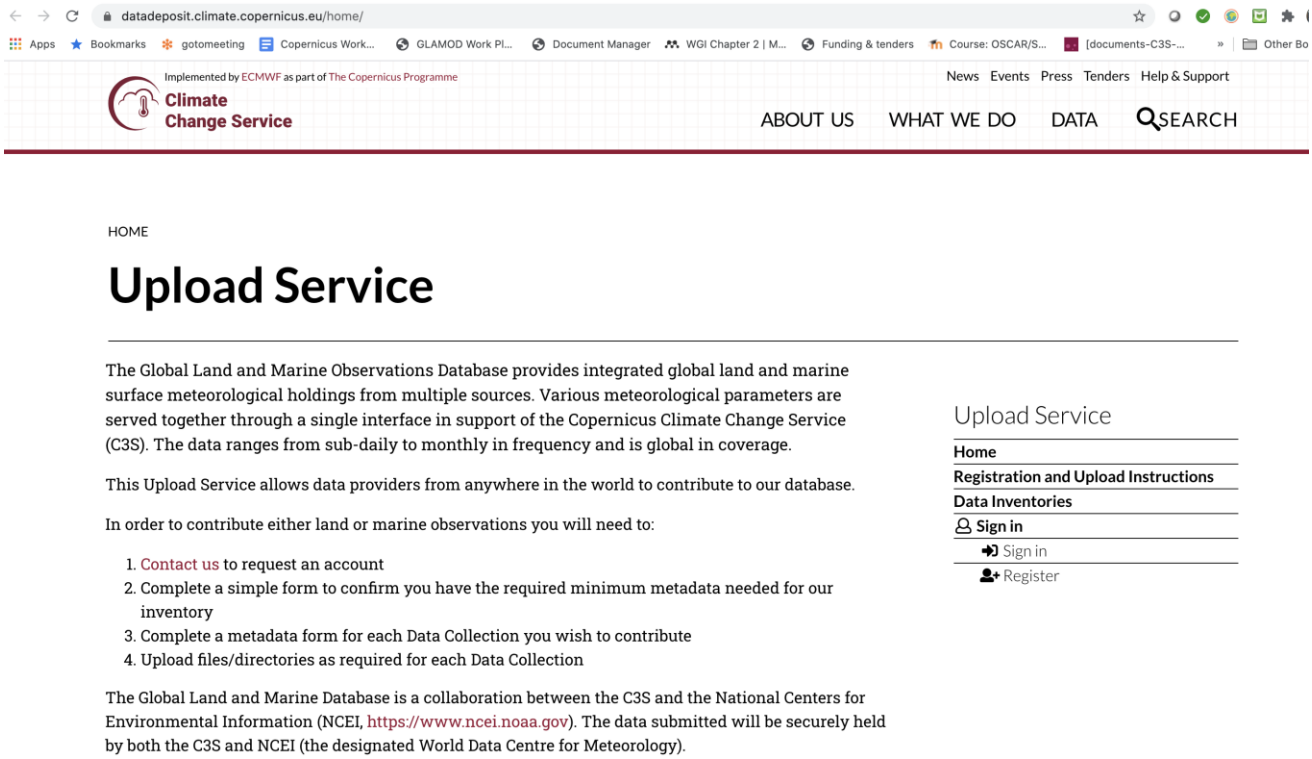
The C3S_311a_Lot2 Group Workspace (GWS) on JASMIN is a shared disk mounted across a number of systems. It is accessible by those working within the C3S_311a_Lot2 service only. This disk is mounted on the data deposition server (Arrivals Server) and the service team *Arrivals Administrator* has access to the Web-uploader with the authority to oversee user accounts, as shown in figure 2.

Figure 2. Web-upload interface for administrators to manage users of the data deposition service.



Users of the service can register via the web-interface and an e-mail will be sent to the administrator, who can authorise access. The user account will be granted permission to write files to a specific directory on the GLAMOD GWS. They can then manually upload files through the web-uploader, as shown in figure 3.

Figure 3. Upload interface of the Arrivals web-uploader.



The user can also request FTP and Rsync access through their login account on the data deposition service. They can then choose the most suitable transfer mechanism for providing their data to the service.

2.1.3 How will data be brought into the service database?

The data ingestion procedure from the Arrivals server is as follows:

1. A directory is created in the GLAMOD GWS for each user.
2. Within that directory the user can set up a "collection".
3. Each collection maps to a sub-directory, which in turn contains a metadata file (containing information provided by the user in a web form).
4. The user can upload any number of files and directories.
5. When the data and metadata is all uploaded, the user clicks "Submit".
6. The directory (and hence the *collection*) is then locked so that no further changes can be made.
7. The Administrator will receive an e-mail detailing the location of the collection.

8. The Administrator will then perform a manual check on the data and metadata and then liaise with the data provider regarding the suitability of adding the dataset into the inventory and database.

2.1.4 How will the Data Provider get feedback on the ingestion process?

Initially, the Administrator will interact with the Data Provider via e-mail. In subsequent versions, this process may become more automated.

2.2 Ingest and processing server

The ingest and processing server receives data from the data deposition server either via automated update scripting or manual command (source dependent) and undertakes conversion to a common intermediate format (domain and timescale dependent) and subsequent checking.

This aspect is presently under development in collaboration with C3S_311a_Lot 1 colleagues and further details shall be given in future iterations of this document once finalised. Initial tests have been completed using a sample dataset provided from the citizen science project “Weather Rescue” led by Prof. Ed Hawkins of Reading University and these data were ingested without issue.

We had received in time for the second data release new data via the data deposition server from three other new data sources since it has come operational see Table 1 for details of data submitted by sources via the data deposition server.

Table 1 shows the data sources that have been deposited the via the data deposition server

Source Name	Timescale	Domain	Data Start/end Years	Variables	Data Policy	Number of stations
The UK Met Office Daily Weather Reports	Sub-daily and Daily	Europe	1899-1910	Temperature, precipitation, pressure, humidity, wind	Open	72
Nanking Meteorological Bulletins	Sub-daily	China	1930-1931	Pressure	Open	52
Deutscher Wetterdienst (DWD) overseas data	Sub-daily	China and South Pacific Islands	1890-1914	Temperature, precipitation, pressure, humidity, wind	Open	74
University of Bern	Sub-daily and daily	Switzerland	1874-2002	Temperature, precipitation, pressure,	Open	70

2.3 Database server

The database server is where the main project databases are held. The PostgreSQL database is installed on a 10TB disk partition and the production database is installed on the server: **glamod2.ceda.ac.uk**.

The 10TB disk partition is part of a large disk array that was purchased specifically to support the kind of performance (in particular an improvement to the available Input/Output Operations Per Second (IOPS)) required by the large database.

It is expected that the database implementation will need to be extended further and a backup database will be housed on a parallel server.

2.4 Front-end server

The main data interface provides an Application Programmable Interface (API) to the contents of the database. This is served through the front-end server. This has the primary role of service data requests from the CDS and possibly other interfaces (such as the C3S Lot 1 Global data registry) to enable serving of the harmonised holdings to C3S customers.

The service is deployed on the same server, **glamod2.ceda.ac.uk**, used to manage the database. We may need to modify this architecture as the data volumes increase.

3. System configuration and operational updates

This section details the system configuration to run the service. In the present architecture there are two production servers and some additional development servers. Only the production system is described here:

Arrivals server: **glamod-arrivals.ceda.ac.uk**

Front-end/database server: **glamod2.ceda.ac.uk**

The servers have the following specification:

Server host	Type	Processors	Memory	Operating System
glamod-arrivals.ceda.ca.uk	Virtual machine	4 CPUs	2GB	CentOS7
glamod2.ceda.ca.uk	Virtual machine	16 CPUs	16GB	CentOS7

The GLAMOD data service provides a Django web-application and PostgreSQL/PostGIS database to support storage and access for data in the Common Data Model (CDM) and the cut-down CDMlite. The components of the service are as follows:

- A PostgreSQL server running a PostGIS database provides storage for the data in the CDM.
- External access to the CDM data is provided through a Django web-application served through an Nginx reverse proxy.

This service runs on the VM: glamod2.ceda.ac.uk

3.1 Dependencies / requirements

The database requires:

- Operating system: CentOS7
- A local PostgreSQL 11 database server running on port 5432
- PostGIS 3.0

The Django cdm-lens service requires:

- Python 3.6+
- Django 2.2.7

The Nginx proxy host has no specific requirements other than the OS, which should be CentOS 7.x.

3.2 Code location

The Ansible playbook for deploying the web application and database is on the CEDA GitLab service is available at: https://breezy.badc.rl.ac.uk/glamod/glamod-playbook [restricted access]

The code is managed in the repositories held under the publicly accessible:

https://github.com/glamod/

Additional configuration information, required to configure and deploy the services, is held in the repositories under the access protected:

https://gitlab.com/glamod/

3.3 Installation and deployment

The deployment process is managed through Ansible playbooks, as follows. Deployment can be carried out on any server that has "root" login access over SSH to the production servers. The playbooks depend on:

- Ansible 2.9+
- Git

The **front-end server** is installed using the following:

```
git clone git@breezy.badc.rl.ac.uk:glamod/glamod-playbook.git
cd glamod-playbook/playbook
ansible-galaxy install -r requirements.yml -f
ansible-playbook -i inventories/production playbook.yml
```

The **data deposition server** is installed using the following:

```
git clone git@breezy.badc.rl.ac.uk:glamod/glamod-arrivals-playbook.git
cd glamod-playbook/playbook
ansible-galaxy install -r requirements.yml -f
ansible-playbook -i inventories/production playbook.yml
```

AIDE, Icinga, Pakiti2 and other monitoring services are installed and managed by the Scientific Computing Department (SCD) who run the JASMIN hardware.

3.3.1 Database configuration

The PostGIS database is installed as part of the above playbook, to: `/var/lib/pgsql`

The main data directory is configured to use the 10TB disk at: `/var/database/data`

Creation of the database

The scripts for creating the CDM-compliant database are available at:

https://github.com/glamod/glamod_database

For the database creation only a single script is required: make_database.sql.

This can be called with:

```
psql -f make_database.sql
```

3.3.2 Nginx Proxy

The Nginx proxy uses a simple reverse proxy configuration created by a template in the playbook. The configuration can be found under: `/etc/nginx/conf.d`

3.4 Visibility

The Django web-application is publicly available on the standard HTTP Port (80) at:

http://glamod2.ceda.ac.uk/v1/

3.5 Logging

Logging for different components of the service are located at:

- Nginx: `/var/log/nginx`
- Tomcat: `/var/log/tomcat`
- Django web-application: `/var/log/cdm-lens/std{out|err}.log`

3.6 Monitoring

Basic service and system checks are performed by Icinga2.

The service manager monitors messages from the LogWatch service and the AIDE log for any suspicious activity. The service manager also monitors backups.

The following files may change during an update, but should not change during normal operation:

- Packages installed via `yum`
- `/etc/sysconfig/iptables`
- Config files under: `/var/lib/tomcats/` and code under: `/usr/local/cdm_lens`

3.7 Patching

The system manager reviews the OS patch level once a week via the Pakiti tool. Pakiti is run by the STFC Scientific Computing Department to patch all servers on JASMIN.

3.8 Backups

3.8.1 Data backups

A policy is being developed for providing backups. There are two distinct types of backup:

1. Backups of raw source decks
2. Backups of database tables

Backups of raw source decks

The backups of the raw source decks are addressed by running a manual script on an agreed timescale. The script carries out the following functions:

1. Takes a directory as input.
2. The directory is root-locked so no changes can be made to it whilst backing up.
3. The directory is renamed with a current date suffix ("_YYYYMMDD")
4. Each sub-directory is scanned and the contents recorded.
5. Each sub-directory is copied to the Elastic Tape system on JASMIN
6. On completion of the backup:
    a. The contents of the backup are recorded in a file manifest
    b. The backed-up directory is renamed to its original name

c. The original ownership is reinstated so that the contents can be modified again

Backups of database tables

At the time of writing the database backup system is not yet in production. The system will involve two versions of the full database running in parallel. PostgreSQL replication tools will enable the backup version to be synchronised with the production database. More information on the method will be included here in the next version of this document.

3.8.2 Machine configuration back-ups

The machine is covered by a 4-week cycle of tape backups in the Atlas tape store. Configuration is held in `/etc/backup/all.params` and run via the `/etc/cron.daily/hpcbackup` cron job. The state of the tape backup can be monitored at: http://bart.esc.rl.ac.uk/backups_report.html [Internal to STFC].

The tape backups of the OS are primarily for security forensic analysis.

3.8.3 Service backups

The applications can all be rebuilt from the Ansible playbooks so no application-specific backups are made.

3.9 Starting and stopping services

Nginx is managed using the service command:

```
systemctl status|start|stop|restart nginx
```

The Django web-application is managed using the service command:

```
circusctl status|start|stop|restart cdm-lens
```

PostgreSQL is managed using the service command:

```
systemctl status|start|stop|restart postgresql-11
```

3.10 Updating the service

To update the database server, WFS and proxy, just run the Ansible playbook as above. The Ansible playbook is designed to be idempotent (meaning that the outcome will be the same if executed once or multiple times) and will only make the required changes.

3.11 Automated system management

3.11.1 Land system updates

Land system updates are presently undertaken for daily records with updates attempted daily. The process starts with the execution of a script that attempts to download the new daily differences [superghcnd_diff_yyyymmdd_to_yyyymmdd.tar.gz file] for GHCND which are currently updated around 3.30pm GMT on the NCEI ftp data access web page (ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/).

This is achieved via a time-based job scheduler CRON job written in Linux on JASMIN group workspace. The FTP download CRON job is set to run daily at 7.25pm GMT and executes a Linux code that downloads the daily GHCND diff file if there is a new one available. The code reads a Last_file.txt file which contains the previous downloaded file and if the last .tar.gz file is different from this file name the code proceeds with the download. The code the updates the Last_file.txt with the new file name ready for the next day's download.

If the Cron job successfully retrieves a new update the next Cron job is set to run at 7.35pm GMT which executes a python code to decompress the GHCND diff .tar.gz file.

There are three files contained in each.tar.gz file:

- update.csv: contains changes to values or flags that were present on yyyymmdd1 (i.e., these values or flags have been altered between yyyymmdd1 and yyyymmdd2)
- insert.csv: contains values that were new on yyyymmdd2 (i.e., that were not yet available on yyyymmdd1, but were newly available on yyyymmdd2)
- delete.csv contains values that were present on yyyymmdd1, but not on yyyymmdd2

We are currently only using the insert file to update the current release with new data values. The regular full data refresh will make the other changes such as deletions and changes etc. to the data. The average insert file contains daily observations for around 15,000 -19,000 stations containing snow, wind, precipitation and temperature observations.

Next Python code is executed at 8.20pm GMT to remove the original .tar.gz file from the directory.

An email is sent to Simon Noone informing him of the status of the daily update progress.

Table 2 The daily update CRON job process

First stage of CRON Job	###ftp_download_ghcnd_diff MAILTO="simon.noone@mu.ie" 25 19 * * * /home/users/sjnoone/bin/diff_down_new ### end
executes a bash script to submit the job to SLURM batch computing on LOTUS	###ftp_download_ghcnd_diff MAILTO="simon.noone@mu.ie" 25 19 * * * /home/users/sjnoone/bin/diff_down_new ### end #!/bin/bash #SBATCH -p short-serial #SBATCH -o %J.out #SBATCH -e %J.err #SBATCH -t 04:00:00 source /home/users/sjnoone/venv/bin/activate python /home/users/sjnoone/ghcnd_ftp_updates_dy/FTP_download_ghcnd_diff_scratch.py
Second stage of CRON job	### run convert file process on Lotus MAILTO="simon.noone@mu.ie" 35 19 * * * /home/users/sjnoone/bin/unzip_diff_new ##end
executes a bash script to run a Python script to untar the file.	#!/bin/bash source /apps/jasmin/jaspy/miniconda_envs/jaspy3.7/m3-4.6.14/bin/activate jaspy3.7-m3-4.6.14-r20200606 python /home/users/sjnoone/ghcnd_ftp_updates_dy/unzip_diff_new.py ##end
Final stage of CRON job	### run remove /gz MAILTO="simon.noone@mu.ie" 30 20 * * * /home/users/sjnoone/ghcnd_ftp_updates_dy/remove_gz ##end

Which executes a bash script to remove the original .tar.gz file from the directory	#!/bin/bash rm /gws/nopw/j04/c3s311a_lot2/data/level0/land/daily_data_processing/superghcnd_daily_updates/*.gz remove_gz ## end

The subsequent executable Linux and Python code for each CRON diff download process can be found at: https://github.com/glamod/glamod-nuim/blob/master/CRON_daily_diff_update.zip

The conversion to CDM formatted files is then achieved via fortran routines. This Fortran construction uses a parent Fortran program to call two daughter programs to separate the GHCND update files into a collection of station files, and then to convert the station files into CDM format. The parent program assesses which GHCND update file has to be processed next and then calls two Fortran daughter programs in sequence. The first daughter program separates the GHCNd file into station files, and the second converts the station files to CDM format. The program is run from the Jasmin Linux command line with ‘./cm2bat to generate the executable aa.exe and then ‘./aa.exe’ to start the program. This will process one GHCND update file from start to finish. The program was mainly used in a Lotus shell script (‘z_lotus_runner’) with a loop that processed groups of 18 files with a 1.5 hour separation time over a period of 24 hours. The program structure with input and output files are summarized in Tables 3 to 5.

Table 3. Summary of Fortran code structure for overall CDM conversion of daily updates from GHCND

Linux script	cm2bat to produce executable aa.exe
Main program	cm2.f
Subroutine structure	cm2.o Subroutine/readin_list_source.o Subroutine/readin_list.o Subroutine/decide_directory20200708.o Subroutine/make_export_names.o Subroutine/export_source_directory_path.o

	Subroutine/export_target_directory_path20200710.o [spawns Fortran program a.exe to do GHCND station separation] [spawns Fortran program b.exe to do CDM conversion]

Table 4. Summary of main input files for overall CDM conversion of daily updates from GHCND

Input file and directory	Handling subroutine	Description
List_source/list_source.dat	readin_list_source.f	Reads in all available GHCND update files
List_completed_obs/ list_completed_obs.dat	readin_list_source.f	Reads in list of GHCND CDM files that have been completed

Table 5. Summary of main output files for overall CDM conversion of daily updates from GHCND

Output file and directory	Generation subroutine	Description
List_source/ list_source.dat	cm2.f	A Unix script is called to create file with a list of the GHCND update files held in /gws/nopw/j04/c3s311a_lot2/data/level0/ land/daily_data_processing/superghcnd_daily_updates/
List_completed_obs/ list_completed_obs.dat	cm2.f	A Unix script is called to create file with a list of the processed GHCND update files held in /work/scratch_pw/akettle/P20200708_ghcnd_dayupdate2/ Step2_makecdm/observations_tables_test/
../Step1_stnsep/ Source_directory/ source_dir.dat	Export_source_directory_path.f	This creates a single line file that identifies which GHCND update file to do the station separation on.
../Step2_makecdm/	Export_target_directory_path20200710.f	This specifies the file names for the CDM output files (header, observation, light),

Target_directory/ targetdir.dat		the associated receipt file, and an administrative file gives the run of the process.

The first daughter program separates GHCND data files into station files. The program is run from the Jasmin Linux command line with ‘./ssu1bat’ to generate the executable a.exe and then ‘./a.exe’ to start the program. The program structure with input and output files are summarized in tables 6 to 8 below.

Table 6. Summary of Fortran code structure for separation into station files

Linux script	ssu1bat to produce executable a.exe
Main program	ssu1.f
Subroutine structure	ssu1.o Subroutine/import_source_directory_dailyupdate.o Subroutine/find_subdir_stn_output.o Subroutine/read_split_data1.o Subroutine/get_stn_name.o Subroutine/output_group_lines.o Subroutine/file_progress_update.o

Table 7 Summary of main input files for separation into station files

Input file and directory	Handling subroutine	Description
/home/users/akettle/Work/P20200708_ghcnd_dayupdate2/ Step1_stnsep/Source_directory/ sourcedir.dat	import_source_directory_dailyupdate.f	Reads in the GHCNd update directory to process
/gws/nopw/j04/c3s311a_lot2/data/level0/land/ daily_data_processing/superghcnd_daily_updates/	read_split_data.f	General designation of the directory path of the GHCNd update data files.

superghcnd_diff_????????_to_????????/ insert.csv		

Table 8 Summary of main output files

Output file and directory	Generation subroutine	Description
/work/scratch-pw/akettle/P20200708_ghcnd_dayupdate2/Step1_stnsep/ ???????????.csv	output_group_lines.f	Outputs the data lines into station files with 11-character names.

The final daughter program converts these to CDM format. The program is run from the Jasmin Linux command line with ‘./gu2_bat’ to generate the executable a.exe and then ‘./b.exe’ to start the program. The program structure with input and output files are summarized in Tables 9 to 11.

Table 9. Summary of Fortran code structure to convert to CDM format

Linux script	gu2_bat to produce executable b.exe
Main program	gu2.f
Subroutine structure	gu2.o Subroutine/read_new_output_directories20200711.o Subroutine/readin_lastfile_number.o Subroutine/erase_files_seq20200711.o Subroutine/create_receipt_file20200710.o Subroutine/readin_source_id2.o Subroutine/get_stnconfig_ghcnd2.o Subroutine/get_elements_stnconfig_line2.o Subroutine/readin_subset.o Subroutine/main_cycler20200710.o Subroutine/get_singles_stnconfig_v5.o Subroutine/export_errorfile.o

	Subroutine/export_file_simple.o
	Subroutine/input_output_single_station20200712.o
	Subroutine/get_lines_1file2_special.o
	Subroutine/get_fields_from_ghcnd_line2_special.o
	Subroutine/find_len_fields2.o
	Subroutine/verify_stn_id2.o
	Subroutine/find_hist_param_elements2.o
	Subroutine/find_distinct_timesteps2.o
	Subroutine/get_sourceid_numcode2.o
	Subroutine/get_source_id_single2.o
	Subroutine/get_record_number2.o
	Subroutine/search_stringfragment2.o
	Subroutine/ambigcase_get_low_record2.o
	Subroutine/total_counts_recordnumber2.o
	Subroutine/convert_string_to_float_ghcnd3.o
	Subroutine/convert_variable_column2.o
	Subroutine/convert_float_to_string2.o
	Subroutine/convert_integer_to_string2.o
	Subroutine/test_precision_variables2.o
	Subroutine/precision_single_vector1a.o
	Subroutine/precision_single_vector2a.o
	Subroutine/find_hist_dailyprecision2.o
	Subroutine/precision_interpreter_tenths2.o
	Subroutine/precision_interpreter_whole2.o
	Subroutine/header_obs_lite_table20200712.o
	Subroutine/sort_vec_obs_ancillary_ghcnd2.o
	Subroutine/find_vector_recordnumber2.o
	Subroutine/assemble_vec_original_precision2.o
	Subroutine/get_numerical_precision2.o
	Subroutine/get_quality_flag_vector2.o
	Subroutine/qc_single_variable2.o
	Subroutine/get_observationvalue_vector2.o

	Subroutine/convert_float_to_string_single2.o Subroutine/get_hght_obs_above_sfc.o Subroutine/get_strvector_distinct.o Subroutine/export_stnconfig_lines20200713.o Subroutine/export_last_index.o Subroutine/finish_receipt_file20200710.o

Table 10. Summary of main input files to convert to CDM format

Input file and directory	Handling subroutine	Description
/work/scratch-pw/akettle/ P20200708_ghcnd_dayupdate2/ Step1_stnsep/ ???????????.csv	get_lines_1file2_special.f	GHCNd station data in the NCEI ASCII format. The station names have the 11 character NCEI identifier format.
/home/users/akettle/Work/P20200304_ghcnd20/ Step2_cdmmake_20200410/Data_ancillary/ GHCNdSource_C3Ssource_IDS_20191101.psv	readin_source_id2.f	File with conversion key between NCEI letter codes and SN number codes.
/home/users/akettle/Work/P20200304_ghcnd20/ Step2_cdmmake_20200410/Data_stnconfig/ daily_station_config_file_30_07_20.psv.csv	get_stnconfig_ghcnd.f	Station configuration file sent by SN on 30 July 2020
/home/users/akettle/Work/P20200304_ghcnd20/ Step2_cdmmake_20200410/Data_ancillary/ f20200406_station_list_GHCND.csv	readin_subset.f	File from SN with subset of the station configuration list.
/home/users/akettle/Work/ P20200708_ghcnd_dayupdate2/ Step2_makecdm/Target_directory/ targetdir.dat	read_new_output_directories20200711.f	Reads in output file names for the CDM files (observation, header, lite), the receipt file, and runtime administration file.

Table 11. Summary of main output files to convert to CDM format

Output file and directory	Generation subroutine	Description
/work/scratch-pw/akettle/ P20200708_ghcnd_dayupdate2/Step2_makecdm/ observations_tables_test/ obs_????????_to_????????.psv	header_obs_lite_table20200712.f	CDM observation table
/work/scratch-pw/akettle/ P20200708_ghcnd_dayupdate2/Step2_makecdm/ header_tables_test/ header_????????_to_????????.psv	header_obs_lite_table20200712.f	CDM header table
/work/scratch-pw/akettle/ P20200708_ghcnd_dayupdate2/Step2_makecdm/ cdm_lite_test/ lite_????????_to_????????.psv	header_obs_lite_table20200712.f	CDM lite table
/work/scratch-pw/akettle/ P20200708_ghcnd_dayupdate2/Step2_makecdm/ receipt_test/ receipt_????????_to_????????.psv	header_obs_lite_table20200712.f	Receipt file
/home/users/akettle/ Work/P20200731_ghcnd_dayupdate3/ Step2_makecdm/Directory_runtime/ runtime_????????_to_????????.psv	create_receipt_file20200710.f, finish_receipt_file20200710.f	Administrative file to verify that processing GHCNd day file completed successfully and to check the running time

These data are not, as yet, then operationally ingested into the publicly facing holdings, but this will be achieved in upcoming releases.

3.11.2 Marine system updates

Here we shall describe how and how frequently marine system updates are performed. Regular data appends are envisaged to start at the earliest only with the final release under the present contract and are dependent upon progress by the upstream ICOADS data provider.

4. Periodic updates

This section is concerned with how to perform periodic updates (major new releases) that require a suite of manual or semi-automated steps. The Section is ordered in the same order as the sequential processing steps. It is not essential to perform precursor steps in this sequence prior to a given step. For example, a change in harmonisation technique can be applied without a commensurate reconsideration of the available data inventories or the CDM.

4.1 Inventory preparation

The first step in both land and marine processing is the ingestion and creation of an inventory of existing holdings. This inventory summarises key facets of the observational records made available from a range of sources. The inventory files and resulting structure are distinct for land and marine reflecting the fundamental distinctions in methods of observation. Presently all data ingest has been performed manually from public-facing servers or via hard-disk transfer.

4.1.1 Land inventory

The archive of land sources is maintained at `{<c3sbase>/data/level0/land}`. Each data source is considered to constitute a data deck and each deck has its own sub-directory.

The land inventory consists of a parent file that contains information on all archived data decks. Then within each data deck there is a station level inventory. All inventory files of a given timestep (sub-daily, daily or monthly) contain the same information.

```
<c3sbase>/data/level0/inventory/
```

```
<c3sbase>/data/level0/land/monthly_data/
<c3sbase>/data/level0/land/daily_data/
<c3sbase>/data/level0/land/sub_daily_data/
```

The source deck inventory has 29 columns and these tables are filled in manually. All column headers and descriptions are presented in Section 3.1 of the Land User Guide. Column headers for the source deck inventory and the corresponding explanations can be found in Table 1 of the appendix in the Land User Guide. We have produced a station deck inventory format that is compliant with international discovery metadata standards. Full details can be found in Section 3.2 of the Land User Guide. New sources found need to undergo the above inventory activities. Aspects of the task are aided by scripting. However, because most sources use distinct formats, in general new read routines will be required to be written for new sources. The read routines used to compile the land based data inventories are written in open source programming languages. All read routine code are saved in the following directory on the JASMIN server `(<c3sbase>/code)`. The code is also shared via the Github repository accessible at https://github.com/glamod/glamod-nuim.

The majority of the inventory compiling work is conducted manually. However, below is the list of R code that has been used to perform some re-formatting actions on two data sources (ISTI and GHCNd) to aid in the compilation of some information for the inventory. The R code are named separately and have inline text within the code explain each step. All the R code can be executed using R version 3.6.1 (2019-07-05).

Add_headers_ISTI_inventory_files.r

Read_inventory_file_ ISTI_online_daily_files.r

Read_inventory_file_ISTI_online_monthly_files.r

read_ GHCN_inventory_file.r

The code can be found at: https://github.com/glamod/glamod-nuim/blob/master/Inventory_R_code_ISTI_GHCND.zip

In addition, there is Python 3.7 code that extracts some inventory information from data sources that have been converted to the Intermediate File Format (IFF) and the code is not source specific. The Inventory_code.py also has inline text to explain each step and is executed when any new source has been converted to the IFF. All the Python 3.7 code can be executed by using Anaconda–Spyder scientific environment or any Python integrated development environment.

The code can be found at: https://github.com/glamod/glamod-nuim/blob/master/inventory_code.py

In all the cases so far, the original source has provided some data information in either document, text file or csv file format which outlined an inventory of the station data for that source. The information was processed by both manual and semi-automated methods to reformat the information to be consistent with the Lot 2 land inventory. The reformatted information was then manually inputted into the relevant Lot 2 land inventory files. A visual check was also conducted with each original source inventory following input into the Lot 2 land inventory files. Land inventory data information folders have been provided for each of the source decks and contain the individual original source data inventory, inventory information re-formatting code, and readme documents explaining how each source deck inventory was compiled and re-formatted. In addition, original source information such as other data readme documents, data policy tables and any metadata that was available are provided. These information folders are available in the following directory on JASMIN:

```
<c3sbase>/data/level0/inventory/all_land_inventory_data_information_folders/
```

The following elements are extracted from the original source data files where available and after re-formatting the information is transferred to the parent source deck inventory and relevant station deck inventory. Not all the following information may be available, but the items that are essential are Station Id or Station name, Latitude/Longitude and elevation.

- Station name
- Station Id
- WMO Id
- Latitude/Longitude
- Elevation
- FIPS Code
- Country
- Continent
- Climate variables available
- Data Start and End years for each variable available

The available data information elements listed above when produced need to be added to the parent source deck inventory and the relevant timescale station deck inventory. Once this has been achieved the updated version of the inventories and the data are moved from the incoming directory into the following directory and relevant sub directory for processing:

```
<c3sbase>/data/level0/inventory/
```

```
<c3sbase>/data/level0/land/monthly_data/
<c3sbase>/data/level0/land/daily_data/
<c3sbase>/data/level0/land/sub_daily_data/
```

4.1.2 Marine inventory

The initial marine data holdings come from a single source, Release 3.0.0 of the International Comprehensive Ocean Atmosphere Data Set (ICOADS) (Freeman et al., 2017). This dataset is the most complete archive of surface marine observations and contains input data from many different sources in a harmonized format (the International Maritime Meteorological Archive (IMMA1) format). Release 3.0.1, the near real time updates to ICOADS, will also be processed within this service. Hereafter, we will refer Release 3.0.0 and 3.0.1 together as R3.

Due to the many different sources ingested into ICOADS, duplicates are present in the complete archive used to generate R3. This complete archive is commonly referred to as the ICOADS R3 "total" files, hereafter R3t. Prior to general release the R3t files undergo basic processing to form the 'final' release files (R3f) described in Freeman et al. (2017). This processing includes limited quality flagging, duplicate elimination and merging of records. Due to choices made, both within the ICOADS processing and in the original source data, reports from a single platform or ship may come from multiple sources in the R3f files when ideally they would come from a single source. Within this service this issue will be addressed with many of the processing options used in the generation of the R3F files reassessed. As such, the R3t form the initial data holdings and have been inventoried.

Within the IMMA format two fields describe the original source and type of data – the source ID (or SID) and deck (DCK). The deck field generally describes the type of data stored within the record. For example deck 143 contains data for "Pacific Marine Environmental Laboratory (PMEL) Buoys". These data can come from multiple sources given by the SID flag. For example the source can be a real time data stream extracted from the WMO global telecommunications system (GTS) or delayed mode data from the appropriate data assembly centre. Typically data from a single source will have come from a single common format prior to conversion to IMMA1.

To understand the data within ICOADS R3t, the monthly files have been split by SID and DCK. These individual files have then been inventoried to provide information on the spatial and temporal extent of the data, characteristic reporting precisions and availability of the following variables: sea level pressure; air temperature; humidity; sea surface temperature; wind speed; cloud cover; sea state statistics (indicated by wave height); and present weather. An example of the summary information can be found in Figure 4 - the location of the observations and observing density summed across the

period of record are shown as a map. The reporting precisions for the location and time information are shown as histograms. Similarly, temporal histograms are used to show the type of data in each SID/DCK combination and availability of the above fields. This information has also been summarised as a tab separated file, with one row per SID/DCK combination.

As with the land data holdings, the marine data are stored on JASMIN. The paths to the original IMMA files are:

```
<c3sbase>/data/level0/marine/sub_daily_data/IMMA1_R3.0.0T/
<c3sbase>/data/level0/marine/sub_daily_data/IMMA1_R3.0.1T/
```

for Release 3.0.0 and 3.0.1 respectively. `<c3sbase>` indicates the base directory on JASMIN for C3S 311a Lot 2, currently `/gws/nopw/j04/c3s311a_lot2/`. There are no daily or monthly summary files for the marine data. R3f data have been processed to give one set of files for each SID/DCK combination in the IMMA format. The paths to the processed files are:

```
<c3sbase>/data/level0a/marine/sub_daily_data/IMMA1_R3.0.0T/<sid>-<dck>/
<c3sbase>/data/level0a/marine/sub_daily_data/IMMA1_R3.0.1T/<sid>-<dck>/
```

Where `<sid>` and `<dck>` are the 3 digit numbers indicating SID and DCK padded with leading zeros when required. The records for the different source and deck combinations have been extracted using the R script extract_siddck.R. A version of this script is stored within each release directory (e.g. IMMA1_R3.0.0T). The usage for the script is given by:

```
> extract_siddck.R <fileNumber>
```

Where the leading `>` indicates the command line on JASMIN. The script reads in a potential list of files to process from the file `filelist.txt` located in the working directory. The argument `<fileNumber>` indicates which file to process. Multiple files can be processed simultaneously using the LOTUS cluster at JASMIN, with multiple jobs submitted via the LSF scheduler. A version of the submission script, `submit_extraction.sh`, is located in each release directory. New files can be processed by updating the `filelist.txt` file and job numbers in the submission script.

For each release, and source and deck directory contains a pdf file visually summarising the data within that directory. These have been created using three R scripts:

<c3sbase>/data/level0a/marine/sub_daily_data/src/read_imma.R
<c3sbase>/data/level0a/marine/sub_daily_data/src/plot_for_inventory.R
<c3sbase>/data/level0a/marine/sub_daily_data/<release>/drive_plot_all.R

<release> Indicates the ICOADS release directory. The first script contains a function to read the required variables from an IMMA file, the second generates the pdf plot. The third, drive_plot_all.R, acts as a wrapper to the first two scripts and drives the plotting functions. This script can be called from the command line:

```
> drive_plot_all <directoryNumber>
```

A list of potential directories to process is read in from the file dirs.txt located in the current working directory. The argument <directoryNumber> indicates the directory to process. As with the extraction code, an LSF scheduler job submission script, submit_plots.sh, is located in the same directory. New sources and decks can be processed by updating the directory list and job submission scripts. To aid ease of use, the pdf files have been combined into a single pdf for each release:

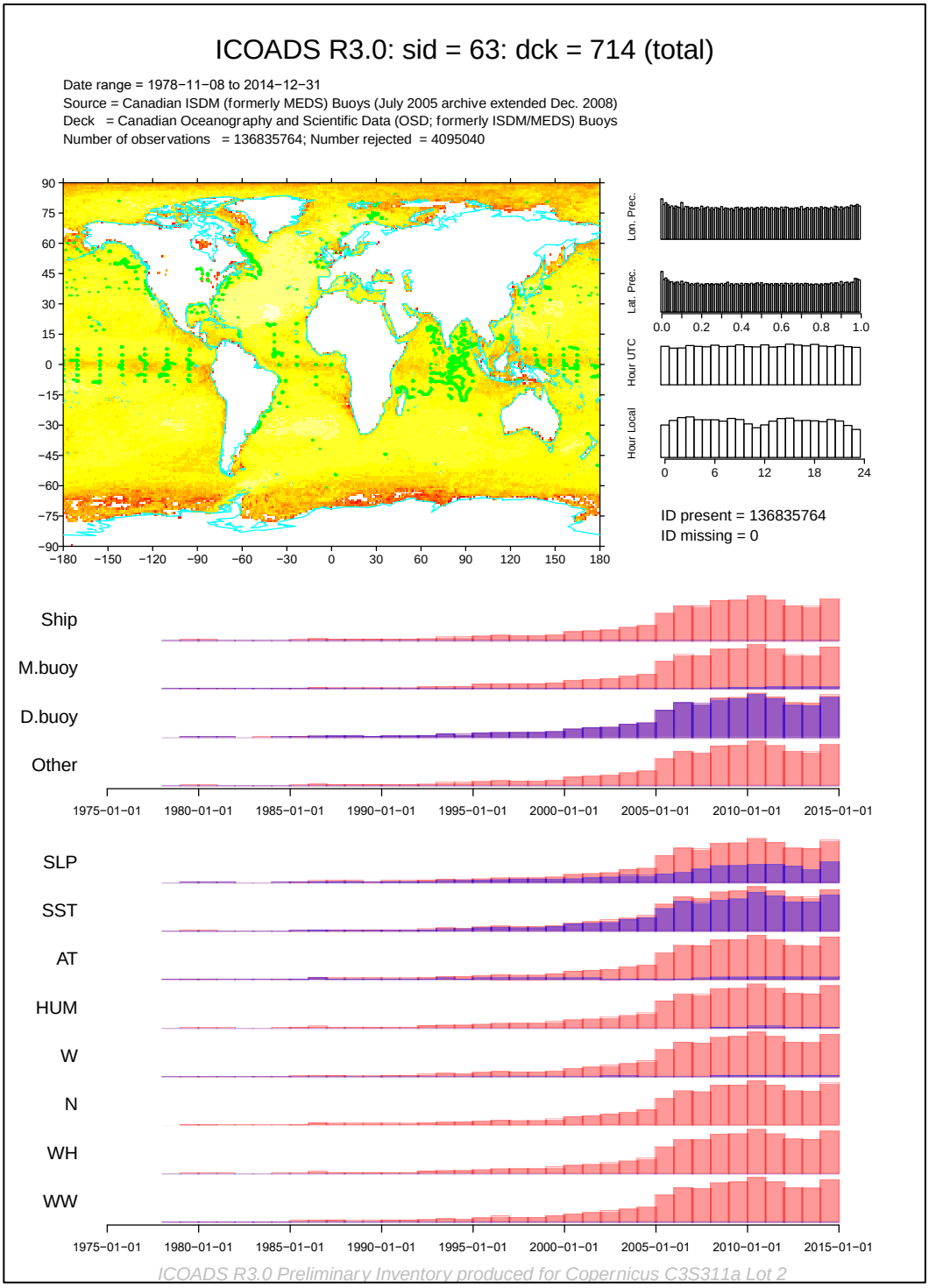
<c3sbase>/data/level0a/inventory/marine/sub_daily/ICOADS_R3.0.0T_inventory_total.pdf
<c3sbase>/data/level0a/inventory/marine/sub_daily/ICOADS_R3.0.1T_inventory_total.pdf

As part of the process one text file per source / deck combination is created summarising the information textually. These have been concatenated to give one inventory file for each release:

<c3sbase>/data/level0a/inventory/marine/sub_daily/ICOADS_R3.0.0T_inventory_total.tsv
<c3sbase>/data/level0a/inventory/marine/sub_daily/ICOADS_R3.0.1T_inventory_total.tsv

As the service is developed the scripts described above will be modified to be more generic and moved under version control within the GLAMOD GitHub git repository.

Figure 4. Example summary plot for marine inventory showing: log density plot of number of observations per 1° grid cell and location of moorings (green dots) and coastal stations (blue dots). Also shown is the reporting precision for location and time variables (top right). Time series showing the proportional make-up of the observations from ships, buoys and others (purple) relative to total number of observations (red) are shown in the middle. The bottom panel shows the proportion of reports with different variables.



4.2 Harmonisation

4.2.1 Merging source decks

Land merge

Here we describe how we perform aspects around the land merge for the sub-daily scale including inter-alia source deck prioritisation, metrics for assessing station similarity, conversion of source file formats to a standardized data file format and how to backfill periods or additional ECVs. This includes information on how to assign ids and in what order of temporal averaging the merge should be performed. It outlines the necessary manual intervention steps and the current merge priority order. The merge for daily and monthly releases is performed by NOAA NCEI for GHCND and documented in their documentation.

Converting sub-daily land data sources from original formats to standardized file formats

Most of the land data sources have many different data formats which makes it very difficult and time consuming to merge into the harmonised data if left in their current format. Therefore, once a data source has been prioritised for merging, we first convert the original file format to an Intermediate File Format (IFF). The IFF format allows for the data source to be merged efficiently. The IFF consists of one variable per source with all necessary station metadata. Table 1 shows all column headers and corresponding descriptions. The variable units are converted from original units to the following units for the IFF:

- temperature (degrees C)
- dew_point_temperature (degrees C)
- wind_direction (degrees)
- wind speed (meters per second)
- sea level pressure (hPa hectopascals)
- station_level_pressure (hPa hectopascals)

The IFF file is saved as a pipe [|] separated file ext = [.psv] one single variable per file.

The IFF files are saved in Level1/Land/ directory on the JASMIN workspace in the relevant Level1a timescale sub-directory, in the relevant variable specific sub-directory, e.g

/gws/nopw/j04/c3s311a_lot2/data/Level1/Land/level1a_sub_daily_data/sea_level_pressure/245

The IFF is named by using the station ID - record number (for multiple station configurations) combined with the variable name (exactly as defined below) combined with source ID using _ separators e.g IE9999-1_sea_level_pressure_245.psv

The following data sources have been converted to IFF from the USAF meta-source using Fortran conversion codes:

220 - WMO_USAF_Sub_Daily_data_Global

221 - AFWA_USAF_Sub_Daily_data_Global

223 - ICAO_USAF_Sub_Daily_data_Global

First the USAF source files which are timeslice collections requires disaggregation into station files. The program is run from the Jasmin Linux command line using './st1bat' to generate the executable a.exe, followed by './a.exe' to run the program. Tables 10-12 outline the software structure, required input files, and the output files. The raw USAF data files contained very few instances of bad data lines with malformed date/time stamps, and these were not passed to the reconstructed station data listing. There were other instances of very long lines over 4000 characters in width (where the normal line width was on the order of 500 characters). These lines were added to the reconstituted station list but truncated at 4000 characters.

Table 12. Summary of Fortran code structure for reconstructing USAF station files that are ordered in time.

Linux script	st1bat to produce executable a.exe
Main program	st1.f
Subroutine structure	st1.o Subroutine/get_ascii_list2.o Subroutine/get_filename_directories3.o Subroutine/find_start_index2.o Subroutine/file_eraser_index2.o Subroutine/write_usaf_summary3.o Subroutine/write_index_archive2.o Subroutine/read_file_datalines4.o

	Subroutine/time_processor2.o Subroutine/std_datetime_fmt2.o Subroutine/str_to_dt_pvwave2.o Subroutine/line_exporter3.o Subroutine/find_firstlast_time2.o Subroutine/export_header_titlelist2.o Subroutine/export_line_overlength2.o Subroutine/test_component_char2.o Subroutine/export_line_bad_datetime2.o Subroutine/summary_exporter3.o Subroutine/test_new_name2.o

Table 13. Summary of main input files.

Input file and directory	Handling subroutine	Description
Data_in/list_sfc_obs_files.dat	get_filename_directories3.f	File with listing of raw ASCII USAF files
/gws/nopw/j04/c3s311a_lot2/data/incoming/usaf/????/usaf-swo-03_????????_????????_sfc_obs.csv	read_file_datalines4.f	Individual raw ASCII USAF files

Table 14. Summary of main output files.

Output file and directory	Generation subroutine	Description
/gws/nopw/j04/c3s311a_lot2/data/level1/land/zscratch_ajk3/L1_lastfile/list_columnheaders.dat	export_header_titlelist2.f	Simple listing of the column headers in a USAF ASCII data file.
/gws/nopw/j04/c3s311a_lot2/data/level1/land/zscratch_ajk3/L1_lastfile/list_overlength.dat	export_line_overlength2.f	File containing USAF data lines with over 4000 characters width

/gws/nopw/j04/c3s311a_lot2/data/level1/land/zscratch_ajk3/L1_lastfile/list_bad_datestamp.dat	export_line_bad_datetime2.f	File containing lines with bad date/time strings; i.e., any date/time stamp with less than 14 characters
/gws/nopw/j04/c3s311a_lot2/data/level1/land/zscratch_ajk3/L1_stnfiles_notsort/*.csv	line_exporter3.f	Station files named by network type and platform id with an unsorted list of data lines from the original USAF files.
/gws/nopw/j04/c3s311a_lot2/data/level1/land/zscratch_ajk3/L1_lastfile/usaf_summary_file.dat	Write_usaf_summary3.f	List of summary information about the USAF ASCII source files

Next it is necessary to sort the separated USAF station files by timestamp. This program sorts the USAF netplat station lines in a time-ordered sequence. The software is run from the Jasmin Linux command line with ‘./so1bat’ to generate the a.exe executable followed by ‘./a.exe’ to run the program. Tables 13-15 outline the program structure and the input/output files.

Table 15. Summary of Fortran code structure for time sorting the separated USAF station update files.

Linux script	so1bat to produce executable a.exe
Main program	so1.f
Subroutine structure	so1.o Subroutine/create_data_lists2.o Subroutine/readin_namelist2.o Subroutine/get_network_istart.o Subroutine/open_close_filelist2.o Subroutine/get_single_list_data2.o Subroutine/find_distinct_usaf_index2.o Subroutine/find_stats_linelength_commas.o Subroutine/sort_line_list3.o

	Subroutine/std_datetime_fmt2.o Subroutine/str_to_dt_pvwave3.o Subroutine/sort_doublelist_pvwave2.o Subroutine/export_sorted_list2.o Subroutine/export_level2_summaryline.o Subroutine/export_l2_lastline.o

Table 16. Summary of main input files for sorting the separated USAF files.

Input file and directory	Handling subroutine	Description
Data_lists/list_wmo.dat	readin_namelist2.f	Listing of USAF WMO stations
Data_lists/list_icao.dat	readin_namelist2.f	Listing of USAF ICAO stations
Data_lists/list_afwa.dat	readin_namelist2.f	Listing of USAF AFWA stations
Data_lists/list_cmans.dat	readin_namelist2.f	Listing of USAF CMANS stations
/gws/nopw/j04/c3s311a_lot2/data/level1/land/'zscratch_ajk3/ L1_stnfiles_notsort/*.csv	get_single_list_data2.f	USAF station file with unsorted data lines

Table 17. Summary of main output files from the sorting of separated USAF files.

Output file and directory	Generation subroutine	Description
/gws/nopw/j04/c3s311a_lot2/data/level1/land/zscratch_ajk3/ L2_stnfiles_sort/*.csv	Export_sorted_list2.f	USAF station file with sorted data lines.

Updates to the USAF source have been included in the second release. First it is necessary to separate the USAF update files into separated USAF station files. The program is run from the Jasmin Linux

command line with ‘./uu1bat’ to generate the executable a.exe and then ‘./a.exe’ to start the program. The program structure with input and output files are summarized in Tables 18 to 20.

Table 18. Summary of Fortran code structure to process the USAF update files to station level.

Linux script	uu1bat to produce executable a.exe
Main program	Uu1.f
Subroutine	uu1.o Subroutine/get_directory_paths.o Subroutine/get_ascii_list.o Subroutine/import_file_listing.o Subroutine/find_vector_path.o Subroutine/process_files.o Subroutine/get_archive_decide_start.o Subroutine/readin_filename_list.o Subroutine/readin_archive_elements.o Subroutine/get_elements_single_file.o Subroutine/create_archive_file.o Subroutine/get_sfcobs_metadata_files.o Subroutine/separate_one_usaf_file1.o Subroutine/export_header_titlelist.o Subroutine/extract_netplat_info.o Subroutine/test_component_char.o Subroutine/test_new_name.o Subroutine/line_exporter.o Subroutine/update_archive_file2.o

Table 19. Summary of main input files to process the USAF update files to station level.

Input file and directory	Handling subroutine	Description
/home/users/akettle/Work/P20200114_usafupdate/	Import_file_listing.f	Get list of USAF update source tar

Datafile_listing/ z_filelist.dat		file names and paths into string vector.
/gws/nopw/j04/c3s311a_lot2/data/level0/land/ sub_daily_data_processing/P20200114_usafupdate/ Data_3archive/*	Get_archive_decide_start.f	Get information from running list of receipt files to decide on start point in case of a computer crash
/gws/nopw/j04/c3s311a_lot2/data/incoming/usaf_update/	Process_files.f	Root directory of USAF update data set from which 1252 source tar files are sequentially drawn for processing

Table 20. Summary of main output files to process the USAF update files to station level.

Output file and directory	Generation subroutine	Description
/gws/nopw/j04/c3s311a_lot2/data/level0/land/ sub_daily_data_processing/P20200114_usafupdate/ Data_1tar/	Process_files.f	Copy one source file into this directory
/gws/nopw/j04/c3s311a_lot2/data/level0/land/ sub_daily_data_processing/P20200114_usafupdate/ Data_2untarpack/	Process_files.f	Untar tarred source file info this directory
/gws/nopw/j04/c3s311a_lot2/data/level0/land/ sub_daily_data_processing/P20200114_usafupdate/ Data_3headerlist/ List_columnsheader.dat	Separate_one_usaf_file1.f	Listing of all column headers in USAF update file showing all available parameters

/gws/nopw/j04/c3s311a_lot2/data/level0/land/ sub_daily_data_processing/P20200114_usafupdate/ Data_3unsorted/*	Separate_one_usaf_file1.f	Collection of USAF station files outputted
gws/nopw/j04/c3s311a_lot2/data/level0/land/ sub_daily_data_processing/P20200114_usafupdate/ Data_3archive/*	Update_archive_file2.f	List of receipt files showing state of process starts and ends.

Next it is necessary to time-sort the USAF update stations. The program is run from the Jasmin Linux command line with ‘./cs1bat’ to generate the executable a.exe and then ‘./a.exe’ to start the program. The program structure with input and output files are summarized in Tables 21 to 23.

Table 21. Summary of Fortran code structure to time sort the USAF update stations.

Linux script	cs1bat to produce executable a.exe
Main program	cs1.f
Subroutine	cs1.o Subroutine/readin_namelist_all.o Subroutine/process_stn_files.o Subroutine/get_lines_single_file.o Subroutine/sort_line_list.o Subroutine/std_datetime_fmt2.o Subroutine/str_to_dt_pvwave3.o Subroutine/sort_doublelist_pvwave2.o Subroutine/find_stats_linelength_commas.o Subroutine/export_sorted_list.o Subroutine/export_level2_summaryline.o

Table 22. Summary of main input files to time sort the USAF update stations.

Input file and directory	Handling subroutine	Description
Source_data/z_filelist.dat	Readin_namelist_all.f	List of unsorted USAF input stations
/work/scratch/akettle/P20200114_usafupdate/Try1c/ Data_3unsorted/*	Get_lines_single_file.f	Unsorted USAF input stations

Table 23. Summary of main output files to time sort the USAF update stations.

Output file and directory	Generation subroutine	Description
/work/scratch/akettle/P20200114_usafupdate/Try1c/ Data_4sorted/*	Export_sorted_list.f	Creates station data line with time-sorted lines
/work/scratch/akettle/P20200114_usafupdate/Try1c/ Data_4diag/line_length_stats.dat	Export_level2_summaryline.f	Creates summary information for each update station: start/end date, number of data lines, min/max line length, min/max comma count

Finally, it is necessary to append the new USAF stations to the original USAF station list. The program is run from the Jasmin Linux command line with ‘./sv1bat’ to generate the executable a.exe and then ‘./a.exe’ to start the program. The program structure with input and output files are summarized in Tables 24 to 26.

Table 24. Summary of Fortran code structure to append USAF station updates.

Linux script	sv1bat to produce executable a.exe
Main program	Sv1.f
Subroutine	sv1.o Subroutine/get_listing_directory.o Subroutine/get_masterlist_stn_cat.o

	Subroutine/output_masterlist_categ.o Subroutine/combine_datasets.o Subroutine/combine_datasets_part1.o Subroutine/combine_datasets_part2.o

Table 25. Summary of main input files to append USAF station updates.

Input file and directory	Handling subroutine	Description
/home/users/akettle/Work/P20200227_usafcompile/ Step1_combine/Output_data/ z_list_main.dat	Get_listing_directory.f	Read in station list to string vector array
/home/users/akettle/Work/P20200227_usafcompile/ Step1_combine/Output_data/ z_list_main.dat	Get_listing_directory.f	Read in station list to string vector array
/gws/nopw/j04/c3s311a_lot2/data/AJK_area/ zscratch_ajk3/L2_stnfiles_sort/*	Combine_datasets_part1.f	Reads content of original USAF station files
/work/scratch/akettle/P20200114_usafupdate/Try1c/ Data_4sorted/*	Combine_datasets_part2.f	Reads content of USAF update station files

Table 26. Summary of main output files to append USAF station updates.

Output file and directory	Generation subroutine	Description
/work/scratch/akettle/P20200227_usafcompile/ Step1_stncombine/*	Combine_datasets_part1.f and Combine_datasets_part2.f	Creates new USAF files with the update information appended to the original station file

Finally, it is necessary to Create IFF files from the time-sorted station files. This program creates IFF files from the USAF station files. The program is run from the Jasmin Linux command line with ‘./im1bat’ to generate the executable a.exe and then ‘./a.exe’ to start the program. The program structure with input and output files are summarized in Tables 27 to 29.

Table 27. Summary of Fortran code structure to create sub-daily IFF files from USAF.

Linux script	Im1bat to produce executable a.exe
Main program	Im1.f
Subroutine	im1.o Subroutine/readin_stnlist.o Subroutine/get_input_elements.o Subroutine/get_elements_from_line.o Subroutine/get_stnconfig_qff.o Subroutine/get_elements_stnconfig_line.o Subroutine/clip_out_new_stnconfig.o Subroutine/readin_metadata_info3.o Subroutine/get_cdmcountry_usaffips.o Subroutine/readin_fips_3lett_conversion2.o Subroutine/readin_countrycode2.o Subroutine/process_files_prelim.o Subroutine/readin_lastfile_index.o Subroutine/get_netplat.o Subroutine/get_stnconfig_singlestn2.o Subroutine/get_metadata_singlestn.o Subroutine/process_single_file5.o Subroutine/get_params_singleline2.o Subroutine/find_minmax_string_vec.o Subroutine/find_nelem_present.o Subroutine/convert_variables2.o Subroutine/convert_single_string_to_float.o Subroutine/time_processor.o Subroutine/std_datetime_fmt.o Subroutine/str_to_jtime_pvwave.o Subroutine/iff_processes.o

	Subroutine/find_distinct_triplet_latlonhgt2.o Subroutine/find_max_distance_span.o Subroutine/export_triplet_info5.o Subroutine/get_key_stats_singlevar.o Subroutine/get_startend_from_6var3.o Subroutine/make_iff_receipt_file2.o Subroutine/make_iff_file_singlevar.o Subroutine/find_number_good.o Subroutine/iff_write_file.o Subroutine/export_triplet_info.o Subroutine/find_max_year_wspd4.o Subroutine/find_maxwind_vector.o Subroutine/find_maxwind_matrix.o Subroutine/month_analysis.o Subroutine/find_prec_foll_timepnt.o Subroutine/find_prec_foll_timepnt_matrix.o Subroutine/choose_first_good_point.o Subroutine/find_histogram_2day.o Subroutine/find_histogram_2day_refine.o Subroutine/export_hiwind_info3.o Subroutine/export_hiwind_info3_refine.o Subroutine/export_last_index.o

Table 28. Summary of main input files to create sub-daily IFF files from USAF.

Input file and directory	Handling subroutine	Description
Data_input/z_filelist.dat	Readin_stnlist.f	Readin list of USAF station files.
Data_input/station_configuration_read.txt	Get_input_elements.f	Get earlier version of station configuration information

Data_input/station_configuration_ffr.csv	Get_stnconfig_qff	Get station configuration information
Data_input/usaf-swo-03_20161109_STNMETADATA.csv	Readin_metadata_info3.f	Get USAF metadata information
Data_input/fips_3lett_mcneill20181016.dat	Get_cdmcountry_usaffips.f	Get information for 2 letter FIPS code and 3 letter iso alpha code
Data_input/modify_subregion.dat	Get_cdmcountry_usaffips.f	Get information for 2 letter CDM code and 3 letter iso alpha code
/work/scratch/akettle/P20200227_usafcompile/ Step1_stncombine/*	Process_single_file5.f	Get information from single USAF file

Table 29. Summary of main output files to create sub-daily IFF files from USAF.

Output file and directory	Generation subroutine	Description
/work/scratch/akettle/P20200227_usafcompile/Step2_iff/	Make_iff_file_singlevar.f	Output IFF file for single variable for single station.
/work/scratch/akettle/P20200227_usafcompile/ Step2_iffinfo2/Accept/*	Export_triplet_info.f	Lat-lon-elevation triplet information for accepted stations
/work/scratch/akettle/P20200227_usafcompile/ Step2_iffinfo2/Reject/*	Export_triplet_info.f	Lat-lon-elevation triplet information for rejected stations
/work/scratch/akettle/P20200227_usafcompile/ Step2_iffinfo2/Latlon0/*	Export_triplet_info.f	Lat-lon-elevation triplet information for stations with zero values of latitude or longitude from the metadata file.

The following data sources have also been converted to IFF using Python 3.7 conversion codes. All the code can be executed by using Anaconda–Spyder scientific environment or any Python integrated

development environment. The code needs to be run interactively as some parts of the script will need to be changed, for example source IDs and time zone differences will vary. The code has full descriptions and instructions within the code. The data sources are treated as static sources presently which means that only one-time operation of these routines is required.

158: Met Eireann synoptic

The Met Eireann conversion code converts each variable one at a time and once converted to IFF a second Python 3.7 code converts the Local time to UTC. The code has full descriptions and instructions within the code. All the Python 3.7 Met Eireann conversion code can be found at:
https://github.com/glamod/glamod-nuim/blob/master/met_eireann_IFF_convert_158.zip

244 - TD_14_NCAR_Sub_Daily_data_US

The TD-14 conversion code converts each variable one at a time and once converted to IFF a second Python 3.7 code converts the Local time to UTC. The code has full descriptions and instructions within the code. All the Python 3.7 TD-14 conversion code can be found at: https://github.com/glamod/glamod-nuim/blob/master/TD-14_convert_to_IFF_2019.zip

245 - TD_13_NCAR_Sub_Daily_data_Global

The TD-13 conversion code converts each variable one at a time. The code has full descriptions and instructions within the code. All the Python 3.7 TD-13 conversion code can be found at: https://github.com/glamod/glamod-nuim/blob/master/TD-13_convert_to_IFF_2019.zip

246 to 315 - ISPD underlying data sources

The ISPD conversion code converts each variable one at a time. The code has full descriptions and instructions within the code. All the Python 3.7 ISPD conversion code can be found at: https://github.com/glamod/glamod-nuim/blob/master/ISPD_conversion_%20to_%20IFF_code_2019.zip

316 - DWRUK_sub_daily_europe

The DWRUK conversion code converts each variable one at a time. The code has full descriptions and instructions within the code. All the Python 3.7 DWRUK conversion code can be found at: https://github.com/glamod/glamod-nuim/blob/master/UKMO_DWRUK_316_convert_SEF_to_IFF.zip

320-NUS-Nanking Meteorological Bulletins

The NUS conversion code converts each variable one at a time. The code has full descriptions and instructions within the code. All the Python 3.7 NUS conversion code can be found at: https://github.com/glamod/glamod-nuim/blob/master/NUS_China_IFF_code_320.zip

321- DWD_overseas data_China_Pacific_Togo

The DWD conversion code converts each variable one at a time. The code has full descriptions and instructions within the code. All the Python 3.7 DWD conversion code can be found at: https://github.com/glamod/glamod-nuim/blob/master/DWD_oversea_IFF_code_321.zip

Table 30. Intermediate File Format (IFF) structure and field descriptors.

Column_Header	***Description***
Source_ID	Source identifier which will be issued by C3S Lot2
Station_ID	Station Identifier - record number based on station configurations. e.g - WMO10002-1
Station_name	Station name
Alias_station_name	Alias station name (if more than one separate with a comma)
Year	Year (GMT) of the observation record
Month	Month (GMT) of the observation record
Day	Day (GMT) of the observation record
Hour	Hour (GMT) of the observation record
Minute	Minute (GMT) of the observation record
Latitude	Latitude coordinate of a geophysical observation (-90.00 to 90.00)
Longitude	The longitude coordinate of a geophysical observation (-180.00 – 180.00)
Elevation	The elevation of a geophysical observation in meters relative to Mean Sea Level
Observed_value	The observed value (see variable units below)
Source_QC_flag	Quality Flags from source for observed value (0 passed 1 failed 3 unknown 9 missing)

Original_observed_value	Original observed value (if available)
Original_observed_value_units	Original observed value units
Report_type_code	This code corresponds to the Geophysical Report e.g FM-12 FM-13 etc
Gravity_corrected_by_source	Gravity Correction made by source to pressure observations (0 passed 1 failed 3 unknown 9 missing)
Homogenization_corrected_by_source	Homogenization corrected by source (0 passed 1 failed 3 unknown 9 missing)

The IFF source files are then integrated into the merge process. Once a mingle list has been produced and stations have been merged the output file is a Merge File Format (MFF).

The MFF consists of multiple variables per source, Table 31 shows all column headers and corresponding descriptions.

The MFF file should be saved as a pipe [|] separated file ext = [.psv] multiple variables per file.

The MFF is named by using the Merged station ID.

The MFF files are saved in Level1/Land/ directory on the JASMIN workspace in the relevant Level1b timescale sub-directory e.g.

/gws/nopw/j04/c3s311a_lot2/data/Level1/Land/level1b_sub_daily_data

The merge code is currently being developed, refined and tested by NOAA NCEI and once available in stable form will be hosted on GitHub.

Table 31. Merge File Format (MFF) structure and field descriptors.

Column_Header	**Description**
Station_ID	Merged Station Identifier
Station_name	station name
Year	Year (GMT) of the observation record
Month	Month (GMT) of the observation record
Day	Day (GMT) of the observation record
Hour	Hour (GMT) of the observation record
Minute	Minute (GMT) of the observation record
Latitude	Latitude coordinate of a geophysical observation (-90.00 to 90.00)
Longitude	The longitude coordinate of a geophysical observation (-180.00 – 180.00)
Elevation	The elevation of a geophysical observation in meters relative to Mean Sea Level
temperature	observed variable
Source_ID	Source identifier for each observation
dew_point_temperature	observed variable
Source_ID	Source identifier for each observation
station_level_pressure	observed variable
Source_ID	Source identifier for each observation
sea_level_pressure	observed variable
Source_ID	Source identifier for each observation
wind_direction	observed variable
Source_ID	Source identifier for each observation
wind_speed	observed variable
Source_ID	Source identifier for each observation

Marine merge

The marine merge will be discussed here in future releases if necessary. For present the marine processing uses existing merge decisions from ICOADS.

4.2.2 Quality Assurance / Quality Control

Land Quality Assurance

For the current release, we are using the quality control (QC) and quality assurance data associated with the GHCN-D and GSOM sources without modification. Further details for each source are available via the journal papers and grey literature describing the processing as discussed below. However, for the second full data release we are applying our own QC checks to the sub-daily data, and details are presented in this section with the scientific basis documented in the Land User Guide. In subsequent releases we shall be developing and deploying new or additional QC/QA which may serve to augment and / or replace the initial quality control summarised herein. Quality control decisions contain irreducible ambiguity and it is important that original data be retained to allow subsequent revisiting of data flagging decisions when new insights or improved techniques accrue.

Daily data QC

The GHCN-D quality control procedure is documented extensively in Menne et al., (2012) with any updates noted on the GHCN-D website (https://www.ncdc.noaa.gov/data-access/land-based-station-data/land-based-datasets/global-historical-climatology-network-ghcn). The quality control (QC) consists of a mix of record exceedance checks, climatological checks, distributional checks and neighbour-based checks (see Table 32). The GHCN-D quality control flags provide information on the tests failed by an observation. This will be codified in full in subsequent releases via a pivot table to the Common Data Model. However, for the present release solely information on whether the tests are all passed or at least one test has failed is provided. We are simply flagging an observation by converting the GHCN-D QC source flags to either passed QC flag (0) or failed QC flag (1). This information should be sufficient to gain user feedback on adequacy of the current release as a whole. Most users will probably never make use of more than this simple flag approach encoded in the observations table.

Monthly data QC

For GSOM the quality control flags of GHCN-D are used to exclude suspect daily values from being used to calculate a monthly average. Some QC flags are carried into the GSOM and we are also simply

flagging an observation by converting the GHCN-D QC source flags in the GSOM to either passed QC flag (0), failed QC flag (1), missing (3) or observed value updated or changed (4). The Land User Guide provides more details of the criteria for calculating the GSOM from the GHCN-D for each variable. These criteria are consistent with WMO guidance. At this juncture no further quality control is applied to the monthly data.

Table 32. Shows the list of GHCN-Daily Dataset Quality Checks and flags relevant to the GSOM data

Source QC Flag	*Description of Quality Control Check*
D	Duplicate check
G	Gap check
I	Internal consistency check
K	Streak/frequent-value check
L	Check on length of multiday period
M	Mega consistency check
N	Naught check
O	Climatological outlier check
R	Lagged range check
S	Spatial consistency check
T	Temporal consistency check
W	Temperature too warm for snow
X	Failed Bounds check
Z	Flagged as a result of an official datzilla investigation

Sub-daily data QC

The documentation on the quality checks applied to the USAF sub-daily data at source is difficult to fully understand but Table 33 shows information that was ascertained on basic quality control checks applied to the data at both inter and intra station level. Due to the ambiguity in the USAF quality control process documentation we have adopted a modified sub-set of the HadISD quality control checks for the sub-daily data in the second full data release (Dunn, 2019). For the present release solely information on whether the tests are all passed or at least one test has failed is provided. Again, we are simply flagging an observation with either passed QC flag (0) or failed QC flag (1). The following section describes these checks in detail. We also apply some data logic checks to the sub-daily data, details are presented further in this section.

Table 33. Shows the USAF sub daily data source basic checks and flags that has been applied to the data on an intra and inter station level.

USAF Source QC Flag	*Description of USAF source Quality Control Check*
0	Original Value Missing or Corrupted.
1	Gross Error Checks (Range and/or domain check),
2	Geophysical Checks (Checking the validity against other parameters).
3	Gross Error Checks and Geophysical Checks.
4	Gross Error Checks and Consistency Checks.
5	Geophysical Checks and Consistency Checks.
6	Gross Error Checks and Geophysical Checks and Consistency Checks

The current set of sub-daily QC tests work on individual stations (intra-station checks) or between stations (neighbour/buddy, or inter-station checks). There are no checks working across time periods (ensuring that monthly, daily and sub-daily data are consistent with each other). These QC tests have been based on the suite of checks from the HadISD dataset (Dunn, 2019, Dunn et al, 2016, Dunn et al, 2012), which themselves were inspired by some of the checks applied the GHCND dataset (Durre et al, 2010). For further details and examples of these tests, please refer to those publications and the Land User Guide. In the current implementation, each test works independently, with no account taken of flags set by tests earlier in the sequence. Hence a single observation can be flagged by many tests, but also thresholds derived from the data themselves will include potentially bad data.

Where thresholds are set from the characteristics of the data themselves, these are stored in separate files. These allow the QC tests to be run with stable thresholds where data have been appended since the last run. If the thresholds were recalculated every time, then it would be possible that additional data could change the thresholds, either unflagging previously flagged observations, or flagging ones that previously would not have been flagged. All the code to run the land QC checks can be found at: https://github.com/glamod/glamod_landQC

To ensure that all relevant libraries are available to the Python scripts which perform the QC, a virtual environment is necessary. The file `venv_requirements.txt` allows the creation of a virtual environment to match what has been used during development and the existing Jaspy environment ensures that the base Python is at version 3 or greater.

```
$> module load jaspy
$> virtualenv venv
$> source venv/bin/activate
$> pip install -r venv_requirements.txt
$> module unload jaspy
```

Currently the land QC scripts are initiated using a simple script which submits the processing to the LOTUS cluster on JASMIN. There is a configuration file which will set the file locations for the input, processing and output files. Currently there are two options, to use a `<c3sbase>` on the JASMIN group workspace or on a scratch space which has better I/O performance. If using the scratch space, then it may be necessary to copy over the mingled files for use in the QC process.

Presuming the ghcnh_stations.txt file created during the mingle is available in `/<c3sbase>/data/Level1/Land/level1b_sub_daily_data/<version>/` then a bash script can be called using the option "I" to select the internal checks.

```
$> bash run_qc_checks.bash I
```

To ensure that the buddy checks are run on files with a consistent level of quality control, the internal checks write QFF files in an intermediate directory (`/<c3sbase>/data/Level1/Land/level1b1_sub_daily_data/<version>/`). To create the file which contains the neighbours for each station, run

```
$> python find_neighbours.py
```

This script can be run in parallel with the internal checks as it works using the station list from the mingle process. Once all these files are present, then the bash script can be called with the option "N" which submits the neighbour-check processes to LOTUS.

```
$> bash run_qc_checks.bash N
```

When this script has completed, summary plots can be created using a further script which runs on LOTUS

$> `bsub < plots_lotus.bash`

The output files for the plots are in `/<c3sbase>/data/Level1/Land/level1c_sub_daily_data_plots/<version>/`. Each station has a configuration file associated with it where thresholds can be stored. This allows a reprocessing as part of an append action, where there should be no changes to the QC flags in the deep past. To action this please see the documentation of the `inter_checks.py` and `intra_checks.py` scripts.

These configuration files are stored in:

`/<c3sbase>/data/Level1/Land/level1c_sub_daily_data_config/<version>/.`

Finally, if the flagging rate is too high for one of the variables, the entire station is withheld at present. These are saved into a subdirectory of the output location “bad_stations”.

For sub-daily data, the quality control checks are conducted on the MFF files and the final output file is a Quality-controlled File Format (QFF). The QFF includes all the quality control flags applied for each observed variable (Table 34).

The QFF are saved as a pipe [|] separated file ext = [.psv] Multiple variables per file.

Table 34. Shows all column headers and corresponding descriptions for the QFF files.

Column_Header	Description
Station_ID	Merged Station Identifier
Station_name	station name
Year	Year (GMT) of the observation record
Month	Month (GMT) of the observation record
Day	Day (GMT) of the observation record
Hour	Hour (GMT) of the observation record
Minute	Minute (GMT) of the observation record
Latitude	Latitude coordinate of a geophysical observation (-90.00 to 90.00)
Longitude	The longitude coordinate of a geophysical observation (-180.00 – 180.00)

Elevation	The elevation of a geophysical observation in meters relative to Mean Sea Level
Temperature	Observed variable
temperature_Source_ID	Source identifier for each observation
temperature_QC_flag	Quality control flag for temperature. denoted by an alpha_numeric relating to whether observed value passed, failed a specific test.
dew_point_temperature	Observed variable
dew_point_temperature_Source_ID	Source identifier for each observation
dew_point_temperature_QC_flag	Quality control flag for dew_point_temperature denoted by an alpha_numeric relating to whether observed value passed, failed a specific test.
station_level_pressure	Observed variable
station_level_pressure_Source_ID	Source identifier for each observation
station_level_pressure_QC_flag	Quality control flag for station_level_pressure denoted by an alpha_numeric relating to whether observed value passed, failed a specific test.
sea_level_pressure	Observed variable
sea_level_pressure_Source_ID	Source identifier for each observation
sea_level_pressure_QC_flag	Quality control flag for sea_level_pressure denoted by an alpha_numeric relating to whether observed value passed, failed a specific test.
wind_direction	Observed variable
wind_direction_Source_ID	Source identifier for each observation
wind_direction_QC_flag	Quality control flag for wind_direction denoted by an alpha_numeric relating to whether observed value passed, failed a specific test.
wind_speed	Observed variable
wind_speed_Source_ID	Source identifier for each observation
wind_speed_QC_flag	Quality control flag for wind_speed denoted by an alpha_numeric relating to whether observed value passed, failed a specific test.

Marine Quality Assurance

The marine Quality Control / Quality Assurance code is pre-existing IP and can be found at

- https://github.com/ET-NCMP/MarineQC

When code is modified this section will be updated and the QC software forked to the service repository.

4.3 Common data Model (CDM) and CDMlite

4.3.1 Understanding the full CDM and the CDMlite

The full Common Data Model (CDM) defines a complete set of tables to capture detailed metadata, data, quality and provenance information for all land and marine observations. The aim of the project is to provide this data to the C3S Climate Data Store.

For the current release, a cut-down version of the CDM has been prioritised in order to ensure that an initial dataset can be delivered and tested. This cut-down version is known as the CDMlite and it reduces the data structure to a single "observation" table. In production, this table is partitioned across three axes so over 1,000 actual tables exist in the database. The structure of the CDMlite is as follows:

```
        Column         |           Type           | Collation | Nullable |
-----------------------+--------------------------+-----------+----------+
 observation_id        | character varying        |           | not null |
 data_policy_licence   | integer                  |           |          |
 date_time             | timestamp with time zone |           |          |
 date_time_meaning     | integer                  |           |          |
 observation_duration  | integer                  |           |          |
 longitude             | numeric                  |           |          |
 latitude              | numeric                  |           |          |
 report_type           | integer                  |           |          |
```

height_above_surface	numeric			
observed_variable	integer			
units	integer			
observation_value	numeric			
value_significance	integer			
platform_type	integer			
station_type	integer			
primary_station_id	character varying			
station_name	character varying			
quality_flag	integer			
location	geography(Point,4326)			

Having archived the data the next step is to convert the data into the CDM/CDMlite structures agreed for C3S in situ holdings across the C3S 311a lots {https://github.com/glamod/common_data_model/blob/master/C3S_D311a_Lot2.2.1.1_201708_Initial_specification_for_CDM_v1.09.pdf}. For both land and marine sources this conversion is carried out at the source deck level, such that the entire sub-directory structure is copied from {raw directory} to {cdm_compatible_directory}. The only distinction is that the data are now held in a format compatible with the CDM.

All the data processing to convert the land second full data release source data files to the CDM format has been performed on the JASMIN Analysis Platform using Fortran code.

The Fortran code can be found at: https://github.com/glamod/glamod-nuim

4.3.2 Land sub-daily second full data release

The sub-daily data for the current release consists of a sub-set of platforms extracted from the USAF data merged with other sub-daily data.

These other sub-daily data that are included in the current data merge come from the following sources:

- The International Surface Pressure Databank (ISPD) which contains 70 underlying sources and consist almost exclusively of sea level pressure and station level pressure data.

- The “TD-13” dataset which came from the NCAR data archive and consists of 10,851 global stations with temperature, precipitation, humidity, wind and pressure observations
- The “TD-14” dataset which also came from the NCAR data archive and consists of 300 stations located across the USA with temperature, precipitation, humidity, wind and pressure observations.
- 500 stations from the NOAA CDMP located across the US with observations of wind, snow, temperature, water vapour and pressure from 1892-1997.
- 72 stations from the Met Office Daily Weather Reports (1900-1910) as part of the citizen science project Weather Rescue (https://weatherrescue.org) and contains observations for temperature, pressure and humidity.

The USAF sub-daily stations have retained the transmitted location metadata associated with each individual observation report. However, the USAF recommend use of their master chronology. For many stations, there exist multiple variations in location and elevation information in the observation report entries throughout the operational period. It is unclear whether these variations relate to real geophysical location moves or rather changes in surveyed location, fat finger typographical errors, changes in reporting of location precision etc. There are also stations that have very few reports or contain obviously inconsistent data. Therefore, it was deemed essential to filter out stations with bad data and suspicious within-report location information to minimise the chances of having to remove stations from subsequent releases.

Retaining solely stations that had no within-report geolocation ambiguity resulted in an overly restrictive selection being retained. Thus, instead that subset with reasonable consistency in geolocation within the reports were retained at the present time. The sub-set of USAF stations that were retained for the present release passed the following criteria:

1. The maximum range in all the latitude and longitude listings of the stations had to be less than or equal to 0.4 degrees (44 km at the equator). This means the station representativity aspects of record homogeneity, even if it has truly moved one or more times, should be relatively manageable for most applications.
2. The station had to have greater than 60 months of good data where a good month was defined as one with at least 20 data points. This means there should be sufficient data to support at least some climate service application areas.
3. The master USAF metadata information in the chronology file for the station had to have a valid latitude, longitude, and elevation report.
4. The master USAF metadata information in the chronology file latitude/longitude and elevations had to agree with one of the observation file triplets within 0.0005 degrees for latitude and longitude. (This takes account of the instances where there was an imperfect match between the metadata file and observation file listings).

All the completed sub daily data CDM tables for the second full data release are saved to a JASMIN directory and can be found at:

/gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/observations_tables/sub_daily

/gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/header_tables/sub_daily

This program to convert from the input QFF files to the CDM format can be run as follows. The program is run from the Jasmin Linux command line with ‘./qf1bat20200605’ to generate the executable a.exe and then ‘./a.exe’ to start the program. The program structure with input and output files are summarized in the tables 35 through 37.

Table 35. Summary of Fortran code structure to convert qff files to CDM and CDM-lite.

Linux script	qf1bat20200605 to produce executable a.exe
Main program	qf20200605.f
Subroutine structure	qf20200605.o Subroutine/get_stnconfig_qff.o Subroutine/get_elements_stnconfig_line.o Subroutine/get_list_qff_files.o Subroutine/process_qff_files20200607.o Subroutine/get_codes_processing_qff.o Subroutine/readin_lastfile_number_qff.o Subroutine/test_qff_file_in_stnconfig.o Subroutine/isolate_single_name.o Subroutine/test_file_segfault.o Subroutine/get_singles_stnconfig_qff2.o Subroutine/process_single_qff_file20200607.o Subroutine/extract_fields_from_line.o Subroutine/find_original_precision_qff.o Subroutine/find_precision_vector_qff.o Subroutine/find_number_elements.o

	Subroutine/convert_var_string_to_float.o Subroutine/string_convert_float_qff.o Subroutine/convert_units_qff.o Subroutine/assess_convprec_pressure.o Subroutine/record_number_from_source_number_qff20200617b.o Subroutine/find_record_number_qff20200617b.o Subroutine/export_record_number_problem.o Subroutine/find_cdm_qc_code.o Subroutine/header_obs_lite_qff20200609.o Subroutine/reconstruct_date_time.o Subroutine/assemble_vector_qff.o Subroutine/count_elements_vector_qff.o Subroutine/sample_screenprint_obs.o Subroutine/sample_screenprint_lite.o Subroutine/find_distinct_recnum_qff.o Subroutine/sample_screenprint_header.o Subroutine/make_new_stnconfig_qff.o Subroutine/get_strvector_distinct.o Subroutine/export_stnconfig_lines.o Subroutine/export_last_index_qff20200612.o

Table 36. Summary of main input files to convert qff files to CDM and CDM-lite.

Input file and directory	Handling subroutine	Description
Data_stnconfig/ prelim_subdy_station_config_12_06_2020.psv.csv	get_stnconfig_qff.f	Station configuration file
Data_middle/z_qff_filelist.dat	get_list_qff_files.f	List of qff files generated from their source directory

/gws/nopw/j04/c3s311a_lot2/data/level1/land/ level1c_sub_daily_data/v20200528/???????????.qff	process_single_qff_file20200607.f	Collection of qff station files

Table 37. Summary of main output files and ancillary output files to convert qff files to CDM and CDM-lite.

Output file and directory	Generation subroutine	Description
/work/scratch-pw/akettle/P20200604_qff_to_cdm/Output/ Header/ header_table_SecondR_???????????.psv	header_obs_lite_qff20200609.f	CDM header files for station collection
/work/scratch-pw/akettle/P20200604_qff_to_cdm/Output/ Observation/ observation_table_SecondR_???????????.psv	header_obs_lite_qff20200609.f	CDM observation files for station collection
/work/scratch-pw/akettle/P20200604_qff_to_cdm/Output/ Lite/ CDM_lite_SecondRelease_???????????.psv	header_obs_lite_qff20200609.f	CDM lite files for station collection
/work/scratch-pw/akettle/P20200604_qff_to_cdm/Output/ Receipt/ ???????????.dat	export_stnconfig_lines.f	Reconstructed station configuration lines with column information for variables and start/end year included
/work/scratch-pw/akettle/ P20200604_qff_to_cdm/Output/ Lastfile/lastfile.dat	export_last_index_qff20200612.f	Record of last completed station index needed to restart the program in the event of a crash or a sequential process done in station chunks

/work/scratch-pw/akettle/ P20200604_qff_to_cdm/Output/ Diagnostics/???????????	record_number_from_source_number_qff20200617b.f	Record of problem station where qff source_id for variable could not be matched to stnconfig list; first incidence of up to the 6 variables.
/work/scratch-pw/akettle/ P20200604_qff_to_cdm/Output/ Diagnostics_segfault/	test_file_segfault	Record of Fortran-unreadable files that generated segmentation faults. They are identified with a preliminary Linux script to read the header, if possible.

The Fortran code for the current daily data conversion to the CDM can be found at: https://github.com/glamod/glamod-nuim

File package name: firstrelease_subdaily_qff.tar.gz

4.3.3 Land daily second full data release

The daily data for the second full release consists of a subset of stations extracted from NOAA's National Centre for Environmental Information (NCEI) Global Historical Climatological Network Daily (GHCN-D). The GHCN-D database consists currently of in excess of 120 thousand stations, although many of these are precipitation only stations. Given the stated aim for a multivariate set of holdings, the current release does not include these stations. The subset of 75,247 global daily stations were selected from the GHCN-D dataset based on the availability of at least two of our target ECVs.

All the completed daily data CDM tables for the second full data release are saved to a JASMIN directory and can be found at:

/gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/observations_tables/daily

/gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/header_tables/daily

The data have been downloaded in a bulk download from NOAA NCEI. The first step is to separate this big NCEI time order file into a set of station files. The program is run from the Jasmin command line using ‘./ss2bat’ to create the a.exe executable and then ‘./a.exe’ to start the program. The program structure with input and output files is summarized in Tables 38 to 40.

Table 38. Summary of Fortran code structure to reorder GHCND into station files.

Linux script	ss2bat to produce executable a.exe
Main program	ss2.f
Subroutine	ss2.o Subroutine/read_split_data.o Subroutine/get_stn_name.o Subroutine/output_group_lines.o Subroutine/file_progress_update.o

Table 39. Summary of main input files to reorder GHCND into station files.

Input file and directory	Handling subroutine	Description
/work/scratch-nompiio/snoone/superghcnd_daily_updates/ full_superghcnd_update/Stns_separated5/ superghcnd_full_20200331.csv	Split_file_data.f	The large NCEI source data file for GHCND

Table 40. Summary of main output files to reorder GHCND into station files.

Output file and directory	Generation subroutine	Description
/work/scratch-nompiio/snoone/superghcnd_daily_updates/ full_superghcnd_update/ Stns_separated5/ ???????????.dat	Output_group_lines.f	Separated GHCND station files that are named with the 11-character NCEI format

The program to convert the resulting GHCND station files to CDM format can be run as follows. The program is run from the Jasmin command line using ‘./gd1bat’ to create the b.exe executable and then ‘./b.exe’ to start the program. The program structure with input and output files is summarized in Tables 41-43.

Table 41. Summary of Fortran code structure to convert GHCND station files to CDM and CDM-lite.

Linux script	gd1bat to produce executable b.exe
Main program	gd20200410.f
Subroutine	gd20200410.o Subroutine/readin_lastfile_number.o Subroutine/erase_files_seq2.o Subroutine/readin_source_id2.o Subroutine/get_stnconfig_ghcnd2.o Subroutine/get_elements_stnconfig_line2.o Subroutine/readin_subset.o Subroutine/main_cycler.o Subroutine/get_singles_stnconfig_v5.o Subroutine/export_errorfile.o Subroutine/input_output_single_station.o Subroutine/get_lines_1file2.o Subroutine/get_fields_from_ghcnd_line2.o Subroutine/find_len_fields2.o Subroutine/verify_stn_id2.o Subroutine/find_hist_param_elements2.o Subroutine/find_distinct_timesteps2.o Subroutine/get_sourceid_numcode2.o Subroutine/get_source_id_single2.o Subroutine/get_record_number2.o Subroutine/search_stringfragment2.o Subroutine/ambigcase_get_low_record2.o

	Subroutine/total_counts_recordnumber2.o Subroutine/convert_string_to_float_ghcnd3.o Subroutine/convert_variable_column2.o Subroutine/convert_float_to_string2.o Subroutine/convert_integer_to_string2.o Subroutine/test_precision_variables2.o Subroutine/precision_single_vector1a.o Subroutine/precision_single_vector2a.o Subroutine/find_hist_dailyprecision2.o Subroutine/precision_interpreter_tenths2.o Subroutine/precision_interpreter_whole2.o Subroutine/header_obs_lite_table2.o Subroutine/sort_vec_obs_ancillary_ghcnd2.o Subroutine/find_vector_recordnumber2.o Subroutine/assemble_vec_original_precision2.o Subroutine/get_numerical_precision2.o Subroutine/get_quality_flag_vector2.o Subroutine/qc_single_variable2.o Subroutine/get_observationvalue_vector2.o Subroutine/convert_float_to_string_single2.o Subroutine/get_hght_obs_above_sfc.o Subroutine/get_strvector_distinct.o Subroutine/export_stnconfig_lines.o Subroutine/export_last_index.o

Table 42. Summary of main input files to convert GHCND station files to CDM and CDM-lite.

Input file and directory	Handling subroutine	Description
/gws/nopw/j04/c3s311a_lot2/data/ level0/land/daily_data/ superghcnd_stations/???????????.csv	get_lines_1file2.f	GHCNd station data in the NCEI ASCII format. The station names have the 11 character NCEI identifier format.

/home/users/akettle/Work/P20200304_ghcnd20/Step2_cdmmake_20200410/ Data_ancillary/ GHCNdSource_C3Ssource_IDS_20191101.psv	readin_source_id2.f	File with conversion key between NCEI letter codes and SN number codes.
/home/users/akettle/Work/P20200304_ghcnd20/Step2_cdmmake_20200410/ Data_stnconfig/ f20200409_preliminary_station_configuration_GHCND.psv.csv	get_stnconfig_ghcnd.f	Full station configuration file from SN
/home/users/akettle/Work/P20200304_ghcnd20/Step2_cdmmake_20200410/ Data_ancillary/ f20200406_station_list_GHCND.csv	readin_subset.f	File from SN with subset of the station configuration list.

Table 43. Summary of main output files to convert GHCND station files to CDM and CDM-lite.

Output file and directory	Generation subroutine	Description
/work/scratch/akettle/P20200304_ghcnd20/ Step2_cdmmake_20200410/Data_outputs/CDM_header/ header_table_FirstR_??????????.psv	header_obs_lite_table2.f	CDM header files with a wildcard denoting the 11 character NCEI station name
/work/scratch/akettle/P20200304_ghcnd20/ Step2_cdmmake_20200410/Data_outputs/CDM_observation/ observation_table_FirstR_??????????.psv	header_obs_lite_table2.f	CDM observation files with a wildcard denoting the 11 character NCEI station name
/work/scratch/akettle/P20200304_ghcnd20/ Step2_cdmmake_20200410/Data_outputs/CDM_observation/ CDM_lite_FirstR_??????????.psv	header_obs_lite_table2.f	CDM lite with a wildcard denoting the 11 character

		NCEI station name
/work/scratch/akettle/P20200304_ghcnd20/ Step2_cdmmake_20200410/Data_outputs/Receipt/ ?????????.dat	Export_stnconfig_lines.f	Lines from the original station configuration file amended with information from the IFF files: variables present and first/last years of data
/work/scratch/akettle/P20200304_ghcnd20/ Step2_cdmmake_20200410/Data_outputs/Lastnumber/ Lastfile.dat	Export_last_index.f	Record of last completed station to restart program in the event of a computer crash
/work/scratch/akettle/P20200304_ghcnd20/ Step2_cdmmake_20200410/Data_outputs/Errorfile/ Errorfile_recordnumber.dat	Export_errorfile.f	List of stations with record number problems

The Fortran code for the current daily data conversion to the CDM can be found at: https://github.com/glamod/glamod-nuim

File package name: firstrelease_daily_ghcnd.tar.gz

4.3.4 Land monthly second full data release

The same selected subset of GHCN-D daily stations were extracted from NCEI's Global Summary Of The Month (GSOM) dataset. The Global Summary of the Month (GSOM) and Global Summary of the Year (GSOY) datasets consist of 55 climatological variables (many being derived quantities such as

heating degree days and growing degree days) computed from summary of the day observations of the Global Historical Climatology Network Daily dataset. There are 72,651 monthly stations extracted from the GSOM dataset that match the GHCN-D daily stations for the present release.

All the completed monthly data CDM tables for the second full data release are saved to a JASMIN directory and can be found at:

/gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/observations_tables/monthly

/gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/header_tables/monthly

The program to convert the GSOM files to CDM format can be run as follows. The program is run from the Jasmin command line using ‘./gm3bat’ to create the a.exe executable and then ‘./a.exe’ to start the program. The program structure with input and output files is summarized in Tables 44 through 46.

Table 44. Summary of Fortran code structure to convert GSOM to CDM and CDM-lite.

Linux script	gmxbat to produced executable a.exe
Main program	gm20200518.f
Subroutine	gm20200518.o Subroutine/readin_source_id2.o Subroutine/get_stnconfig_ghcnd2.o Subroutine/get_elements_stnconfig_line2.o Subroutine/get_scinput_info.o Subroutine/readin_subset.o Subroutine/process_gsom_files20200520.o Subroutine/get_codes_gsom_processing20200527.o Subroutine/station_loop_process20200520.o Subroutine/info_from_stnconfig.o Subroutine/info_from_stnconfig_vector.o Subroutine/readin_lastfile_number20200603.o Subroutine/get_singles_stnconfig_v5.o

	Subroutine/export_test1.o
	Subroutine/get_gsom_datalines20200523.o
	Subroutine/field_extractor_gen.o
	Subroutine/find_variable_indices.o
	Subroutine/find_variable_index_single.o
	Subroutine/field_extractor_quot.o
	Subroutine/get_data_into_vectors2.o
	Subroutine/get_date_into_vector.o
	Subroutine/get_singlestring_into_vector20200523.o
	Subroutine/latlon_4decimal20200523.o
	Subroutine/find_original_precision2.o
	Subroutine/find_precision_vector3.o
	Subroutine/find_number_elements.o
	Subroutine/get_qc_from_attrib.o
	Subroutine/qc_checker.o
	Subroutine/get_source_char_from_attrib3.o
	Subroutine/get_single_source_char.o
	Subroutine/get_single_source_char2.o
	Subroutine/get_matrix_horiz_search.o
	Subroutine/find_altern_letter.o
	Subroutine/export_test2.o
	Subroutine/source_number_from_character.o
	Subroutine/source_number_from_character_single.o
	Subroutine/record_number_from_source_number3.o
	Subroutine/find_record_number_single3.o
	Subroutine/get_mat_altern_sourcenum.o
	Subroutine/fix_record_number.o
	Subroutine/fix_datzilla.o
	Subroutine/export_test4.o
	Subroutine/modify_qc_code.o
	Subroutine/convert_variables2.o
	Subroutine/declare_converted_var_precision.o

	Subroutine/count_occurrence_variables.o Subroutine/make_header_observation_lite20200522.o Subroutine/assemble_vector_gsom.o Subroutine/count_elements_vector.o Subroutine/find_distinct_recnum_gsom.o Subroutine/make_new_stnconfig2.o Subroutine/get_strvector_distinct.o Subroutine/export_stnconfig_lines20200522.o Subroutine/export_last_index20200603.o

Table 45. Summary of main input files to convert GSOM to CDM and CDM-lite.

Input file and directory	Handling subroutine	Description
/home/users/akettle/Work/P20200518_gsombig/Data_ancillary/ GHCNdSource_C3Ssource_IDS_20191101.psv	readin_source_id2.f	Conversion key between NCEI source letter code and CDM source number code.
/home/users/akettle/Work/P20200518_gsombig/Data_stnconfig/ f20200409_preliminary_station_configuration_GHCND.psv.csv	get_stnconfig_ghcnd2.f	Station configuration file
/home/users/akettle/Work/P20200518_gsombig/Data_ancillary/ f20200406_station_list_GHCND.csv	readin_subset.f	List of stations for release
/work/scratch/akettle/P20200518_gsombig/Data_source/ ???????????.csv	get_gsom_datalines4.f	NCEI monthly station files

Table 46. Summary of main output files to convert GSOM to CDM and CDM-lite.

Output file and directory	Generation subroutine	Description
/work/scratch/akettle/P20200518_gsombig/Data_output/	make_header_observation_lite20200522.f	CDM header station files

Header/ header_table_FirstR_???????????.psv		
/work/scratch/akettle/P20200518_gsombig/Data_output/ Observation/ observation_table_FirstR_???????????.psv	make_header_observation_lite20200522.f	CDM observation station files
/work/scratch/akettle/P20200518_gsombig/Data_output/ Lite/ CDM_lite_FirstR_???????????.psv	make_header_observation_lite20200522.f	CDOM lite files
/work/scratch/akettle/P20200518_gsombig/Data_output/ Receipt/ ???????????.dat	export_stnconfig_lines20200522.f	Station configuration lines for each station with information on the variable present and start/end years

The Fortran code for the monthly data conversion to the CDM can be found at:

https://github.com/glamod/glamod-nuim

File package name: firstrelease_monthly_gsom.tar.gz

Full details step by step on how to produce the CDM station_configuration and station_configuration_optional tables from the current station merge/mingle lists are available at:

https://github.com/glamod/glamod-nuim/blob/master/steps%20in%20prep%20of%20station%20config%20and%20optional%20data%20tables.docx

File name: steps in prep of station config and optional data tables.docx

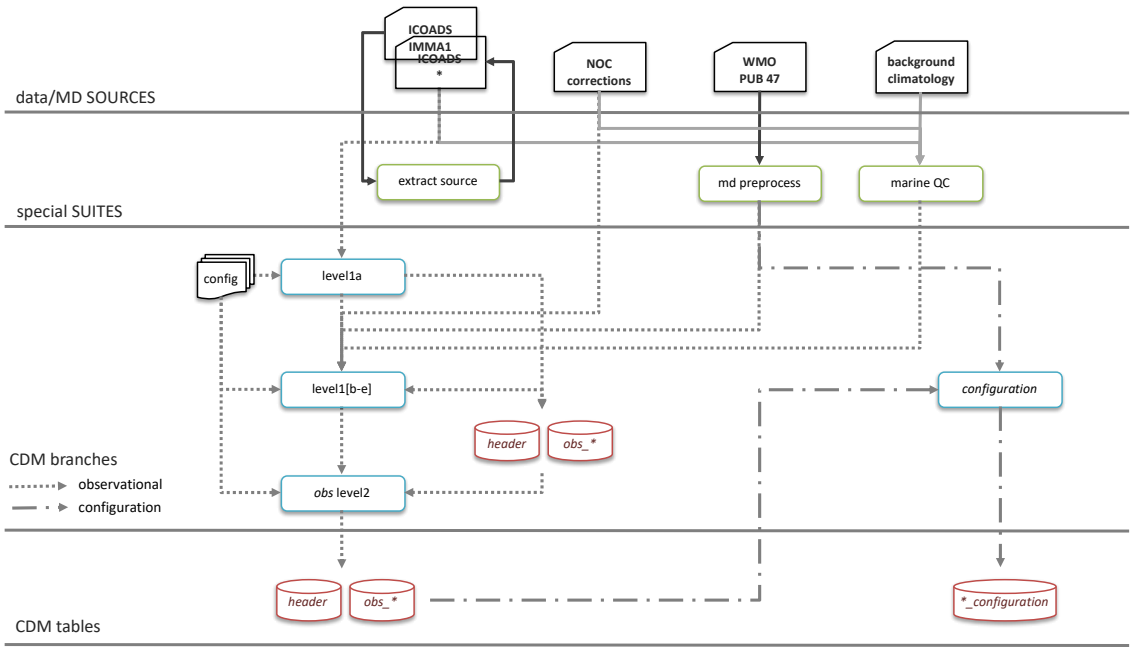
4.4 Marine data processing

Data released for the sub-daily met-ocean observations is based on the ICOADS Release 3 Total dataset for the period 1851 – 2010. More information on this dataset is provided in the Marine User Guide. In addition to providing access to the met-ocean observations, the data have been improved with:

- rescue of additional information from the ICOADS supplemental records;
- homogenization and merge with external metadata sources (WMO Publication 47);
- enhanced duplicate detection and linkage of observations from the same platform;
- enhanced QC procedures.

The marine processing is split into two main processing branches, one harmonizing the observations to be consistent with the common data model (CDM) and writing the data to files ready for ingestion to the database and the other processing and generating files containing the station, source and sensor configuration tables (Figure 5). For this release, whilst optimizing the data delivery, the data files have undergone a further processing to a simplified version of the CDM (CDMlite). This CDMlite excludes the configuration tables and contains only a subset of the header and observations tables entries as detailed previously.

Figure 5. Marine data processing scheme.



Within this version of the Technical Service Document we describe the processing applied to generate the 1851 – 1949 sub-period in data release_2.0. Section 4.4.1 links every data release to its corresponding version of the Technical Service Document.

Additionally, the software tools used to create the data summaries and figures that are included in the Marine User Guide, are maintained, starting from its Fourth version, in a separate repository (see Section 4.4.1). It also includes the necessary tools to generate the individual source-deck reports to evaluate the quality of the processed data prior to delivery to the CDS. Instructions on its use are included in the *docs* directory of the repository.

4.4.1 Release and software naming convention

Each release of met-ocean data has an associated release of the processing software available from the service software repository. The current and past releases are listed in Table 47 together with links to the software release.

Table 47(a). Technical service documents and marine software versions corresponding to each of the C3S data releases.

data release	**release update**	**technical service document**	**remote_repo_url**	**SW version**
Beta	NA	C3S_311a_Lot2.3.4.2-2018_201904_Second_version_Technical_service_document_v1.pdf	NA	
r092019	000000	C3S_311a_Lot2.3.4.2-2019_201911_Third_version_Technical_service_document_v1.pdf	https://github.com/glamod/glamod-**marine-processing**.git	v1.0
release_2.0	000000	C3S_311a_Lot2.3.4.2-2020_202004_Fourth_version_Technical_service_document_v1.pdf (this document)		v1.1

Table 47(b). Marine User Guide versions and corresponding software versions

Marine User Guide	**remote_repo_url**	**SW version**
C3S_311a_Lot2.3.4.2-2020_202004_Fourth_version_Marine_User_Guide_v1.pdf	https://github.com/glamod/marine-user-guide2	v1.0

4.4.2 List of abbreviations and paths

Table 48 and Table 49 list the common abbreviations and directory paths used in the marine processing documentation.

Table 48. List of abbreviations.

release	data release identifier. e.g. release_2.0
update	data release update identifier. e.g. 000000
dataset	original dataset or archive. e.g. ICOADS_R3.0.0T
user	linux user running the suites. e.g. iregon
release_sw	name used to name the directory where the marine SW is cloned to can be <*release*>, glamod-marine-processing[_<*version*>]…..

Table 49. Common marine directory paths.

gws	/group_workspaces/jasmin2/glamod_marine
gws_sm	/gws/smf/j04/c3s311a_lot2
gws_common	/gws/nopw/j04/c3s311a_lot2/
parent_code_path	<*gws_sm*>/code/marine_code
code_path	<*parent_code_path*>/<*release_sw*>
obs_code_path	<*code_path*>/obs-suite
metadata_code_path	<*code_path*>/metadata-suite
qc_code_path	<*code_path*>/qc-suite
data_path	<*gws*>/data
config_code_path	<*code_path*>/*config-suite*

Note that some of the above paths are specific to the facilities where the software and data are currently deployed (CEDA-JASMIN). If the software and data are ported to another system the paths

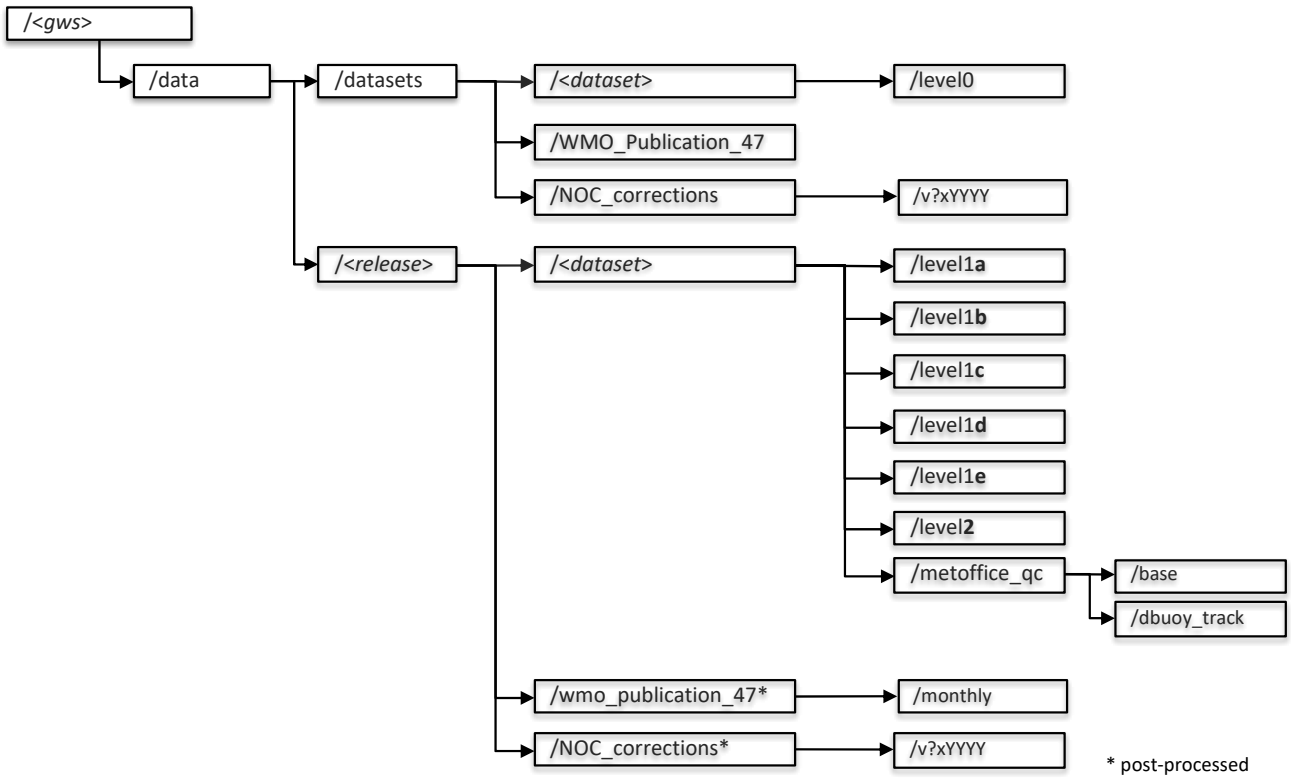
will need to be updated. *gws*, *gws_sm* and *gws_common* are specific to the current set up of JASMIN's group workspaces and the group workspaces assigned to this service.

4.4.3 Data layout

Figure 6 shows a schematic representation of the directory structure for the C3S in situ marine data expected by the processing software. There are two main components:

1. ***datasets***: contains the original datasets and ancillary information, metadata archives and any other auxiliary files in their original formats.
2. <***release***>: each release is stored in a dedicated directory split by source dataset, with the data further divided by processing level (see 4.4.5). These subdirectories include any data derived from the source input data, such as QC flags such as the Met Office QC scheme. The current release is based on a single source dataset (ICOADS R3.0.0T). In addition to the observational data files, the <release> directory contains data from the auxiliary datasets pre-processed to the format required by the processing software.

Figure 6. Marine data directory structure.

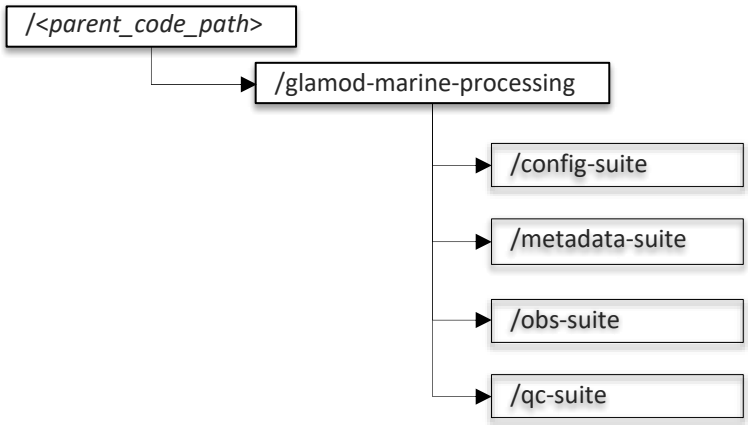


4.4.4 Software environment and processing software

The marine data processing software is currently formed of four different processing suites (Figure 7):

1) obs-suite: Set of python and shell scripts to harmonize and convert input observational data sources to CDM formatted files. This includes merging the CDM files with the output from the metadata and qc suites.
2) metadata-suite: Set of python and shell scripts to harmonize and reformat WMO Publication 47 metadata ready for merging with the observations.
3) qc-suite: Set of python and shell scripts to analyse the observations and set QC flags.
4) config-suite: Set of python and shell scripts to build the configuration (source, station and instrument) tables.

Figure 7. Marine code suites.



These are all included in the glamod-marine-processing repository. To install the processing suites they first need to be cloned from the repository indicated in Table 45:

```
cd <parent_code_path>
git clone <remote_repo_url> --branch <sw_version> --single-branch <release_sw>
```

Each suite forms a standalone set of python and shell scripts and it is recommended to create a new python virtual environment for each suite. The following sections describe the individual suites and initialisation of the virtual environments.

Observation processing suite

After cloning the repository, edit the `<obs_code_path>/setpaths.sh` file and set the environment variables as indicated in Table 50. This script sets the paths for the processing software and data files, including a scratch directory for the user running the software. Table 51 assigns the values for the environment variables to the paths defined in Table 50 (<…>).

Table 50. Environment variables used in the observation processing suite.

Environment variable	Setting
home_directory	*<gws>*
home_directory_smf	*<gws_smf>*
code_directory	*<obs_code_path>*
data_directory	*<data_path>*
scratch_path	/work/scratch-nopw/*<user>*

The `<obs_code_path>/setenv0.sh` script initialises the processing environment. It needs to be edited and the pyEnvironment_directory environmental variable set to the path of the corresponding python environment installation (`<obs_code_path>/pyenvs/env0`). It also needs to be modified to include the path to the system python libraries in the LD_LIBRARY_PATH variable.

Once the these scripts have been modified, the python virtual environment needs to be initialised. The following two blocks of commands need to be applied once for the requirements_env0.txt file in <*obs_code_path*>/pyenvs.

```
cd <obs_code_path>/pyenvs
module load jaspy/3.7
virtualenv –system-site-packages env0
```

The first command changes directory to the directory used to store the virtual environment files. The second loads the python (v3.7) software using the Modules Linux package. The final command initialises the virtual environment. Once initialised, the python environment needs to be activated and required python modules installed:

```
source <env>/bin/activate
pip install -r requirements_env0.txt
```

Four additional python modules have been developed for this service, Table 49 lists these modules and which versions have been used with the current data release. For each module listed the following need to be run:

```
cd <obs_code_path>/modules/python
git clone <remote_repo_url> --branch <module_version> --single-branch <module_local>
```

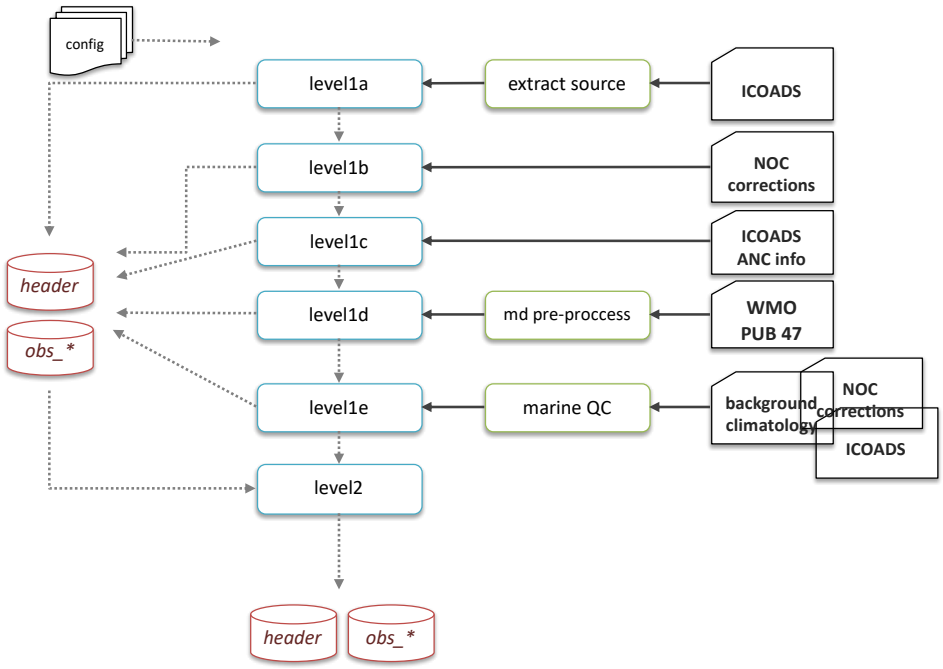
Table 51. Observational branch python modules versions.

SW version	**module**	**module_local**	**remote_repo_url**	**module_version**
v1.2	CDM mapper	cdm	git@git.noc.ac.uk:iregon/cdm-mapper.git	v1.2
	Data reader	mdf_reader	git@git.noc.ac.uk:iregon/mdf_reader.git	v1.2
	Metadata fixes	metmetpy	git@git.noc.ac.uk:iregon/metmetpy.git	v1.0
	Pandas operations	pandas_operations	git@git.noc.ac.uk:iregon/pandas_operations.git	v1.2

4.4.5 Running the observation processing branch

The **observation processing branch** is based on a set of chained processes, each step feeding into the next one and with the initial dataset (previously prepared for inclusion in the marine processing) transitioning through a series of levels from the first mapping to the CDM in level1a to the final set of curated observational CDM compatible files in level2 (header and observations-*). Some levels are fed with data from additional external datasets or with the output of dedicated suites that run concurrently to the overall scheme (Figure 8). The different levels are described below.

Figure 8. Observational branch.



- ***level1a:*** is the first mapping of the dataset to the CDM. Prior to mapping, the input data files are validated against the schema / data model and code tables defining the input. Reports failing this validation are discarded. For ICOADS, this includes also the rescue of any additional information from the supplemental attachment that has been identified as adding value. At this level the data are partitioned by date (monthly files), ICOADS source and card deck information and observed parameter. For each month a set of files is created containing the header and observation tables (header and `observations-[at|sst|dpt|wbt|slp|ws|wd]`).
- ***level1b:*** is the data improved with corrections and/or additional information resulting from the linkage and duplicate identification process. Reassignment of reports to different monthly files can result from this process after datetime corrections.
- ***level1c:*** is the data with metadata (currently primary station identification and datetime) validation performed and applied.

- ***level1d:*** data is enriched with external meta-data where available. For ship data the additional meta data source is WMO Publication 47 metadata.
- ***level1e:*** final quality control flags are added at this level, resulting from the position, parameter and tracking quality control processes.
- ***level2:*** data ready to ingest in the database. Data in level1e is inspected as data filtering might apply and part of the initial data set might be rejected to be inserted in the CDS database.

This section (and sub sections) describes the processing required to progress through the levels, with a description of the processing environment (paths, abbreviations, configuration files and launcher scripts) given first followed by the specific commands for each level.

Abbreviations and paths

Table 52 lists the abbreviations and directory paths specific to the observation processing branch. The abbreviations in angular brackets are given in earlier tables.

Table 52. Observational branch abbreviations and directory paths.

release_path	<*data_path*>/<*release*>
dataset_path	<*release_path*>/<*dataset*>
*level*_path*	<*dataset_path*>/level*
scripts_path	<*obs_code_path*>/scripts
config_files_path	<*obs_code_path*>/configuration_files/<*release*>_<*update*>/<*dataset*>

Configuration files

Configuration files manage what data is processed from a dataset in a data release and how it is processed. They are <*release*> (and <*update*>) and <*dataset*> specific and are located in directories <*config_files_path*>/<*release*>_<*update*>/<*dataset*>.

1. *process_list*: file source_deck_list.txt is an ascii file with list of ICOADS source (sid), deck(dck) pairs to process in a single column as <sid-dck>.
   It will typically contain the full list of <sid-dck> to include in a release, but can be split to process in batches.
2. *release_periods*: file release_periods.json is the file containing the start and end year of processing for each source and deck in the release.

3. *level1*_config:* files level1*.json have the processing options for the python script running each the level's process plus, optionally, options for submitting jobs for that process.
4. *level2_config:* file level2.json has the level2 data composition (this file is created after completing level1e data inspection as indicated below)

Launcher scripts

Due to the large volume of data and number of data files the processing is run using the JASMIN LOTUS cluster and LSF scheduler. The submission of jobs to the scheduler and processing is controlled via a series of shell scripts, hereafter referred to as launcher scripts.

Processing steps in from level1a to level1e, which are performed individually on the data partitions (by year, month, source, deck and variable), share the same launcher. This launcher script takes a different configuration file per level and submits an array of monthly subjobs for every source and deck included in the <*process_list*>:

- Every submitted subjob is the python script that performs the level specific processing as indicated in the level configuration file;
- The launcher stores the intermediate files resulting from the subjob submission (input and output bsub files) in *<scratch-path>*/*<level>*/sid-dck. *<scratch-path>* is generated by setpaths.sh and *<scratch-path>*/*<level>*/sid-dck cleaned on a sid-dck basis by the launcher.
- The launcher establishes the necessary dependencies to launch a job (process_array_output_hdlr.sh) to parse the bsub intermediate files in *<scratch-path>*/, rename them according to the job input file (yyyy-mm) and termination status and move them to the corresponding level log directory.

The complete calling sequence of this launcher is:

```
<scripts_path>/process_array_launcher.sh  <level1*_config> <release_periods> \
    <process_list> -f 0|1 -r 0|1 -s yyyy -e yyyy
```

The optional arguments to this script are:

- `-f`: is used to trigger the failed only mode in job submission. Setting `-f` to `0` will only submit subjobs failed in a previous run, those with a `failed` extension in its log file.
- `-r`: is used to trigger the cleaning script once all jobs in the job array have finished successfully. Setting `-r` to `0` will remove all the preceding level files, setting to `1` (or not including it) will leave the files in place. This option is disabled to remove initial dataset data files when running level1a and is only recommended to be used from level1c onwards.
- `-s`: start processing year (defaults to initial year declared in file `<release_periods>`)
- `-e`: end processing year (defaults to end year declared in file `<release_periods>`)

Setting up a release and dataset

Every new release or new dataset in a release needs to have its corresponding directory structure initialised to accommodate the new data holdings with:

```
source <obs_code_path>/setpaths.sh
source <obs_code_path>/setenv0.sh
python <scripts_path>/make_release_source_tree.py  <data_path> <release> \
    <dataset> <release_periods>
```

This script does not overwrite existing directories and is safe to run on an existing directory structure.

level1a

Level 1a contains the initial data converted from the input data sources (level0) to files compatible with the CDM. For the ICOADS / IMMA formatted files processed for the first data release the files are converted with the following commands:

```
source <obs_code_path>/setpaths.sh
source <obs_code_path>/setenv0.sh
python <scripts_path>/level1a.py <level1a_config> <data_path> sid-dck year month
```

where `sid-dck` is the ICOADS source ID and deck to process, `year` the year of the data file to process and `month` the month of the data file to process. To facilitate the processing of a large number of files `level1a.py` can be run in batch mode:

```
<scripts_path>/process_array_launcher.sh  <level1a_config> <release_periods>  \
    <process_list> -f 0|1 -s yyyy -e yyyy
```

This executes an array of monthly subjobs per ICOADS source and deck included in the `<process_list>` and logs to `<data_path>/<release>/<dataset>/level1a/log/sid-dck/`. Log files are `yyyy-mm-<release>-<update>.ext` with `ext` either `ok` or `failed` depending on the subjob termination status.

List *.failed in the *sid-dck* level1a log directories to find if any went wrong.

level1b

Level 1b integrates external files containing enhanced information on the duplicate status of the observations, corrected date/time and locations, and linked station IDs with the level1a data. As part of the integration a weather report may move between months if an error in the date had previously been identified and the correct data is for a different month. A description of the processing used to generate these external files is described in the marine duplication identification document (available upon request, to be published shortly). It should be noted that this processing is currently external to C3S311a_lot2 but will be integrated in a future release.

The external files need to be copied to the datasets directory prior to processing. Once copied to the required directory structure the files need to be reformatted for integration with the level1a files. As with level1a, the processing is done via python and shell scripts using the LSF scheduler:

```
source <obs_code_path>/setpaths.sh
source <obs_code_path>/setenv0.sh
bsub -J <option>_cor -oo <scratch_path>/<option>_cor.o -eo <scratch_path>/<option>_cor.o \
-q short-serial -W 03:00 -M 1000 <scripts_path>/noc_corrections_postprocess.sh <release>
<cor_version> <opt> <year_init> <year_end>
```

Where `<opt>` is one of `id`, `datepos` or `duplicates` and *`cor_version`* is `v1x2019` for the current data release. Table 53 shows the location of the correction files before and after reformatting.

Table 53. Location of correction files before and after reformatting.

BEFORE in datasets/NOC_corrections/*<cor_version>*	***AFTER*** in *<release_path>*/NOC_corrections/*<cor_version>*
./**CHANGED_DATEPOS**	./**timestamp/**
	./**latitude/**
	./**longitude/**
./**CHANGED_IDS_UPDATE2**	./**id/**
./**DUP_FILES**	

BEFORE in datasets/NOC_corrections/*<cor_version>*	***AFTER*** in *<release_path>*/NOC_corrections/*<cor_version>*
./**DUP_FILES_ID**	./**duplicate_flags/** ./**duplicates/**

The reformatted files are merged with the level1a data by the following commands:

```
source <obs_code_path>/setpaths.sh
source <obs_code_path>/setenv0.sh
python <scripts_path>/level1b.py <level1b_config> <data_path> sid-dck year month
```

where `sid-dck` is the ICOADS source ID and deck to process, `year` the year of the data file to process and `month` the month of the data file to process. To facilitate the processing of a large number of files `level1b.py` can be run in batch mode:

```
<scripts_path>/process_array_launcher.sh  <level1b_config> <release_periods>  \
    <process_list> -f 0|1 -s yyyy -e yyyy
```

This executes an array of monthly subjobs per ICOADS source and deck included in the `<process_list>` and logs to *<data_path>*/*<release>*/*<dataset>*/level1b/log/*sid-dck*/. Log files are *yyyy-mm*-<release>-<update>.*ext* with *ext* either `ok` or `failed` depending on the subjob termination status.

List *.failed in the *sid-dck* level1b log directories to find if any went wrong.

level1c

The level1c files contain reports from level1b that have been further validated following corrections to the date/time, location and station ID. Those failing validation are rejected and archived for future analysis. Additionally, datetime corrections applied previously in level1b, can potentially result in reports being relocated to a different month. These reports are moved to their correct monthly file in this level.

To generate level1c files, the individual sid-dck monthly files in level1b are processed with:

```
source <obs_code_path>/setpaths.sh
source <obs_code_path>/setenv0.sh
python <scripts_path>/level1c.py <level1c_config> <data_path> sid-dck year month
```

where `sid-dck` is the ICOADS source ID and deck to process, `year` the year of the data file to process and `month` the month of the data file to process.

As with other levels, the processing to level1c can be run in batch mode:

```
<scripts_path>/process_array_launcher.sh  <level1c_config> <release_periods>  \
    <process_list> -f 0|1 -s yyyy -e yyyy -r 0|1
```

This executes an array of monthly subjobs per ICOADS source and deck included in the `<process_list>` and logs to *<data_path>*/*<release>*/*<dataset>*/level1c/log/*sid-dck*/. Log files are *yyyy-mm*-<release>-<update>.*ext* with *ext* either `ok` or `failed` depending on the subjob termination status.

List *.failed in the *sid-dck* level1c log directories to find if any went wrong.

level1d

The level1d files contain the level1c data merged with external metadata (where available). For this release, the level1c have been merged with WMO Publication 47 metadata to set instrument heights, station names and platform sub types (i.e. type of ship). Prior to merging, the WMO Publication 47 metadata have been harmonised, quality controlled and pre-processed (see Metadata pre-processing in 4.4.6). These pre-processed files are merged with the level1c data with the following command:

```
source <obs_code_path>/setpaths.sh
source <obs_code_path>/setenv0.sh
python <scripts_path>/level1d.py <level1d_config> <data_path> sid-dck year month
```

where `sid-dck` is the ICOADS source ID and deck to process, `year` the year of the data file to process and `month` the month of the data file to process.

The launcher script to run the full level1d processing is run with:

```
<scripts_path>/process_array_launcher.sh  <level1d_config> <release_periods>  \
<process_list> -f 0|1 -s yyyy -e yyyy -r 0|1
```

This executes an array of monthly subjobs per ICOADS source and deck included in the `<process_list>` and logs to *<data_path>*`/`*<release>*`/`*<dataset>*`/level1d/log/`*sid-dck*`/`. Log files are *yyyy-mm*`-<release>-<update>.`*ext* with *ext* either `ok` or `failed` depending on the subjob termination status.

List *.failed in the *sid-dck* level1d log directories to find if any went wrong.

level1e

The level1e processing merges the data quality flags from the Met Office QC suite (see Quality control in 4.4.6) with the data from level 1d. The QC software generates two sets of QC files, one basic QC of the observations from all platforms and an enhanced track and quality check for drifting buoy data. The basic QC flags are merged with the following script:

```
source <obs_code_path>/setpaths.sh
source <obs_code_path>/setenv0.sh
python <scripts_path>/level1e.py <level1e_config> <data_path> sid-dck year month
```

where `sid-dck` is the ICOADS source ID and deck to process, `year` the year of the data file to process and `month` the month of the data file to process.

The launcher script for level1e is run with:

```
<scripts_path>/process_array_launcher.sh  <level1e_config> <release_periods>  \
<process_list> | -f 0|1 | -s yyyy | -e yyyy | -r 0|1
```

This executes an array of monthly subjobs per ICOADS source and deck included in the `<process_list>` and logs to *<data_path>*`/`*<release>*`/`*<dataset>*`/level1e/log/`*sid-dck*`/`. Log files are *yyyy-mm*`-<release>-<update>.`*ext* with *ext* either `ok` or `failed` depending on the subjob termination status.

List *.failed in the *sid-dck* level1e log directories to find if any went wrong.

Create and inspect the individual source-deck reports on the data in level1e (see software repository in 4.4.1) to evaluate the quality of the data, before proceeding to level2.

After the basic QC flags have been merged the enhanced drifting buoy flags need to be merged with the level1e data. This process is described under Quality control in 4.4.6 but will be moved to the level1e processing in a future update. Once the drifting buoy flags have been merged the data files will no longer change and summary data reports (see **Error! Reference source not found.** in this section) need to be generated prior to the data moving to level2.

level2

After visual inspection of the reports generated in level1e, only observation tables reaching a minimum quality standard proceed to level2: this might imply rejecting a full sid-dck dataset or an observational table or change the period of data to release. The level1e data composition that has been used to generate the level2 product of every release is configured in `level2.json` file available in `<config_files_path>/<release>_<update>/<dataset>` (*level2_config*). Prior to first use this file needs to be created. This can be done using the following commands:

```
source <obs_code_path>/setpaths.sh
source <obs_code_path>/setenv0.sh
python level2_config.py <release_periods> year_ini year_end
```

This script creates the selection file `level2.json` in the execution directory. The parameters `year_init` and `year_end` are used to set the final period of data release, that might be different to that initially processed. The data period of each of the individual source-deck data partition is adjusted in the level2.json file according to these arguments. After checking data quality on the level1e reports, edit the data selection file as needed to create the final dataset composition:

- Remove a full sid-dck from level2 by setting to *true* the sid-dck 'exclude' tag.
- Remove an observation table from the full dataset by adding it to the list under the general 'params_exclude' tag.
- Remove an observation table from a sid-dck by adding it to the list under the sid-dck 'params_exclude' tag.
- Adjust the release period of a sid-dck by modifying the 'year_init|end' tags of the sid-dck
- Observation tables to be removed have to be named as observations-[at|sst|dpt|wbt|wd|ws|slp]

- All edits need to be consistent with JSON formatting rules.

Once file *level2.json* has been edited the file needs to be copied to `<config_files_path>/<release>_<update>/<dataset>` for it to be version controlled. The level 2 processing is then run using:

```
source <obs_code_path>/setpaths.sh
source <obs_code_path>/setenv0.sh
python <scripts_path>/level2.py <data_path> <release> <update> <dataset> sid-dck \
<level2_config>
```

where `sid-dck` is the ICOADS source ID and deck to process. The launcher script for level2 is run with:

```
<scripts_path>/level2.sh <release> <update> <dataset> <level2_config> <process_list>
```

This executes a job per ICOADS source and deck included in the `<process_list>` and logs to `<data_path>/<release>/<dataset>/level2/log/sid-dck/level2.o.`

Grep exit in level2.o files in the `sid-dck` level2 log directories to find if any went wrong.

4.4.6 Special suites

Extract source

Extraction of the individual sources and decks from ICOADS is described in 4.1.2. In a future release this processing will be updated to be part of the observation processing suite.

Metadata pre-processing

WMO Publication 47 (WMO47 hereafter) contains instrumental and platform level metadata for the Voluntary Observing Ships and has been published at regular intervals by the WMO since 1956. A full description of the information available is given in Kent *et al.* (2007). As part of the C3S 311a Lot 2 service this metadata has been processed and harmonised following Berry and Kent (2006). As with the observations, the processing of WMO47 has been performed using a suite of python scripts in a

virtual environment and these have been included in the marine processing repository described above. When the repository is first cloned (section 0) the virtual environment needs to be setup:

```
cd <metadata_code_path>/pyenvs
module load jaspy/3.7
virtualenv --system-site-packages env0
source env0/bin/activate
pip install -r requirements_<env>.txt
```

Before the scripts can be run a working directory needs to be set up and the required files copied to the working directory:

```
mkdir <gws>/working/metadata-suite/
cd <gws>/working/metadata-suite/
mkdir logs logs/failed logs/successful
mkdir logs2 logs2/failed logs2/successful
mkdir merge_logs merge_logs/failed merge_logs/successful
mkdir extract_logs extract_logs/failed extract_logs/successful
cp <metadata_code_path>/lotus_scripts/*.sh ./
cp <metadata_code_path>/setenv0.sh ./
```

Where `<gws>` is the path to the group workspace. The processing of WMO47 can now proceed and is based on three processing steps. The first step splits the individual metadata files into files based on the country that has recruited the VOS, removes duplicate entries within each file and harmonizes the code tables used. The second step combines the files to one master file per contributing country, assigns validity dates and assigns a unique ID to each record. The country master files are also combined to one overall master file as part of this process. The third step extracts monthly files from the master file for merging with the observational branch. Each step is run on the LOTUS cluster at JASMIN.

Step 1:

```
cd <gws>/working/metadata-suite/
bsub < submit_split.sh
```

Log files are written to the `logs` and `logs2` directories created earlier, with the LSF logs written to logs. Deviations from the expected code tables are written to files in the logs2 directory, a small number are expected due to the operational nature of WMO47 but these files should be checked. The output data files are written to `<gws>/data/wmo_publication_47/split/`. The LSF jobs status is indicated by files of the format `split_<jobIDX>.success` and `split_<jobIDX>.failed` where `_<jobIDX>` is the job array index number. If no `split_<jobIDX>.failed` files are created processing can proceed to step 2 otherwise the cause of failure needs to be identified and the jobs resubmitted. Only failed jobs are resubmitted by the job submission script.

Step 2:

```
bsub < submit_merge.sh
```

Log files are written to the `merge_logs` directory and job status indicated by the presence of `merge_<jobIDX>.success` and `merge_<jobIDX>.failed` directory. As with step 1, if any `merge_<jobIDX>.failed` files are created the cause of the failure needs to be identified and the jobs resubmitted. Output data files are written to `<gws>/data/wmo_publication_47/master/`.

Step 3:

```
bsub < submit_extract.sh
```

Log files are written to the `extract_logs` directory and job status indicated by the presence of `extract_<jobIDX>.success` and `extract_<jobIDX>.failed` directory. As with the other steps, if any `extract_<jobIDX>.failed` files are created the cause of the failure needs to be identified and the jobs resubmitted. Output data files are written to `<gws>/data/wmo_publication_47/monthly/`.

Once all processing steps have been completed `the *.success` and `*.failed` files can be removed.

Quality control

The quality control applied is described in Kennedy et al., (2017). As with the other suites the quality control suite needs to be setup prior to use. First the python virtual environment needs to be setup and required packages installed:

```
cd <qc_code_path>/pyenvs
module load jaspy/3.7
virtualenv --system-site-packages env0
source env0/bin/activate
pip3 install -r requirements_<env>.txt
```

Next, the working directory needs to be configured for running the QC suite:

```
cd <gws>/working/qc_suite
mkdir corrected_data tracking # scratch area for intermdediate files
mkdir logs_qc logs_qc_hr logs_pp # log directories
mkdir logs_recombine logs_merge logs_reindex logs_merge_lvl1e # log directories
```

Prior to running the Met Office Quality Control suite the ICOADS input data files need to be merged with the NOC correction files (see level1b in 4.4.5). These files contain corrections to callsigns / platform identifiers, linking observations from the same platform together and correcting the date / time for known errors. The input data files and NOC corrections files are merged using *<qc_code_path>*/scripts/preprocess.py script. For ease of use an LSF submission script is included in the repository and can be run with:

```
cd <gws>/working/qc-suite/
bsub < submit_preprocessing.sh
```

The output from this is written to the `corrected_data` directory under `<gws>/working/qc-suite/`. Log files are written to the `logs_pp` directory and the status of the jobs indicated by the presence of files of the form `preprocess_<jobIdx>.success` and `preprocess_<jobIdx>.failed`, where `<jobIdx>` indicates the job array index number. If no `preprocess_<jobIdx>.failed` files are created the processing can move to the QC stage. If any `preprocess_<jobIdx>.failed` files are created the cause of failure needs to be resolved and the jobs resubmitted. Only those that have failed will be rerun by the submission script. Once the pre-processing has run without issues the output files need to be recombined to single monthly files:

```
cd <gws>/working/qc-suite/
bsub < combine.sh
```

Once the process has finished the log files, written to `<gws>/working/qc-suite/logs_recombine/`, should be checked and if no errors are present the Met Office QC suite can then be run:

```
cd <gws>/working/qc-suite/
bsub < submit_qc.sh
bsub < submit_qc_hr.sh
```

This runs both the regular and high-resolution Met Office QC suites. Log files are written to the `logs_qc` and `logs_qc_hr` directories and the job status indicated by the presence of `qc_<jobIdx>.success`, `qc_<jobIdx>.failed`, `qc_hr_<jobIdx>.success` and `qc_hr_<jobIdx>.failed` files. Again, the cause of any failure needs to be identified by checking the logs files, resolved and the jobs resubmitted. Only those that have failed will be rerun.

For drifting buoys a final set of enhanced track checking and QC checks are performed, checking for issues such as the buoy having run aground or sensor failures. The QC checks are run using:

```
cd <gws_common>/code/marine_code/PRE_C3S/Met_Office_Marine_QC/
python Tracking_QC_wrapper.py
```

The merge of the basic QC flags with the observations is described in Section 4.4.5.9 but due to the organisation of the output from the enhanced drifting buoy QC some additional steps are required. These are currently included in the QC suite but will be moved to the observation processing suite in a later release.

Prior to merging the drifting buoy QC flags with the level 1e data the flags need to be re-indexed and merged from one file per drifter to one file per calendar month. This is performed in several steps. First, the data are split to multiple monthly files:

```
cd <gws>/working/qc-suite/
bsub < submit_reindex.sh
```

Log files are written to `<gws>/working/qc-suite/logs_reindex` and the output to monthly files under `<gws>/working/qc-suite/tracking/`. As with the QC processing, success or failure is indicated by the presence of `reindex_<jobIdx>.success` and `reindex_<jobIdx>.failed` files. Due to limitations on the parallel writing of files, one file per LSF job per month is written. These are combined with:

```
cd <gws>/working/qc-suite/
```

```
bsub < submit_merge.sh
```

to create one file per month, logs are written to `<gws>/working/qc-suite/logs_merge`. The final step merges the monthly files with the level1e data:

```
cd <gws>/working/qc-suite/
bsub < submit_merge_lvl1e.sh
```

Currently the data are written to a temporary directory (*<data_path><release><dataset>*`/level1e_tmp/063-714/`). Provided there are no exceptions the data can be copied to the level1e directory for the drifting buoy sid-dck combination (*<data_path><release><dataset>*`/level1e/063-714/`) to replace the existing files.

Configuration Tables

The configuration tables have been generated using the scripts under the *config_code_path*. Before use the python environment needs to be initialised:

```
cd <config_code_path>/pyenvs
module load jaspy/3.7
virtualenv --system-site-packages env0
source env0/bin/activate
pip3 install -r requirements.txt
```

Next the working directories need to be configured:

```
mkdir <gws>/working/config-suite/
cd <gws>/working/config-suite/
mkdir logs_sources logs_concat_yr logs_concat_all logs_stations
mkdir sources out
```

and the lotus job submission scripts copied to the working directory:

```
cp <config_code_path>/lotus_scripts/*.sh ./
```

The scripts to create the station configuration tables are run in three steps. The first combines the WMO Publication 47 metadata with the ICOADS data files to create monthly summaries by station / platform ID:

```
cd <gws>/working/config-suite/
bsub < submit_stations.sh
```

The next step concatenates and merges the monthly summaries to create annual summaries:

```
bsub < submit_concat_yr.sh
```

And the final step concatenates to form a single station configuration file:

```
bsub < submit_concat_all.sh
```

As with the other marine processing stages the log files need to be checked at each step, any errors identified and resolved and the processing rerun.

The source configuration table is generated in a single step:

```
bsub < submit_sources.sh
```

Once completed successfully the data files need to be copied to the release directory:

```
cp ./out/station_configuration.csv <dataset_path>/level2/configuration/
cp ./source.psv <dataset_path>/level2/configuration/
```

4.5 Loading data into the database

The instructions here are for the CDMlite rather than the full CDM. The loading consists of a number of stages that are managed on different parts of JASMIN.

NOTE: This section assumes that the database has been created and the partitions have been set up using the Ansible Playbook (see Section 3.3).

All the code for ingesting the marine and land data is available from the following repository:

https://github.com/glamod/glamod-ingest/

4.5.1 Loading the marine data

Re-structuring the input files

The first stage involves restructuring the input data so that the files are in a structure that can be directly loaded into PostgreSQL database partitions (using the "\COPY" SQL command). This involves grouping all the source decks data into single directories by year. The script is run on LOTUS and the intermediate data is written to the `/work/scratch-nopw/` directory:

```
cd $WORK_DIR/
git clone https:// github.com/glamod/glamod-ingest
cd glamod-ingest/
export RELEASE=r2.0
nohup ./scripts/marine/prepare-all-marine-lite.sh $RELEASE batch 2>&1 >
       prepare.marine.output.txt &
```

These tasks are run on the high-memory nodes (as defined in the script) and can take up 24 hours to complete. The restructuring task also includes various, functions, checks and fixes such as:

- Merge header and observation records

- Set "report_type" to 0
- Set the "height_above_surface" value from input fields
- Interpret a number of missing value indicators and set to NULL

Generating SQL scripts

After restructuring the files, a separate script is run as a set of LOTUS jobs to generate SQL scripts that will load the restructured pipe-separated (PSV) files:

```
cd $WORK_DIR/glamod-ingest/
./scripts/marine/create-sqls-marine.sh $RELEASE r2.0
```

This writes a set of commands to load entire PSV files directly in to database partitions, e.g.:

```
$ cat /gws/nopw/j04/c3s311a_lot2/data/ingest/marine/sql/0/load-0-1999.sql
\cd '/work/scratch-nopw/astephen/glamod/r2.0/cdmlite/prepare/marine/1999/'
\COPY lite_2_0.observations_1999_marine_0 FROM '063-714-1999-r2.0-000000.psv' WITH CSV HEADER
DELIMITER AS '|' NULL AS 'NULL'
\COPY lite_2_0.observations_1999_marine_0 FROM '100-792-1999-r2.0-000000.psv' WITH CSV HEADER
DELIMITER AS '|' NULL AS 'NULL'
\COPY lite_2_0.observations_1999_marine_0 FROM '103-792-1999-r2.0-000000.psv' WITH CSV HEADER
DELIMITER AS '|' NULL AS 'NULL'
\COPY lite_2_0.observations_1999_marine_0 FROM '112-926-1999-r2.0-000000.psv' WITH CSV HEADER
DELIMITER AS '|' NULL AS 'NULL'
\COPY lite_2_0.observations_1999_marine_0 FROM '113-927-1999-r2.0-000000.psv' WITH CSV HEADER
DELIMITER AS '|' NULL AS 'NULL'
\COPY lite_2_0.observations_1999_marine_0 FROM '114-992-1999-r2.0-000000.psv' WITH CSV HEADER
DELIMITER AS '|' NULL AS 'NULL'
```

Loading the data into the database partitions

The loader script is run as a single process for marine data. It is spawned under "nohup" so that it will complete even if the SSH connection to the server is interrupted:

```
cd $WORK_DIR/glamod-ingest/
nohup ./scripts/marine/load-marine-sql.sh 0 > load.marine.0.txt &
```

Log files are written to:

```
/gws/nopw/j04/c3s311a_lot2/data/ingest/marine/populate/
```

The logs can be analysed for any errors. A successful process will report the number of records copied in to the database partition per PSV file, e.g.:

```
$ cat /gws/nopw/j04/c3s311a_lot2/data/ingest/marine/populate/load-0-1999.sql.log
COPY 1964636
COPY 1175883
COPY 50241
COPY 4942575
COPY 172003
COPY 814021
```

4.5.2 Loading the land data

Re-structuring the input files

The first stage involves restructuring the input data so that the files are in a structure that can be directly loaded into PostgreSQL database partitions (using the "\COPY" SQL command).

Since there are hundreds of thousands of input files, grouped by station, it is important to batch them into groups that can be managed as processing units on LOTUS. This is done with the script:

```
git clone https://github.com/glamod/glamod-ingest

cd glamod-ingest/

# set environment
source setup-env-sci.sh
python scripts/land/decide-land-batches.py
```

The batch details are written to:

```
/gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/batches/cdmlite_batch_rules.txt
```

Some example batches are:

```
$ head -3 /gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/batches/cdmlite_batch_rules.txt

path_prefix|batch_id|of_n_batches|batch_length

/gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/cdm_lite/daily/CDM_lite_SecondRelease_ACW00011*|daily-CDM_lite_SecondRelease_ACW00011|5567|2

/gws/nopw/j04/c3s311a_lot2/data/level2/land/r202005/cdm_lite/daily/CDM_lite_SecondRelease_AE000041*|daily-CDM_lite_SecondRelease_AE000041|5567|1
```

The restructure script uses those batch identifiers when running all the jobs on LOTUS and the intermediate data is written to the `/work/scratch-nompiio/` directory:

```
cd $WORK_DIR/glamod-ingest/

RELEASE=r2.0

./scripts/land/prepare-all-land-lite.sh $RELEASE batch
```

These tasks can take up 12 hours to complete. The restructuring task also includes various, functions, checks and fixes such as:

- Set "report_type" field
- Set "platform_type" field
- Set the "height_above_surface" value from input fields

Generating SQL scripts

After restructuring the files, a separate script is run as a set of LOTUS jobs to generate SQL scripts that will load the restructured pipe-separated (PSV) files:

```
cd $WORK_DIR/glamod-ingest/

./scripts/land/create-sqls-land.sh r2.0 0
./scripts/land/create-sqls-land.sh r2.0 2
./scripts/land/create-sqls-land.sh r2.0 3
```

This writes a set of commands to load entire PSV files directly in to database partitions, e.g.:

```
$ head -3 /gws/smf/j04/c3s311a_lot2/workflow/r2.0/lite/land/sql/outputs/0/load-0-1999.sql

\cd '/work/scratch-nopw/astephen/glamod/r2.0/cdmlite/prepare/land/0/1999/'

\COPY lite_2_0.observations_1999_land_0 FROM '0-1999-sub_daily-
        CDM_lite_SecondRelease_AAI0000TN.psv' WITH CSV HEADER DELIMITER AS '|' NULL AS
        'NULL'

\COPY lite_2_0.observations_1999_land_0 FROM '0-1999-sub_daily-
        CDM_lite_SecondRelease_ACI0000TA.psv' WITH CSV HEADER DELIMITER AS '|' NULL AS
        'NULL'
```

Loading the data into the database partitions

The loader script is run as three scripts in parallel for the land data. Each script is spawned under "nohup" so that they will complete even if the SSH connection to the server is interrupted:

```
cd $WORK_DIR/glamod-ingest/
RELEASE=r2.0
for report_type in 0 2 3; do
    nohup ./scripts/land/load-land-sql.sh $RELEASE $report_type >
                load-land-${report_type}.output.txt &
done
```

Log files are written to:

```
/gws/smf/j04/c3s311a_lot2/workflow/r2.0/lite/land/populate/log/
```

The logs can be analysed for any errors. A successful process will report the number of records copied in to the database partition per PSV file, e.g.:

```
$ cat /gws/smf/j04/c3s311a_lot2/workflow/r2.0/lite/land/populate/log/load-0-1801.sql.log
COPY 730
COPY 1095
COPY 251
COPY 1095
COPY 1093
```

4.6 Optimising the database

There are a number of changes made to the operational database in order to improve performance. These are recorded here.

4.6.1 Use of partitions

Since the "observations_table" and "header_table" would be Tbytes in volume it is imperative that these are partitioned. The partitioning script is used to generate partitions by:

- Land/marine
- Year
- Report type

This partitioning is managed by the Ansible playbook that makes the DB.

4.6.2 Sorting (clustering) of partitions

The tables are sorted by date/time. In PostgreSQL this is known as "clustering". This script manages clustering:

https://github.com/glamod/glamod-ingest/blob/master/src/db-maker/make-cluster-sqls.py

4.6.3 Indexing of partitions

Each partition is indexed on the following fields:

- Date/time
- Observed variable
- Location

The indexes are managed by the Ansible playbook that makes the DB.

4.6.4 Database configuration

The following settings were changed in the database configuration file to improve performance when processing large memory queries:

In: /var/database/data/postgresql.conf

```
shared_buffers = 6GB
work_mem = 2048MB
```

5. People

Role	Details	In C3S311a Lot 2	Comments
Computing Service Manager	Technical person responsible for making sure the service is running, and for monitoring.	Ag Stephens	
System manager	Responsible for patching the servers	Peter Chiu	
Computing Service owner	Person who "owns" the requirement for the service to be running, e.g. from an SLA point of view, not necessarily the same as the service manager.	Peter Thorne	
Others	Names of any other people associated with this service writers, requesters etc.	David Berry (developer) Simon Noone (developer) Anthony Kettle (developer)	

6. Links

This technical service document is accompanied by two data user guides – one each for the land, (C3S_D311a_Lot2.3.4.3_2020_202007_Fourth_version_Land_Data_User_guide _v1) and the marine data holdings, (C3S_D311a_Lot2.3.4.4_2020_202006_Fourth_version_Marine_User_Guide_v1) which can be found on their respective CDS catalogue entries. Taken together with the current document these documents provide an end-to-end description of the processing system and its maintenance.

7. Other information

Anything not covered by the above sections shall be added here.

8. References

Berry, D.I. and Kent, E.C. (2006) *WMO Publication 47: Consistency Checking and Gap Filling, Version 1.0* Southampton, GB. Southampton Oceanography Centre 77pp. (available from https://eprints.soton.ac.uk/341419/)

Chamberlain S. (2017), rnoaa: 'NOAA' Weather Data from R. R package version 0.7.0. https://CRAN.R-project.org/package=rnoaa

Dunn, R. J. H., et al. (2016), Expanding HadISD: quality-controlled, sub-daily station data from 1931, Geoscientific Instrumentation, Methods and Data Systems, 5, 473-491.

Dunn, R. J. H., et al. (2012), HadISD: A Quality Controlled global synoptic report database for selected variables at long-term stations from 1973-2011, Climate of the Past, 8, 1649-1679.

Dunn, R. J. H., Willett, K. M., Morice, C. P., and Parker, D. E.: (2014), Pairwise homogeneity assessment of HadISD, Clim. Past, 10, 1501-1522, https://doi.org/10.5194/cp-10-1501-2014.

Durre, I., M. J. Menne, B. E. Gleason, T. G. Houston, and R. S. Vose, (2010), Comprehensive automated quality assurance of daily surface observations. J. Appl. Meteor. Climatol., 49, 1615–1633, doi:10.1175/2010JAMC2375.1.

Freeman E, Woodruff SD, Worley SJ, Lubker SJ, Kent EC, Angel WE, Berry DI, Brohan P, Eastman R, Gates L, Gloeden W, Ji Z, Lawrimore J, Rayner NA, Rosenhagen G, Smith SR. (2017), ICOADS Release 3.0: A Major Update to the Historical Marine Climate Record, *International Journal of Climatology*, 37, 2211–2232. DOI:10.1002/joc.4775

Kennedy, J. J., C. Atkinson and K. Willett, 2017: Marine Data System Quality Control, UK Met Office technical note, 23pp.

Kent, Elizabeth C.; Woodruff, Scott D.; Berry, David I. 2007. Metadata from WMO Publication No. 47 and an Assessment of Voluntary Observing Ships Observation Heights in ICOADS, *Journal of Atmospheric and Oceanic Technology*, 24 (2). 214-234. doi:10.1175/JTECH1949.1.

Menne, M.J., I. Durre, R.S. Vose, B.E. Gleason, and T.G. Houston, (2012), An Overview of the Global Historical Climatology Network-Daily Database. J. Atmos. Oceanic Technol., 29, 897–910, https://doi.org/10.1175/JTECH-D-11-00103.1.

Rennie J.J. et al., (2014), The international surface temperature initiative global land surface databank: monthly temperature data release description and methods, Geoscience Data Journal, 1, 75-102.

Copernicus Climate Change Service

ECMWF - Shinfield Park, Reading RG2 9AX, UK

Contact: info@copernicus-climate.eu

climate.copernicus.eu copernicus.eu ecmwf.int