NWP SAF

AAPP Pre-processing Module for Arctic Weather Satellite (AWS)

AAPP-AWS User Manual

Version: 1.0

20 January 2025

AAPP-AWS User Manual

Change record			
Version	Date	Author / changed by	Remarks
0.1	30/10/2023	Nigel Atkinson	Initial draft
0.2	29/02/2024	Nigel Atkinson	Add BUFR read capability
0.3	22/10/2024	Nigel Atkinson	AAPP-AWS code packaged with core AAPP
1.0	20/01/2025	Nigel Atkinson	Release version

Table of Contents

1. INTRODUCTION

This document is the user manual for the AAPP pre-processing module for the Arctic Weather Satellite (AWS). The software is referred to as AAPP-AWS.

It incorporates the following elements:

- Installation Guide (section 2)
- Run-time instructions (section 3)

1.1 Reference documents

RD-1: Description of AWS:
https://www.esa.int/Applications/Observing_the_Earth/Meteorological_missions/Arctic_Weather_Satellite/The_instrument

RD-2: MWIPP User Manual:
https://nwp-saf.eumetsat.int/site/software/mwipp/

1.2 Functionality summary

AAPP-AWS has the following functionality:

- Ingest level 1B data from AWS, in netCDF or BUFR format.
- Re-map the brightness temperatures from the four feedhorns to a common geolocation, creating a level 1C product.
- BUFR encode at level 1B or level 1C, using a BUFR template developed originally for the TROPICS mission.
- If the input file is netCDF, then an option is provided to append the re-mapped brightness temperatures to the input file.
- For the 1C product, an option is available to perform an $n \times n$ spatial average for selected channels, if the user wishes to reduce the noise in these channels.

The re-mapping step comprises only a bilinear interpolation – it does not change any of the beam widths. The re-mapping is based on code from the NWP SAF’s MWIPP package, see RD-2 section 5.5.

In order to establish the re-mapping parameters, the software assumes that the input data are in the correct time order and that the latitudes and longitudes are valid for about 30 successive scans in the centre of the time period.

1.3 System requirements

To install and run AAPP-AWS, you need the following:

- A Linux system, or equivalent
- A Fortran-90 compiler that supports Fortran-2003, e.g. gfortran or ifort
- ecCodes library, a recent version supporting BUFR tables v40 (see section 2.2)
- If you are building ecCodes, cmake is required
- hdf5 library. Version 1.10.x has been used in testing.

- netCDF C and Fortran libraries
- gmake or equivalent

If you are not sure what versions of the external libraries are installed on your system, try running commands such as:

```
bufr_dump -V
h5dump -V
cmake --version
```

The following combinations of compilers have been used to test the software (i.e. AAPP, AAPP-AWS and the dependency libraries):

- gfortran, gcc, g++ version 8.1.0
- ifort, icc, icpc version 2021.2.0

2. INSTALLATION

This section explains how to install AAPP-AWS and its dependencies (ecCodes, hdf5, netCDF) assuming you do *not* have administrator privilege – i.e. the software packages are installed in local directories. If you are running as administrator, you should follow your local conventions.

AAPP-AWS is released as a part of AAPP (starting from AAPP v8.14). If you unpack the file AAPP_8.14.tgz (or current version, if later), you will find that a directory AAPP-AWS has been created. The code is built independently from the core AAPP and it uses its own makefiles. Thus the user does *not* need to first build the core AAPP. However, at least two of its dependencies (ecCodes and hdf5) are also used in the core AAPP, so you should not need to build them twice.

2.1 Download AAPP (v8.14 or higher)

After logging on to the NWP SAF web site, the AAPP releases are available on the "Software Downloads" section of the web site. You can either use a "full" release or an "update" release.

Note that, although the source code for AAPP and AAPP-AWS is distributed in the same tarfiles, you do *not* need to build the core AAPP in order to use the AWS processor.

We also recommend downloading the *install_aapp8.sh* script (available on the NWP SAF server[1]), for installing AAPP-AWS and its dependencies, although this is not required.

2.2 Download and install ecCodes, hdf5 and netCDF

If they are not already installed on your system, the simplest way to install ecCodes, hdf5 and netCDF is to run the *install_aapp8.sh* script[1], as follows:

```
./install_aapp8.sh 1   # to install hdf5
./install_aapp8.sh 3   # to install ecCodes
```

[1] https://nwp-saf.eumetsat.int/downloads/aapp_data_files/

```
./install_aapp8.sh 12  # to install netCDF-C
./install_aapp8.sh 13  # to install netCDF-F
```

Note that netCDF dependencies will be installed into the same directory as hdf5.

By default the packages will be downloaded, built and installed in the current working directory. If you would prefer to build/install the packages into a different directory, execute the script as follows:

```
TOP=<dependent packages directory> ./install_aapp8.sh <item number>
```

By default, the gfortran compiler is used, together with gcc and g++. If you wish to use Intel compilers, you can first set environment variables:

```
export FORTRAN_COMPILER=ifort
export CC=icc
export CXX=icpc
```

Important note

In order to obtain BUFR tables v40 required by AAPP-AWS, you will need ecCodes version of at least 2.31.0. In turn, this implies that you need a C++ compiler supporting the C++17 standard. In practice, if you use GNU compilers, this implies that **you need at least gcc v8.1.0**. If this is a problem then it is possible to build an earlier version of ecCodes but to copy the BUFR tables v40 from a newer version.

Note also that if you use gcc v11.x then some changes may be required to compiler flags – see OPS-LRS bug fixes https://nwp-saf.eumetsat.int/site/software/aapp/updates/

2.3 Install AAPP-AWS

The AAPP-AWS installation process requires the following steps in order to build and install AAPP-AWS:

1) configuration of AAPP-AWS makefiles
2) `make`
3) `make install` (optionally)

Note that this process *only* configures and installs AAPP-AWS, not core AAPP.

The build and install directories generated by the installation should have the contents shown below.

- The AAPP-AWS build directory structure should contain:

```
/bin     (contains executables, and aapp_aws_env.sh)
/build   (contains generated modules and libraries)
/src
    /bin   (main programs)
    /libaws
    /makefiles
```

- The AAPP-AWS install directory (if used) should contain:
  /bin (executables)
  /*aapp_aws_env.sh*

The following sections describe the recommended approach to installing AAPP-AWS from a full release tarfile (section 2.3.1) or an update release tarfile (section 2.3.2). For users who prefer to carry out the configuration and build steps manually, these steps are described in detail in section 2.3.3.

2.3.1 Install AAPP-AWS from a Full Release

The simplest way to install AAPP-AWS from an AAPP "full release" tarfile is to run the *install_aapp8.sh* script[1] as follows:

```
export TOP=<dependent packages directory (as in section 2.2)>
mv <AAPP tarfile> ${TOP}/tarfiles
./install_aapp8.sh 14  # to install (configure and build) AAPP-AWS
```

By default, this assumes that install_aapp8.sh script has already been run to install the package dependencies; AAPP-AWS will be installed into the same location: `${TOP}/install`. To source dependencies installed elsewhere, modify the value of `<LIB>_INSTALL_DIR` at the top of the script. Remember to use a Fortran compiler for AAPP-AWS that is compatible with the C/C++ compiler you used for hdf5, netCDF and ecCodes (e.g. gfortran 8.x). See comments in Section 2.2 about use of the environment variables FORTRAN_COMPILER, CC and CXX.

2.3.2 Install AAPP-AWS from an Update Release

If using the v8.14 update or later release, unpack the AAPP "update release" tar file, and install the code as follows:

```
# Unpack the AAPP update tar file
cd <AAPP build directory (containing AAPP, metop-tools etc)>
tar -xvmzf <path to update release tarfile>

# Configure AAPP-AWS
# Note: only needed if installing AAPP-AWS at v8.14 or for the first time
# See section 2.3.3 for details
cd AAPP-AWS

./configure_aapp_aws.sh <options - see section 2.3.3>

# Build AAPP-AWS
cd AAPP-AWS/src
make (OR make clean)
make install (optional)
```

Note that this step only installs AAPP-AWS; to update AAPP (as usual) run the `make` command from the AAPP build directory.

2.3.3 Configure and build AAPP-AWS – manual approach

For users who prefer to build the software manually, detailed instructions are provided in this section.

Configure AAPP-AWS:

Unpack AAPP in the usual way, as described in the release note. From AAPP v8.14 onwards, you will find that there is a new top-level directory ***AAPP-AWS***. Navigate to the `AAPP-AWS` directory, then run the *configure_aapp_aws.sh* script. By default it uses bash shell, but will work equally well under ksh if you change the first line.

The *configure_aapp_aws.sh* script can be executed in several ways depending on where the libraries (ecCodes, netCDF and hdf5) have been installed, as described below in (i)-(iii). Be careful if your system already has environment variables defined for `ECCODES_LIB` and/or `HDF5_LIB` and/or `NETCDF_LIB` that are unrelated to AAPP-AWS. In this case either *unset* them or follow the instructions in (iii). See Table 1 for a description of the input variables to the *configure_aapp_aws.sh* script.

(i) If you have installed ecCodes, hdf5 and netCDF yourself (as in 2.2), define and export the environment variables and run the *configure_aapp_aws.sh* as follows:

```
export HDF5_INSTALL_DIR=<path to hdf5 installation>
export NETCDF_INSTALL_DIR=<path to netcdf installation> # optional
export ECCODES_INSTALL_DIR=<path to eccodes installation>
export FC=<fortran compiler>    # optional (defaults to gfortran)
./configure_aapp_aws.sh
```

Note that, if `NETCDF_INSTALL_DIR` is not explicitly defined, it is assumed to be the same directory as `HDF5_INSTALL_DIR`.

If all dependencies are installed in one location, you can alternatively export the environment variable `AAPP_AWS_DEPENDENCIES` to search for and use the libraries within this directory, i.e.

```
export AAPP_AWS_DEPENDENCIES=<path to all package installations>
export FC=<fortran compiler>    # optional (defaults to gfortran)
./configure_aapp_aws.sh
```

(ii) If ecCodes, hdf5 and netCDF are centrally installed, the corresponding `<package>_INSTALL_DIR` environment variables do not need to be set, as the script will try to determine the library locations:

```
./configure_aapp_aws.sh --fortran-compiler=$FC
```

(iii) Alternatively, you can execute *configure_aapp_aws.sh* specifying the various directories explicitly as arguments or environment variables. See Table 1.

Table 1: User inputs to the script *configure_aapp_aws.sh*

Script argument	Equivalent environment variable	Purpose
--fortran-compiler=	FC	Compiler: defaults to gfortran; alternative option is ifort.

-	AAPP_AWS_DEPENDENCIES	Directory contains “lib” (or “lib64”) and “include” subdirectories for ecCodes, hdf5 and netcdf.
-	ECCODES_INSTALL_DIR	Directory contains “lib” (or “lib64”) and “include” subdirectories.
-	HDF5_INSTALL_DIR	Directory contains “lib” (or “lib64”) and “include” subdirectories.
-	NETCDF_INSTALL_DIR	Directory contains “lib” (or “lib64”) and “include” subdirectories.
--eccodes-lib=	ECCODES_LIB	Location of the ecCodes library files, e.g. *libeccodes_f90.so*
--hdf5-lib=	HDF5_LIB	Location of the hdf5 library files e.g. *libhdf.so*
--netcdf-lib=	NETCDF_LIB	Location of the netCDF library files e.g. *libnetcdff.so*
--eccodes-inc=	ECCODES_INC	Location of the ecCodes include files, e.g. *eccodes.mod*
--hdf5-inc=	HDF5_INC	Location of the hdf5 include files, e.g. *hdf5.mod*
--netcdf-inc=	NETCDF_INC	Location of the netcdf include files, e.g. netcdf.*mod*
--prefix=		Installation directory (optional)

User inputs are prioritised as follows:
- script arguments **always** take priority over environment variables
- `AAPP_AWS_DEPENDENCIES` takes priority over other environment variables (provided libraries are found in this location)
- `HDF5_LIB` / `HDF5_INC` environment variables take priority over `HDF5_INSTALL_DIR` (similarly for netcdf and ecCodes).

Optional arguments:
Note that it is not necessary to specify the location of the “include” directory if it is on the same level as the “lib” directory (e.g. `/usr/include` … `/usr/lib`) or if it is a subdirectory of “lib”. Additionally, the use of `--netcdf-lib` and `--netcdf-inc` flags is optional (default to the same locations as `--hdf5-lib` and `--hdf5-inc`). Similarly, `NETCDF_INSTALL_DIR` defaults to the same directory as `HDF5_INSTALL_DIR`.

You may find that the hdf5 libraries are centrally installed on your system but have not been built with Fortran enabled (*libhdf5_fortran.so* missing), or are a rather old version (v1.8.x) and have not been built with Fortran2003 support. In this case the script will warn you, and you should follow the instructions in 2.2 to create a new library.

The *configure_aapp_aws.sh* script firstly runs some test programs to verify whether the external libraries are OK. Secondly it creates the necessary directories used in the build process. Thirdly, it creates a file called *Makefile.ARCH* which is used by all the lower-level makefiles. Finally, it creates a file *aapp_aws_env.sh* which is placed in the “bin” directory and should be sourced at run time (this file sets up `PATH` and, if necessary, `LD_LIBRARY_PATH`).

If you prefer, you can create your own version of *Makefile.ARCH*, taking *Makefile.ARCH.in* as a template.

Build AAPP-AWS:

In the *src* directory, just type `make` to compile the software.

If you have specified a *prefix* in the configure step, you can then run `make install` .

Use `make clean` if you want to re-build after changing the source code or packages in the dependency libraries.

3. RUNNING THE SOFTWARE

3.1 General comments

Two main programs are currently provided: *aws_netcdf_to_bufr_1b.exe* and *aws_netcdf_to_bufr_1c.exe*. Both take level 1B as input. The difference is that *aws_netcdf_to_bufr_1c.exe* includes onward processing to level 1C.

The input can be either netCDF format or BUFR. For BUFR input, the sequence must be 3-10-078, which is the template designed for TROPICS data. The software assumes that input files with a file name suffix ".bin", ".bufr" or ".bfr" are BUFR.

The normal combinations of input and output are shown in Table 2.

Table 2: Input and output combinations for AAPP-AWS

Program	Input	Output
aws_netcdf_to_bufr_1b.exe	netCDF	BUFR (1b)
aws_netcdf_to_bufr_1c.exe	netCDF	BUFR (1c) with optional inclusion of mapped BTs in the netCDF
aws_netcdf_to_bufr_1c.exe	BUFR (1b)	BUFR (1c)

It is recommended to source the environment file `aapp_aws_env.sh` before running any main program, in order to set up your `PATH` and/or `LD_LIBRARY_PATH`:

```
   source ${AAPP_AWS_HOME}/aapp_aws_env.sh
```
or
```
   . ${AAPP_AWS_HOME}/aapp_aws_env.sh
```

where `${AAPP_AWS_HOME}` is the full path of the directory containing the executables (the *bin* directory).

3.2 aws_netcdf_to_bufr_1b.exe

aws_netcdf_to_bufr_1b.exe is run as follows:

```
aws_netcdf_to_bufr_1b.exe -i infile -o outfile_bufr [-n bufr_namelist]
```

The optional namelist file allows you to control details of the BUFR encoding, e.g.

```
&aws_bufr
  originating_centre = 255
  sub_centre = 0
  master_table = 40
  local_table = 0
  local_subtype = 0
  scans_per_message = 4
  spot_thin = 1
  scan_thin = 1
/
```

Note that the spot_thin and scan_thin variables would normally remain at 1, in order to preserve the level 1B sampling

3.3 aws_netcdf_to_bufr_1c.exe

aws_netcdf_to_bufr_1c.exe is run as follows:

```
aws_netcdf_to_bufr_1c.exe [-i infile] [-io inoutfile] [-o outfile_bufr] [-n
bufr_namelist] [-b band] [-a averaging_namelist]
```

Compared with the 1b program, here we have the additional option of writing the mapped brightness temperatures back to the original netCDF file, in a new group “mapped”. The BUFR output contains only one set of geolocations – those of the chosen reference band (provided by the “-b” option). The default reference band is band 1 (50-55 GHz).

This executable also offers the capability to average adjacent fields of view for specified channels in order to reduce instrument noise, via an “averaging” namelist, specified by the “-a “ option. For example, to apply 3x3 averaging to the eight 50 GHz channels (channels 1 to 8), you create a namelist file (e.g. aws_average.nl) containing:

```
&AWS_averaging
  nchan_to_modify = 8
  channel = 1 2 3 4 5 6 7 8
  nxaverage = 3 3 3 3 3 3 3 3
  nyaverage = 3 3 3 3 3 3 3 3
/
```

The averaging software will only use spots that have brightness temperature greater than 0 and less than 350K. The *nxaverage* and *nyaverage* values should be odd (i.e. 1, 3, 5, etc.).

Also, you may wish to specify *spot_thin* and *scan_thin* in the BUFR namelist in order to match the output to the resolution of your NWP system.

4. BUFR SEQUENCE

The same BUFR sequence is used for level 1B and level 1C – then difference being that at level 1B we have 4 geolocation groups, with different numbers of channels in each group (8, 1, 6, 4 respectively) and at level 1C there is only one geolocation group, containing all channels.

Channels are numbered 1 through 19 in the BUFR output (not 11 to 44 as in [RD-1]).

```
! The following BUFR sequence for Tropics is used (310078):
!
!  001007         SATELLITE IDENTIFIER
!  001016         SATELLITE SUB IDENTIFIER
!  002019         SATELLITE INSTRUMENTS
!  002020         SATELLITE CLASSIFICATION
!  001033         IDENTIFICATION OF ORIGINATING/GENERATING CENTRE
!  001034         IDENTIFICATION OF ORIGINATING/GENERATING SUB-CENTRE
!  301011         YEAR, MONTH, DAY
!  301013         HOUR,MINUTE, SECOND
!  005040         ORBIT NUMBER
!  201132         CHANGE DATA WIDTH
!  005041         SCAN LINE NUMBER
!  201000         CHANGE DATA WIDTH         CANCEL
!  005043         FIELD OF VIEW NUMBER
!  033079         GRANULE LEVEL QUALITY FLAGS
!  033080         SCAN LEVEL QUALITY FLAG
!  033078         GEOLOCATION QUALITY
!  007002         HEIGHT OR ALTITUDE
!  102003         REPLICATE 2 DESCRIPTORS 3 TIMES
!  008097         METHOD USED TO CALCULATE THE AVERAGE INSTRUMENT TEMPERATURE
!  012164         INSTRUMENT TEMPERATURE
!  117000         DELAYED REPLICATION OF 17 DESCRIPTORS
!  031001         DELAYED DESCRIPTOR REPLICATION FACTOR
!  005001         LATITUDE (HIGH ACCURACY)
!  006001         LONGITUDE (HIGH ACCURACY)
!  007024         SATELLITE ZENITH ANGLE
!  005021         BEARING OR AZIMUTH
!  007025         SOLAR ZENITH ANGLE
!  005022         SOLAR AZIMUTH
!  109000         DELAYED REPLICATION OF 9 DESCRIPTORS
!  031001         DELAYED REPLICATION FACTOR
!  005042         CHANNEL NUMBER
!  002153         SATELLITE CHANNEL CENTRE FREQUENCY
!  002154         SATELLITE CHANNEL BAND WIDTH
!  002104         ANTENNA POLARISATION (all channels are QV according to OSCAR)
!  012066         ANTENNA TEMPERATURE
!  012163         BRIGHTNESS TEMPERATURE
!  012158         NOISE-EQUIVALENT DELTA TEMPERATURE WHILE VIEWING COLD TARGET
!  012159         NOISE-EQUIVALENT DELTA TEMPERATURE WHILE VIEWING WARM TARGET
!  033094         CALIBRATION QUALITY CONTROL FLAGS
```

The quality flags are defined in Table 3. We have tried to follow as closely as possible the official WMO definitions, though there are a few deviations.

Table 3: Quality flags used in the AWS processor. In the BUFR bit number, bit 1 is the most significant bit.

Descriptor	Description	BUFR bit number	Numeric value	Meaning	netCDF origin
033079	Granule level quality flags (16-bit flag table)	6	1024	Missing input products	L1B_quality_flag bit 0
		7	512	Data gaps	L1B_quality_flag bit 1
		8	256	Corrupted input products	L1B_quality_flag bit 2
		9	128	Instrument anomaly	L1B_quality_flag bit 3
		10	64	Missing or degraded aux data	L1B_quality_flag bit 4
		11	32	Degraded due to manoeuvre	L1B_quality_flag bit 5
033080	Scan level quality flags (20 bit flag table)	6	16384	Position flag earth view	aws_position_flag_earthview
		9	2048	Instrument temperature quality flag	aws_instrument_temperature_quality_flag
		12	256	PRT processing flag	prt_processing_flag
		18	4	Position flag cold view	aws_position_flag_coldview
		19	2	Position flag warm view	aws_position_flag_warmview
033094	Calibration quality control flags (24 bit flag table)	17	128	Solar or lunar intrusion	aws_sun_contamination_flag aws_moon_contamination_flag
		19	32	Cold calibration consistency	aws_position_flag_coldview
		20	16	Warm calibration consistency	aws_position_flag_warmview
033078	Geolocation quality (4 bit code table)	-	-	Not currently used	-