# NWP SAF

Satellite Application Facility for Numerical Weather Prediction

Document NWPSAF-KN-UD-005
Version 1.0.16
December 2008

# AWDP User Manual and Reference Guide

*Anton Verhoef, Jur Vogelzang, Jeroen Verspeek and Ad Stoffelen*

***KNMI, De Bilt, the Netherlands***

# AWDP User Manual and Reference Guide

## KNMI, De Bilt, the Netherlands

This documentation was developed within the context of the EUMETSAT Satellite Application Facility on Numerical Weather Prediction (NWP SAF), under the Cooperation Agreement dated 16 December, 2003, between EUMETSAT and the Met Office, UK, by one or more partners within the NWP SAF. The partners in the NWP SAF are the Met Office, ECMWF, KNMI and Météo France.



| **Change record** | | | |
|---|---|---|---|
| **Version** | **Date** | **Author / changed by** | **Remarks** |
| 1.0j | Jun 2007 | Anton Verhoef | First draft |
| 1.0k | Oct 2007 | Anton Verhoef | Adapted for AWDP version 1.0k |
| 1.0.13 | Mar 2008 | Anton Verhoef | Adapted for AWDP version 1.0.13 |
| 1.0.14 | Oct 2008 | Anton Verhoef | First version for external review |
| 1.0.16 | Dec 2008 | Anton Verhoef | Modified according to DRI comments |

# Contents

# Preface

## Preface to version 1.0j

Software code for processing satellite data may become very complex. On the one hand, it consists of code related to the technical details of the satellite and instruments; on the other hand, the code drives complex algorithms to create the physical end products. Therefore, the EUMETSAT Satellite Application Facility (SAF) project for Numerical Weather Prediction (NWP) has included some explicit activities aiming at enhancing the modularity, readability and portability of the processing code.

For several years, the KNMI observation research group has been developing processing code to supply Near Real Time (NRT) level 2 surface wind products based on the ERS and SeaWinds Scatterometer level 1b Normalized Radar Cross Section data ($\sigma_0$). This work is coordinated and supervised by Ad Stoffelen. In the beginning only an adaptation of his ERS code existed. Later Marcos Portabella and Julia Figa added modifications and extensions to improve, e.g., the wind retrieval and quality control algorithms. In 2003, John de Vries finished the first official release of a processor within the NWP SAF. This processor was called the QuikSCAT Data Processor (QDP).

Meanwhile, Jos de Kloe has been updating the code for ERS scatterometer wind processing. For many parts of the process steps (e.g., the BUFR handling and part of the wind retrieval) a large overlap with SeaWinds Data processing coding exists. The KNMI Scatterometer Team is working towards generic NRT scatterometer processing. As a result, a new modular processing code for SeaWinds data was developed within the NWP SAF: the SeaWinds Data Processor (SDP) as successor of QDP.

Based on the generic code already available for SeaWinds and ERS processing, a new ASCAT Wind Data Processor (AWDP) was developed. This document is the corresponding reference manual. I hope this manual will strongly contribute to the comprehension of future developers and of users interested in the details of the processing.

Many persons contributed (directly or indirectly) to the development of the scatterometer software at KNMI: Hans Bonekamp, Jos de Kloe, Marcos Portabella, Ad Stoffelen, Anton Verhoef, Jeroen Verspeek, Jur Vogelzang and John de Vries are (in alphabetical order) the most important contributors.

Anton Verhoef, June 2007

# Chapter 1

# Introduction

## 1.1 Aims and scope

The ASCAT Wind Data Processor (AWDP) is a software package written in Fortran 90 for handling data from the Advanced Scatterometer (ASCAT) and European Remote Sensing satellite (ERS) scatterometer instruments. Details of these instruments can be found on several web sites and in several other documents, see e.g. [*Portabella*, 2002; *Stoffelen*, 1998] and information on the ESA and EUMETSAT web sites.

AWDP generates surface winds based on ASCAT and ERS data. It allows performing the ambiguity removal with the Two-dimensional Variational Ambiguity Removal (2DVAR) method and it supports the Multiple Solution Scheme (MSS). The output of AWDP consists of wind vectors which represent surface winds within the ground swath of the scatterometer. Input of AWDP is Normalized Radar Cross Section (NRCS, $\sigma_0$) data. These data may be real-time. The input files of AWDP are in BUFR or Product Format Specification (PFS, native MetOp) format. BUFR input may be provided using the BUFR templates for ERS or ASCAT; output is always written using the ASCAT BUFR template. Moreover, AWDP needs Numerical Weather Prediction (NWP) model winds as a first guess for the Ambiguity Removal step. These data need to be provided in GRIB.

## 1.2 Development of AWDP

AWDP is developed within the Numerical Weather Prediction Satellite Application Facility (NWP SAF) and Ocean and Sea Ice Satellite Application Facility (OSI SAF) programs as code which can be run in an operational setting. The coding is in Fortran 90 and has followed the procedures specified for the NWP SAF. Special attention has been paid to robustness and readability. AWDP may be run on every modern Unix or Linux machine. In principle, AWDP can also be run on a Windows machine if a Unix emulator like Cygwin is installed.

## 1.3 Testing AWDP

Modules are tested by test programs and test routines. Many test routines or test support routines are part of the modules themselves. Test programs can be compiled separately. For the AWDP program, the description of the test programs and the results of the testing are reported in [*Verhoef et. al.*, 2008].

## 1.4 User Manual and Reference Guide

This document is intended as the complete reference book for AWDP.

Chapter 2 is the user manual (UM) for the AWDP program. This chapter provides the basic information for installing, compiling, and running AWDP. Chapter 3 contains the Product Specification (PS) of the AWDP program. Reading the UM and the PS should provide sufficient information to the user who wants to apply the AWDP program as a black box.

The subsequent chapters are of interest to developers and users who need more specific information on how the processing is done. The Top Level Design (TLD) of the code and the Module Design (MD) of the AWDP code can be found in Chapter 4. Several modules are very generic for NRT scatterometer data processing. Examples are the modules for the BUFR and GRIB handling, ambiguity removal, and parts of the wind retrieval. These generic modules are part of the generic scatterometer (genscat) layer and are described in Chapter 5 to Chapter 9.

The appendices of this document contain a complete calling tree of the AWDP program up to and including the genscat layer. The appendices also contain a list of ASCAT BUFR data descriptors and a list of acronyms.

## 1.5 Conventions

Names of physical quantities (e.g., wind speed components *u* and *v*), modules (e.g. *BufrMod*), subroutines and identifiers are printed italic.

Names of directories and subdirectories (e.g. `awdp/src`), files (e.g. `awdp.F90`), and commands (e.g. `awdp -f input`) are printed in Courier. Software systems in general are addressed using the normal font (e.g. AWDP, genscat).

Hyperlinks are printed in blue and underlined (e.g. http://www.knmi.nl/scatterometer/).

References are in square brackets with the name of the author italic (e.g. [*Stoffelen*, 1998]).

# Chapter 2

# AWDP User Manual

This chapter is the user manual of the AWDP program. Sections 2.1 and 2.2 give general information about AWDP. Section 2.3 provides information on how to install, compile, and link the AWDP software. The command line arguments of AWDP are discussed in section 2.4. Section 2.5 gives information on a script for running AWDP.

Please note that any questions or problems regarding the installation or use of AWDP can be addressed at the NWP SAF helpdesk at http://www.nwpsaf.org/.

## 2.1 Why using the AWDP program?

Scatterometers provide valuable observational data over the world's oceans. Therefore, successful assimilation of scatterometer data in numerical weather prediction systems generally improves weather forecasts. The AWDP program has been developed to fully exploit scatterometer data. It is meant to form the key component of the observation operator for surface winds in data assimilation systems.

The general scheme of AWDP (and any other wind scatterometer data processor) is given in figure 2.1. The input of the AWDP program is the EUMETSAT ASCAT level 1b BUFR or PFS wind product or the ESA ERS level 2 BUFR wind product. Besides this, GRIB input containing land-sea mask, sea surface temperature or soil temperature on level 0, and first guess winds over the globe is necessary.

The AWDP processing chain contains several steps (see figure 2.1):

1. Pre-processing. The input file is decoded and the radar backscatter ($\sigma_0$) values are written in the data structures of AWDP. Some quality control on the input data is done.
2. Collocation with NWP data. The GRIB files containing NWP data are read and the values for land fraction, sea surface temperature and first guess winds are interpolated and stored with the information of each WVC.
3. Inversion. The $\sigma_0$ values are compared to the Geophysical Model Function (GMF) by means of a Maximum Likelihood Estimator (MLE). The wind vectors that give the best description of the $\sigma_0$ values (the solutions) are retained. The MLE is also used to assign a probability to each wind vector. The normal scheme allows 2 solutions at most, but in the Multiple Solution Scheme (MSS) the maximum number of solutions is 144.

4. Quality Control. Solutions that lie far away from the GMF are likely to be contaminated by, e.g., sea ice or confused sea state. During Quality Control these solutions are identified and flagged.

5. Ambiguity Removal. This procedure identifies the most probable solution using some form of external information. AWDP uses a two-dimensional variational scheme (2DVAR) as default. A cost function is minimized that consists of a background wind field and all solutions with their probability, using meteorological balance, mass conservation and continuity as constraints.

6. Quality Monitoring. The last step is to output quality indicators to an ASCII monitoring file and to write the results in a BUFR format output file.

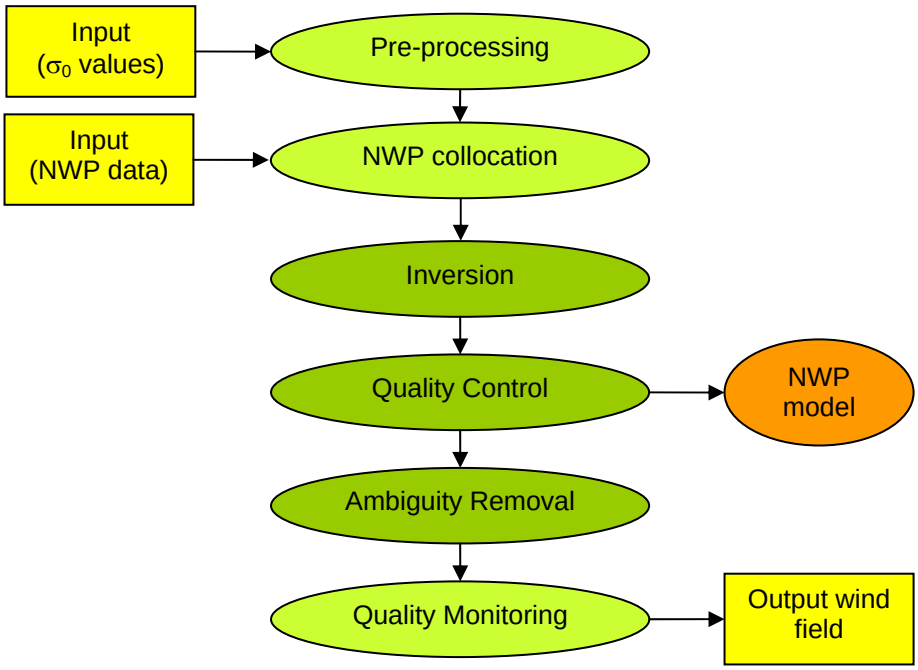


**Figure 2.1** AWDP processing scheme. The wind vectors and their probabilities after Quality Control may be fed directly in the Data Assimilation step of a Numerical Weather Prediction model.

Step 1, 2 and 6 of the processing chain are rather trivial; the real work is done in steps 3, 4, and 5. Note that an undesirable dependency arises if the output wind field is assimilated into a numerical weather prediction (NWP) model: in the 2DVAR Ambiguity Removal step (i) a background wind field is used and (ii) meteorological balance constraints causing spatially correlated error. Therefore it is recommended to feed the wind solutions and their probabilities directly into the NWP data assimilation step after Quality Control, as indicated in figure 2.1. If this is done, the Ambiguity Removal step can be skipped and consequently, no forecast winds are necessary in the NWP input. No impact tests have been performed to date by assimilating AWDP outputs after ambiguity removal.

As further detailed in Chapter 3, AWDP profits from developments in

- Inversion and output of the full probability density function of the vector wind (Multiple Solution Scheme, MSS).

- Quality Control (QC).
- Meteorologically balanced Ambiguity Removal (2DVAR).
- Quality monitoring.
- Capability to process ASCAT data on both 25 km and 12.5 km cell spacing.

A complete specification of the AWDP program can be found in the Product Specification in Chapter 3. The program is based on generic genscat routines for inversion, ambiguity removal, and BUFR and GRIB file handling. These routines are discussed in more detail in Chapter 5 to Chapter 9.

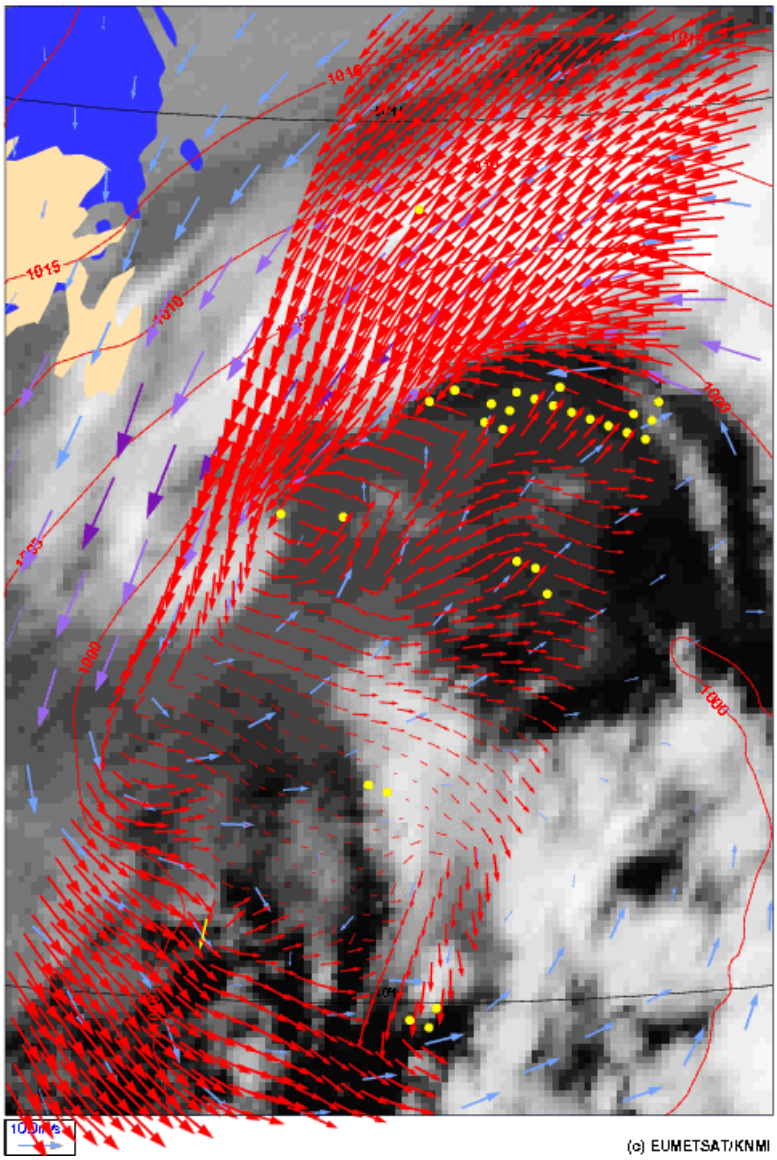


**Figure 2.2** AWDP wind field retrieved for 15 April 2007, approximately 13 h UTC, at 25 km cell spacing, near Newfoundland overlaid on a GOES IR satellite image. The yellow dots are rejected WVCs. The blue and violet arrows are a 6 hours forecast from the KNMI HIRLAM model.

## 2.2 Modes of using AWDP

There are several modes to assimilate the ASCAT data in NWP models using AWDP. Anyway, the first thing to assure oneself of is the absence of biases by making scatter plots between ASCAT and NWP model first guess for at least wind speed, but wind direction and wind components would also be of interest to guarantee consistency.

The operational ASCAT wind product, available as a deliverable from the EUMETSAT OSI SAF project, could be the starting point for NWP assimilation:

1. The unique solution at every WVC may be assimilated as if it were buoy data. This is the fastest way and one exploits the data to a large extent. For a small advantage, AWDP could be installed to provide 2DVAR solutions based on the local first guess.
2. The AWDP software may be used to modify the 3DVAR or 4DVAR data assimilation system to work with the ambiguous wind solutions and their probabilities at every WVC. This represents some investment, but is applicable for all scatterometer data. The advantage with respect to option 1 in the ambiguity removal occurs only occasionally, but always in dynamic atmospheric cases (storms or cyclones) that are really relevant.

Both options can be based on AWDP in standard or MSS mode, and at various resolutions. MSS is somewhat more dependent on the first guess and balance constraints in 2DVAR than the standard AWDP, but much less noisy. A noticeable advantage is thus obtained by using option 2 and potentially the full hi-res benefit of the ASCAT data is achieved. At the moment, the 12.5-km data are experimental, since at KNMI we are now objectively evaluating the added value of MSS and 2DVAR at 12.5 km. The mode of using AWDP thus depends on the opportunities, experience, and time the user has to experiment with ASCAT data in the NWP system under consideration.

The AWDP program can, of course, also be used to create a stand-alone wind product, e.g., for nowcasting purposes. Such a stand-alone ASCAT wind product is a deliverable of the OSI SAF project. More information on this project can be found on http://www.osi-saf.org/.

## 2.3 Installing AWDP

AWDP is written in Fortran 90 (with a few low level modules in C) and is designed to run on a modern computer system under Linux or Unix. AWDP needs a Fortran 90 compiler and a C compiler for installation. AWDP comes along with a complete make system for compilation. In some cases, the Makefiles call installation scripts which are written in Bourne shell to enhance portability. When compiled, AWDP requires about 100-150 Mb disk space.

In principle, AWDP may also run under Windows. However, it needs the BUFR and GRIB libraries from ECMWF, and this poses some restrictions on the systems supported. Under Windows one must use a (free) Unix emulator like Cygwin (see http://www.cygwin.com/ for more information and download, and section 2.3.6 for some directions).

To install AWDP, the following steps must be taken:

1. Copy the AWDP package (file `AWDP<version>.tar.gz`) to the directory from which AWDP will be applied, and unzip and untar it. This will create subdirectories `awdp` and `genscat` that contain all code needed (see section 2.3.1), and a script called `InstallAWDP`

for easy compilation.

2. Download the ECMWF BUFR library file `bufr_000360.tar.gz` (or another version not earlier than 000240) and copy it to directory `genscat/support/bufr`. See also section 2.3.3.
3. Download the ECMWF GRIB library file `gribex_000360.tar.gz` (or another version not earlier than 000220) and copy it to directory `genscat/support/grib`. See also section 2.3.4.
4. Go to the top directory and run the `./InstallAWDP` script. The script will ask for the compiler used and it will invoke the make system for compilation and linking of the software (see also section 2.3.5).

AWDP is now ready for use, provided that the environment variables discussed in section 2.3.2 have the proper settings. See also sections 2.4 and 2.5 for directions on how to run AWDP.

### 2.3.1 Directories and files

All code for AWDP is stored in a file named `AWDP<version>.tar.gz` that is made available in the framework of the NWP SAF project. This file should be placed in the directory from which AWDP is to be run. After unzipping (with `gunzip AWDP<version>.tar.gz`) and untarring (with `tar -xf AWDP<version>.tar`), the AWDP package is extracted in subdirectories `awdp` and `genscat`, which are located in the directory where the tar file was located. Subdirectories `awdp` and `genscat` each contain a number of files and subdirectories. A copy of the release notes can also be found in the directory `awdp/docs`.

Tables 2.1 and 2.2 list the contents of directories `awdp` and `genscat`, respectively, together with the main contents of the various parts. Depending on the distribution, more directories may be present, but these are of less importance to the user.

| Name | Contents |
|---|---|
| `doc` | Documentation, including this document |
| `execs` | Link to `awdp` executable, shell script for running AWDP |
| `src` | Source code for AWDP program and supporting routines |
| `test` | Example BUFR and GRIB input files for testing purposes. |

**Table 2.1** Contents of directory `awdp`.

| Name | Contents |
|---|---|
| `ambrem` | Ambiguity removal routines |
| `ambrem/twodvar` | KNMI 2DVAR ambiguity removal routines |
| `inversion` | Inversion and quality control routines |
| `main` | Dummy subdirectory to facilitate the make system |
| `support` | General purpose routines sorted in subdirectories |
| `support/BFGS` | Minimization routines needed in 2DVAR |
| `support/bufr` | BUFR tables (in subdirectory) and file handling routines |
| `support/Compiler_Features` | Compiler specific routines, mainly command line handling |
| `support/convert` | Conversion between wind speed/direction and *u* and *v* |
| `support/datetime` | Date and time conversion routines |

| Name | Contents |
|---|---|
| support/ErrorHandler | Error handling routines |
| support/file | File handling routines |
| support/grib | GRIB file handling routines |
| support/num | Numerical definitions and number handling routines |
| support/pfs | PFS file handling routines |
| support/singletonfft | FFT routines needed in minimization |
| support/sort | Sorting routines |

**Table 2.2** Contents of directory genscat.

Directories awdp and genscat and their subdirectories contain various file types:

- Fortran 90 source code, recognizable by the .F90 extension;
- C source code, recognizable by the .c extension;
- Files and scripts that are part of the make system for compilation like Makefile_thisdir, Makefile, use_, Objects.txt and Set_Makeoptions (see 2.3.4 for more details);
- Scripts for the execution of AWDP in directory awdp/execs;
- Look-up tables and BUFR tables needed by AWDP;
- Files with information like Readme.txt.

After compilation, the subdirectories with the source code will also contain the object codes of the various modules and routines.

### 2.3.2 Environment variables

AWDP needs a number of environment variables to be set. These are listed in table 2.3 together with their possible values.

| Name | Value |
|---|---|
| $BUFR_TABLES | genscat/support/bufr/bufr_tables/ |
| $LOCAL_DEFINITION_TEMPLATES | genscat/support/grib/gribtemplates |
| $INVERSION_LUTSDIR | genscat/inversion |
| $LUT_FILENAME_C_VV | Any existing, writable path + file name |

**Table 2.3** Environment variables for AWDP.

The $BUFR_TABLES variable guides AWDP to the BUFR tables needed to read the input and write the output. The $LOCAL_DEFINITION_TEMPLATES variable is necessary for a proper functioning of the GRIB decoding software.

The variable $INVERSION_LUTSDIR should point to a directory containing some look up tables (extension .asc) that are used by the inversion software. The necessary tables are delivered with genscat.

The variable $LUT_FILENAME_C_VV points AWDP to the correct C band GMF lookup table at VV polarisation. It should contain a file name including a valid path. If the file does not exist, it will be created when the inversion is invoked for the first WVC. In order to prevent confusion, it is advised to use standard file names <path>/cmod5.dat, <path>/cmod5_5.dat,

`<path>/cmod5_n.dat`, or `<path>/cmod6.dat`, since the inversion software uses the file name to determine which CMOD version is used.

### 2.3.3 Installing the BUFR library

AWDP needs the ECMWF BUFR library for its input and output operations. Only ECMWF is allowed to distribute this software. It can be obtained free of charge from ECMWF at the web page http://www.ecmwf.int/products/data/software/bufr.html. The package contains scripts for compilation and installation. The reader is referred to this site for assistance in downloading and installing the BUFR Library.

Directory `genscat/support/bufr` contains the shell script `make.bufr.lib`. It unzips, untars, and compiles the BUFR library file which is downloaded from ECMWF and placed into this directory. This script is part of the genscat make system and it is automatically invoked when compiling genscat. The current version is tested with BUFR version 000360, but later versions (or earlier, but not earlier than 000240) can be used. However, AWDP is not tested with later versions.

BUFR file handling at the lowest level is difficult to achieve. Therefore some routines were coded in C. These routines are collected in library *bufrio* (see also section 8.4). Its source code is located in file `bufrio.c` in subdirectory `genscat/support/bufr`. Compilation is done within the genscat make system and requires no further action from the user (see 2.3.5).

### 2.3.4 Installing the GRIB library

AWDP needs the ECMWF GRIB library for its input operations. Only ECMWF is allowed to distribute this software. It can be obtained free of charge from ECMWF at the web page http://www.ecmwf.int/products/data/software/grib.html. The package contains scripts for compilation and installation. The reader is referred to this site for assistance in downloading and installing the GRIB Library.

Directory `genscat/support/grib` contains the shell script `make.grib.lib`. It unzips, untars, and compiles the GRIB library file which is downloaded from ECMWF and placed into this directory. This script is part of the genscat make system and it is automatically invoked when compiling genscat. The current version is tested with GRIB version 000360, but later versions (or earlier, but not earlier than 000220) can be used. However, AWDP is not tested with later versions.

GRIB file handling at the lowest level is difficult to achieve. Therefore some routines were coded in C. These routines are collected in library *gribio* (see also section 9.4). Its source code is located in file `gribio.c` in subdirectory `genscat/support/grib`. Compilation is done within the genscat make system and requires no further action from the user (see 2.3.5).

### 2.3.5 Compilation and linking

Compilation and linking of AWDP under Linux or Unix is done in three steps:

1. Set the compiler environment variables according to the choice entered on request. This can be done by running the appropriate `use_*` scripts in directory `genscat`.
2. Go to directory `genscat` and type `make`.
3. Go to directory `awdp` and type `make` to produce the executable `awdp` in directory

`awdp/src`.

Before activating the make system, some environment variables identifying the compiler should be set. These variables are listed in table 2.4. The environment variables in table 2.4 can be set by using one of the `use_*` scripts located in directory `genscat.` Table 2.5 shows the properties of these scripts. The scripts are available in Bourne shell (extension `.bsh`) and in C shell (extension `.csh`). Note that if one of the environment variables is not set, the default `f90` and `cc` commands on the Unix platform will be invoked. Note that in the top directory a script called `InstallAWDP` is provided that asks the user which compiler he wants to use and invokes the appropriate `use_*` script (step 1 above), after which the compilation in the `genscat` and `awdp` directories is performed (steps 2 and 3 above).

| Variable | Function |
|---|---|
| `$GENSCAT_F77` | Reference to Fortran 77 compiler |
| `$GENSCAT_F90` | Reference to Fortran 90 compiler |
| `$GENSCAT_CC` | Reference to C compiler |
| `$GENSCAT_LINK` | Reference to linker for Fortran objects |
| `$GENSCAT_CLINK` | Reference to linker for C objects |
| `$GENSCAT_SHLINK` | Reference to linker for shared objects |

**Table 2.4** Environment variables for compilation and linking.

| Script | Fortran compiler | C compiler | Remarks |
|---|---|---|---|
| `use_g95` | `g95` | `gcc` | GNU compilers by A. Vaught |
| `use_gfortran` | `gfortran` | `gcc` | GNU-GCC 4.0 compiler collection |
| `use_ifort` | `ifort` | `icc` | Intel Fortran and C compilers |
| `use_pgf90` | `pgf90` | `gcc` | Portland Fortran compiler |

**Table 2.5** Properties of the use_* scripts.

Example: To select the GNU g95 compiler under Bourne, Bash or Korn shell type "`. use_g95.bsh`", the dot being absolutely necessary in order to apply the compiler selection to the current shell. Under C shell the equivalent command reads "`source use_g95.csh`".

If the user wants to use a Fortran or C compiler not included in table 2.6, he can make his own version of the `use_*` script, or set the environment variables for compilation and linking manually.

AWDP is delivered with a complete make system for compilation and linking under Unix or Linux. The make system is designed as portable as possible, and system dependent features are avoided. As a consequence, some tasks must be transferred to shell scripts. The make system consists of two parts: one for AWDP and one for genscat. The genscat part should be run first. For compilation and linking of the genscat part, the user should move to the `genscat` directory and simply enter `make`.

The `Makefile` refers to each subdirectory of genscat, invoking execution of the local `Makefile` and, in cases where a subdirectory contains code as well as a subdirectory containing code, `Makefile_thisdir`. The Makefiles need supplementary information from the files

`Objects.txt` which are present in each directory containing code. The settings for the compilers are located in file `Makeoptions` in directory `genscat`. This file is generated by the Bourne shell script `Set_Makeoptions` which is called automatically by the genscat make system. The local `Makefile` in subdirectory `genscat/support/bufr` calls the script `make.bufr.lib` for compilation of the BUFR library (see 2.3.3). It also contains the Fortran program `test_modules` that generates the binary BUFR tables B and D from the ASCII tables already present, and is executed automatically by the make system. Program `test_modules` can also be used to test the genscat BUFR module. The `Makefile` in subdirectory `genscat/support/bufr/bufr_tables` calls some shell scripts, which make symbolic links (using the `ln -s` command) of the generic binary BUFR tables B and D under different names. There are four different naming conventions in BUFR version 000240 to 000280, and binary files are generated for each of them. Symbolic links are not guaranteed to work on each platform (e.g. by some versions of Cygwin under Windows XP), so in some cases it may be necessary to replace the `ln -s` by `cp` (copy). Further information on the make system is given in the inline comments in the scripts and Makefiles.

Compilation and linking of the AWDP part is done in a similar manner: go to the `awdp` directory and enter `make`. As with genscat, the make system will execute Makefiles in every subdirectory of `awdp`. The result is the executable `awdp` in directory `awdp/src` and a symbolic link to this executable in `awdp/execs`. AWDP is now ready for use. The make system of AWDP doesn't need any further files except the genscat file `Makeoptions`. This is the reason why genscat should be compiled first.

When recompiling (part of) AWDP or genscat with the make system, for instance when installing a new version of the BUFR library, one should be sure to enter `make clean` first. To recompile part of the software invoke the make system where needed. The compiler settings from file `Makeoptions` in directory `genscat` will be used again. If a change in this settings is necessary, type `make clean` in the genscat directory and `Makeoptions` will be removed. Don't forget to rerun the `use_*` commands to select the right compiler.

### 2.3.6 Some remarks for Cygwin users

AWDP can be used under Cygwin, a Unix emulator running under Windows. Installing and running AWDP under Cygwin is almost the same as under Unix or Linux, but the following points may be helpful for Cygwin users.

- The GNU g95 compiler comes standard with Cygwin (version 1.5.25-11 and later) so it is always possible to install AWDP using g95.
- Cygwin has its own path naming convention, for example: `C:/awdp` under Windows becomes `/cygdrive/c/awdp` under Cygwin.
- Don't forget to run the `dos2unix` command on scripts edited under Windows, otherwise Cygwin won't recognize the file as a script!

## 2.4 Command line options

The AWDP program is started from directory `awdp/execs` with the command

**`awdp [options/modes] -f <BUFR/PFS file> [-nwpfl <file>]`**

with <> indicating obligatory input, and `[]` indicating non-obligatory input. The following command line options are available.

**`-f <input file>`** Process a BUFR or PFS input file with name `input file`.
AWDP detects if the input file is in BUFR format. If not, it attempts to read the input as PFS file. The BUFR input file should either have the ASCAT or the ERS format. The PFS file should contain 25 or 12.5 km level 1b data, not full resolution level 1b data.
Example: `awdp -f ascat_20070426_test_250.l1_bufr` will process this file. The results will be written to a new BUFR file, see below in this section for the output file naming convention. It is possible to concatenate multiple BUFR input files into one using the Unix `cat` command, but PFS files must be processed one by one.

**`-nwpfl <file>`** Read a list of GRIB file names in the file named `file`.
The files in the list are read and the GRIB data are used in the wind processing. The most convenient way to construct a file list is to use the Unix command `ls -1 GRIB file pattern > file`. If no GRIB data are used, only the land masking which is present in the level 1b beam information will be used. No ice screening will be performed (unless the `-icemodel` option is used). Ambiguity removal will be performed only if model winds are already present in the input BUFR file (i.e., in case of reprocessing of a level 2 file) or if the `-armeth 1strank` option is used (i.e., selection of the 1st rank wind solution). If level 2 data are reprocessed and no NWP data are read, the `qual_sigma0` flag which was set in the initial processing is evaluated and it will be used to determine if a WVC contains suitable backscatter data for wind inversion.

Several options for the processing can be invoked.

**`-szffl <file>`** Read a list of full resolution PFS file names in the file named `file`.
The files in the list are read and the full resolution PFS data (SZF) are used to replace the 25-km/12.5-km beam data. This option is intended for research activities. The beam data ($\sigma_0$ values, incidence and azimuth angles) which are read from the BUFR or PFS level 1b input file, are replaced by average values of the data from the full resolution file which are located within a certain radius (typically 20 kilometers) from the WVC location.

**`-stressparam`** Get stress-related parameters derived from GRIB files.
This option is intended for research activities. More information can be found in the Fortran code of AWDP.

**`-noinv`** Switch off inversion (default is switched on).

**-icemodel <IM>** Choose ice screening method to be used: 0 (default), 1 or 2.
The value 0 results in no ice screening, except when a GRIB file containing sea surface temperature is read. The value 1 invokes a simple (non-Bayesian) ice model which does not keep history of the water or ice state of each location. Value 2 invokes the Bayesian ice model which keeps the history of each location and uses this history to determine the sea or ice state of a WVC. The ice screening can be combined with the ice screening which is done in the GRIB collocation. In this case, the SST of the GRIB file will be used to assign a WVC as 'surely ice' when the SST is below a certain value or 'surely water' when the SST is above a certain value.

**-noamb** Switch off ambiguity removal (default is switched on).
This option is useful if the selection of the scatterometer wind solution is left to the data assimilation procedure of a Numerical Weather Prediction model. In other words: the NWP model is fed with a number of solutions and their probability, and finds the best value when comparing with other data sources.

**-nowrite** Do not produce BUFR output (default is switched on).

**-ignorel1flags** Ignore the setting of level 1b $\sigma_0$ related flags in BUFR input.
If this option is switched on, the value of the flags and quality indicators in the beam information, including the sigma0 usability and land fraction, is neglected.

**-cmod <N>** Choose CMOD version 4, 5, 55, 5n (default) or 6.
With this option, the user can choose between GMF version CMOD4, CMOD5, CMOD5.5 (CMOD5 + 0.5 m/s), CMOD5n (CMOD5 for neutral winds) or CMOD6. The GMF table is generated by the program and written to a binary file named `c-vv2.dat` in the current directory, if it does not yet exist. Alternatively, the user may specify a file name (including path) in the environment variable `$LUT_FILENAME_C_VV`. If `$LUT_FILENAME_C_VV` is present, it will be used to store the GMF. A second file with the same name and extension `.zspace` is also generated. Note: the old GMF files need to be removed if new files need to be generated, i.e., if a different GMF version is requested.

**-calval** Ignore setting of $\sigma_0$ usability flags and perform $\sigma_0$ calibration.
During the ASCAT calibration and validation period, all $\sigma_0$ usability flags in the level 1b product are set, i.e., all beams are marked as invalid. These flags are ignored if this option is used. Moreover, a calibration of the $\sigma_0$. values is performed. See [*Verspeek, Portabella, Stoffelen and Verhoef*, 2007] for more details.

**-mss** Use the Multiple Solution Scheme for Ambiguity Removal.
If the Multiple Solution Scheme (MSS) is switched on, AWDP internally works with 144 different solutions for the wind vector. If MSS is switched

off, AWDP calculates two solutions at most. MSS is switched off as default.

**`-armeth <meth>`** Choose ambiguity removal method.
Valid methods are: `1strank` - the wind solution with the lowest distance to the GMF (residual) is selected, `bgclosest` - the wind solution closest to the background model wind is selected, `prescat` - see section 6.5, `2dvar` - 2DVAR, see section 6.4. The default is `2dvar`.

**`-par, -ana, -tc, -varqc, -ocf`** Various options intended for research activities.
More information can be found in the Fortran code of AWDP and genscat.

**`-binof <file>`** Write selected data of each WVC to a binary output file.
Data are written to a binary file `<file>`. This option is intended for research activities. More information on the file format can be found in the Fortran code of AWDP.

**`-writeall`** Write all data to BUFR output, including level 2 input data.
In the normal near-real time processing, a mixture of level 1b and (recent) level 2 data is fed into AWDP in order to provide more data, which is beneficial for ambiguity removal. Only those data rows which were level 1b input, are written to the level 2 output file. This option overrides this behaviour and writes all rows to the output file.

**`-handleall`** Perform NWP collocation, inversion, ambiguity removal and output writing in all WVCs.
By default, these steps are done only for WVCs which are level 1b input, see the description at the `–writeall` option. This option is useful for reprocessing level 2 data.

**`-nws <N>`** Write N wind solutions in BUFR output (default 2).
The number of wind solutions to be written into the ASCAT BUFR format is flexible due to the use of the so-called delayed replication and can be chosen between 1 (providing only the selected wind solution) and 144 (providing all wind solutions in MSS processing).

**`-subc <SC>`** Set id of sub-centre in each WVC of the BUFR output to SC.
By default it is copied from input.

**`-mon`** Switch on the monitoring function.
The monitoring results are written in an ASCII file with the name `monitoring_report.txt`. By default, no monitoring file is produced.

**`-verbosity <L>`** Set the verbosity level to L (default is 0).
If the verbosity level is -1 or smaller, no output is written to the standard output except error messages. If the verbosity level equals 0 only some top level processing information is written to output. If the verbosity level is 1 or greater, also additional information is given.

The normal mode of operation of AWDP is wind processing, i.e., a BUFR or PFS file is read and

the various processing steps are performed. Note that by default, AWDP does not recalculate data that are already present in the input. For example, if a WVC already contains model winds then the GRIB collocation will not be done for this WVC; if a WVC already contains wind solutions then the wind inversion will not be performed. This behaviour is desired when near-real time processing is performed and a mixture of level 1b and level 2 files is fed into AWDP. If one wants to perform reprocessing of level 2 files, the behaviour of AWDP can be changed by the command line options, e.g. the `-handleall` option.

Besides the wind processing, some other modes of operation are available. If one of the modes is invoked, AWDP internally sets some of the options in order to obtain the desired result. Note that these modes are always used in combination with the `-f <input file>` option.

**`-mononly`** Write the monitoring file without any processing.

**`-properties`** Write some properties of the last row of the input file.
The acquisition date and time and the sub-centre id are written to a small ASCII output file `properties.txt`.

**`-writeonly`** Write all data to BUFR output without processing.
This mode is useful to copy an input file to BUFR output without processing.

Running the command `awdp` without any command line options will display a list of all available command line options with a short explanation on the console. Running the command `awdp` with an illegal option will produce the same output, but preceded by an error message.

The output will be written into a BUFR file with the name

`INSTR_YYYYMMDD_HHMMSS_SAT_ORBIT_srv_o_SMPL(_CONT).l2_bufr`,

where

- `INSTR` is the instrument, `ascat` or `scatt`.
- `YYYYMMDD_HHMMSS` is the acquisition date and time (UTC) of the first data in the file.
- `SAT` is the satellite (6 characters), `ers1__`, `ers2__`, `metopa` or `metopb`.
- `ORBIT` is the orbit number (5 digits) of the first data in the file, `00000` for ERS data.
- `SMPL` is the WVC sampling (cell spacing), `250` for 25 km and `125` for 12.5 km.
- `_CONT` (contents) is omitted if the data contains both wind and soil moisture data. Otherwise it is set to `_ovw` (Ocean Vector Winds) or `_ssm`.

Example: `ascat_20070426_095102_metopa_02681_srv_o_250_ovw.l2_bufr`

## 2.5 Scripts

Directory `awdp/execs` contains a Bourne shell script `awdp_run` for running awdp with the correct environment variables. The script can be invoked with all valid command line options for awdp. In the same directory, there is also a script `awdp_gui.py` available. This script provides a convenient graphical user interface and builds and runs an AWDP command line depending on settings of available radio buttons, check boxes et cetera. This script requires Python to be installed

on your system. It may be necessary to change some of the environment variables set in the top part of the script.

## 2.6 Test data and test programs

Directory `awdp/tests` contains two BUFR files for testing the AWDP executable.. File `ascat_20070426_test_250.l1_bufr` contains ASCAT level 1b data from 26 April 2007, 9:51 to 10:29 UTC with 25 km cell spacing. The same data, but on 12.5 km cell spacing is available in file `ascat_20070426_test_125.l1_bufr`. The files `ECMWF*.grib` contain the necessary NWP data (SST, land-sea mask and wind forecasts) to perform the NWP collocation step.

The user can test the proper functioning of AWDP using the files in the `awdp/tests` directory. To do this, first create a small file containing a list of NWP files:

```
ls -1 ECMWF_200704260000_0* > nwpflist
```

Then run AWDP on 25-km and 12.5-km cell spacing:

```
../execs/awdp_run -f ascat_20070426_test_250.l1_bufr -mon –calval
-nwpfl nwpflist
```

```
../execs/awdp_run -f ascat_20070426_test_125.l1_bufr –mon -calval
-nwpfl nwpflist
```

The result should be two ASCAT level 2 files in BUFR format, called
`ascat_20070426_095102_metopa_02681_srv_o_250_ovw.l2_bufr`
and
`ascat_20070426_095100_metopa_02681_srv_o_125_ovw.l2_bufr`,
respectively.

Figure 2.3 shows the global coverage of the test run on 25 km. The colours indicate the magnitude of the wind speed as indicated by the legend. The result on 12.5 km should be very similar to this.

Directory `awdp/tests` also contains an ERS BUFR file in ESA format, called `scatt_20070426_test_250.l1_bufr` in ESA BUFR format. The data are from the same date as the ASCAT data in this directory and they can be processed using the same ECMWF files:

```
../execs/awdp_run -f scatt_20070426_test_250.l1_bufr – mon -nwpfl
nwpflist
```

The result should be an output file in ASCAT BUFR format, called
`scatt_20070426_063627_ers2___00000_srv_o_250_ovw.l2_bufr`

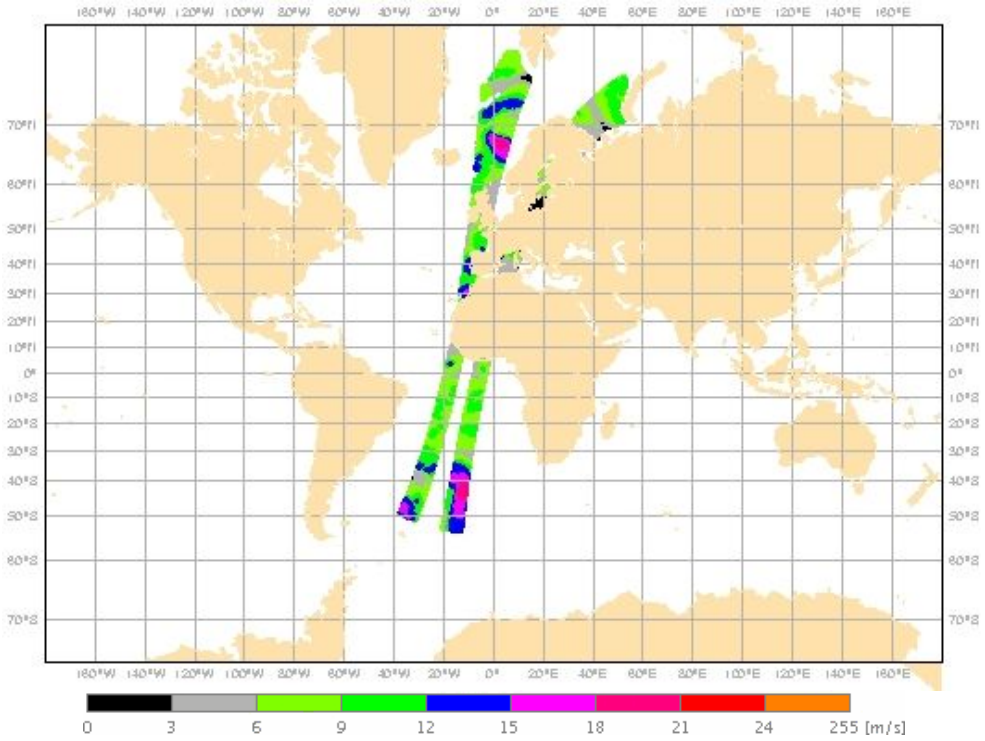


**Figure 2.3** Global coverage of the test run. Wind speed results for the 25 km product are shown.

Directory `genscat/support/bufr` contains a test program named `test_modules`. It is invoked by the genscat make system to construct the BUFR tables required by AWDP, but it can also be used to test the genscat BUFR module. The program is used as follows:

```
test_modules [BUFRinput]
```

where `BUFRinput` is the BUFR input file.

If omitted, the program uses as default input the file `testreading.bufr` in directory `genscat/support/bufr`. The output is written to a BUFR file named `testwriting.bufr`. The directory also contains a shell script named `run_test_modules` that sets the environment variables required and executes the program. Further information can be found in the comment lines of the source code of `test_modules`.

Directory `genscat/support/grib` contains test programs named `test_read_GRIB1`, `test_read_GRIB2` and `test_read_GRIB3`. The programs can be run from the command line and read in the GRIB file `testfile.grib` in directory `genscat/support/grib`. Some properties of this file are written to ASCII output files. Note that the environment variable `$LOCAL_DEFINITION_TEMPLATES` needs to be set to directory `(…)/genscat/support/grib/gribtemplates`.

Subdirectories `Compiler_Features`, `convert`, `ErrorHandler`, `singletonfft`, `file`, `BFGS`, `num`, `sort` and `datetime` of `genscat/support` contain test programs for the module in that subdirectory. The test programs write their result to the standard output. In some cases, a copy of the output is contained in the `.output` files for comparison. Table 2.6 gives an overview of the genscat test programs.

| Subdirectory | Program name | Output file | Remarks |
|---|---|---|---|
| bufr | test_modules | testwriting.bufr | Part of make system |
| grib | test_read_GRIB* | several | GRIB handling |
| Compiler_Features | TestCompiler_Features | - | Command line handling |
| convert | test_convert | test_convert.output | Wind speed conversion |
| ErrorHandler | TestErrorHandler | - | Error handling |
| singletonfft | TestSingleton | - | Fast Fourier Transform |
| file | TestLunManager | TestLunManager.output | File management |
| BFGS | Test_BFGS | - | Minimization |
| num | test_numerics | test_numerics.output | Numerical issues |
| pfs | test_pfs_ascat | - | Read PFS file |
| sort | SortModTest | SortModTest.output | Array sorting |
| datetime | TestDateTimeMod | TestDateTimeMod.output | Date and time conversion |

**Table 2.6** Test programs in genscat/support.

## 2.7 Documentation

Directory `awdp/doc` contains documentation on AWDP, including this document. Further information can be found in the readme text files, and in the comments in scripts, Makefiles and source code.

# Chapter 3

# AWDP product specification

## 3.1 Purpose of program AWDP

The ASCAT Wind Data Processor (AWDP) program has been developed to fully exploit $\sigma_0$ data from the ASCAT scatterometer instrument on the MetOp satellites or the AMI scatterometer instrument on the European Remote Sensing (ERS) satellites, to generate surface winds. AWDP may be used for real-time data processing. The main application of AWDP is to form the core of an Observation Operator for ASCAT scatterometer data within an operational Numerical Weather Prediction System.

Program AWDP is also a level 2 data processor. It reads data from the EUMETSAT level 1b ASCAT BUFR or PFS product or from the ESA ERS scatterometer BUFR product. AWDP applies algorithms for inversion, quality control, and Ambiguity Removal at various spatial resolutions. These methods are mainly developed and published by KNMI. The output of AWDP is a BUFR file in ASCAT BUFR format.

## 3.2 Output specification

The wind vectors generated by AWDP represent the instantaneous mean surface wind at 10 m anemometer height in a 2D array of Wind Vector Cells (WVCs) with specified size ($25 \times 25$ km$^2$ or $12.5 \times 12.5$ km$^2$, depending on the cell spacing of the input product). These WVCs are part of the ground swath of the instrument.

In conventional mode, the wind output for every WVC consists of up to 2 ambiguities (wind vector alternatives, with varying probabilities). The selected wind vector is indicated by a selection index. For every WVC additional parameters are stored. These are e.g.: latitude, longitude, time information, orbit and node numbers, background wind vector, cell quality flag, and information on the scatterometer beams including $\sigma_0$ and $K_p$ data. The output file is structured according to the same conventions as the ASCAT level 1b input, also if ERS data are processed.

The ASCAT BUFR format consist of three main sections: one section containing level 1b information which is copied from the input data, one section containing Surface Soil Moisture (SSM) level 2 information, which is also copied from the input, and one section containing level 2 wind data, which is calculated in AWDP. The ASCAT BUFR data descriptors are listed in Appendix C.

## 3.3 Input specification

Input of AWDP is the ASCAT level 1b BUFR or PFS Data Product. These products are created by EUMETSAT; see [*WMO*, 2007] and [*Figa-Saldaña and Wilson,* 2005]. Alternatively, the ERS scatterometer wind product in BUFR can be used as input; see [*UK Met Office*, 2001].

It is also possible to reprocess level 2 ASCAT or ERS data in ASCAT BUFR format, and treat them as if they are level 1b data. To achieve this, some command line options need to be set; see section 2.4.

Apart from the scatterometer data, GRIB files containing NWP output with global coverage are necessary for the wind processing. At least three wind forecasts with forecast time intervals of 3 hours are necessary to perform interpolation with respect to time and location. Apart from this, GRIB fields of Sea Surface Temperature and Land Sea Mask are necessary for land and ice masking.

## 3.4 System requirements

Table 3.1 shows the platform and compiler combinations for which AWDP has been tested. However, the program is designed to run on any Unix (Linux) based computer platform with a Fortran compiler and a C compiler. The equivalent of a modern personal computer will suffice to provide a timely NRT wind product. AWDP requires about 100-150 MB disk space when installed and compiled.

| Platform | Fortran compiler | C compiler |
|---|---|---|
| Suse Linux work station | Portland pgf90<br>GNU g95<br>GNU gfortran | GNU gcc |
| SunOS Unix | Sun Fortran | Sun C |
| SGI Irix | MIPSpro Fortran compiler | MIPSpro C compiler |
| SGI Altix | Intel Fortran compiler | Intel C compiler |
| Windows XP PC with Cygwin | GNU g95 | GNU gcc |

**Table 3.1** Platform and compiler combinations for which AWDP has been tested.

AWDP may also run in other environments, provided that the environment variables discussed in section 2.2 are set to the proper values, and that the BUFR library is properly installed. For Windows a Unix emulator like Cygwin is needed.

## 3.5 Details of functionality

### 3.5.1 BUFR IO and coding

Data sets of near-real time meteorological observations are generally coded in the Binary Universal Form for Representation (BUFR). BUFR is a machine independent data representation system (but it contains binary data, so care must be taken in reading and writing these data under different operating systems). A BUFR message (record) contains observational data of any sort in a self-descriptive manner. The description includes the parameter identification and its unit, decimal, and scaling specifications. The actual data are in binary code. The meta data are stored in

BUFR tables. These tables are therefore essential to decode and encode the data.

BUFR tables are issued by the various meteorological centres. The largest part of the data descriptors specified in the BUFR tables follows the official BUFR descriptor standards maintained by the World Meteorological Organization (WMO, http://www.wmo.int/). However, for their different observational products meteorological centres do locally introduce additional descriptors in their BUFR tables.

Appendix C contains a listing of the data descriptors of the BUFR data input and the BUFR data output of the AWDP program in the ASCAT BUFR product format. For more details on BUFR, the reader is referred to [*Dragosavac*, 1994].

ECMWF maintains a library of routines for reading (writing) and decoding (encoding) the binary BUFR messages. This library forms the basis of the genscat BUFR module and hence the AWDP program BUFR interface, see Chapter 8.

### 3.5.2 Product resolution

An important feature of the AWDP program is that it may produce a level 2 wind product on different resolutions. The resolution of the level 2 wind product is the same as that of the level 1b input product. ASCAT data are available in two different resolutions: 50 km resolution with 25 km cell spacing (also known as the ASCAT operational product, SZO) and 25 km resolution with 12.5 km cell spacing (also known as the ASCAT research product, SZR). Of course, there is a trade off between the resolution and the statistical error of the mean wind vectors. The statistical error of the wind vectors for the higher resolution is a topic for further testing.

### 3.5.3 WVC triplet completion and row merging

AWDP sorts the WVC rows in the input file by their acquisition date and time and merges WVC information if duplicate rows occur. The duplicate information is considered and the output will contain as much useful information as is available in the input WVCs. This is especially useful if direct readout data from different ground stations is processed. Sometimes a WVC from one ground station contains only fore beam information, whereas the corresponding WVC from a second ground station contains only the mid and aft beam information. AWDP will combine the information and it will process and write one WVC containing all three beams.

### 3.5.4 Quality control

The quality of every WVC is controlled. Before processing the beam data, checks are done on the completeness and usability of the $\sigma_0$ data. After the wind inversion step, the distance of the first rank wind solutions to the GMF is considered. If this value is too large, the wind solutions are flagged. The $K_p$ values are also considered. If one of the three beam $K_p$ values is above a threshold which is wind speed dependent, the wind information is flagged [*Stoffelen*, 1998].

### 3.5.5 Inversion

In the inversion step of wind retrieval, the radar backscatter observations in terms of the Normalized Radar Cross Sections ($\sigma_0$'s) are converted into a set of ambiguous wind vector solutions. In fact, a Geophysical Model Function (GMF) is used to map a wind vector (specified in

term of wind speed and wind direction) to a $\sigma_0$ value. The GMF depends not only on wind speed and wind direction but also on the measurement geometry (relative azimuth and incidence angle) and beam parameters (frequency and polarization). Currently, the CMOD5 GMF which was developed for ERS is in use, see [*Hersbach, Stoffelen and de Haan*, 2007], but improvements are under study.

The AWDP program also includes the Multiple Solution Scheme (MSS). In MSS mode, a large number of wind vector solutions is produced, typically 144. The wind vector solutions are ranked according to their probability based on the MLE and constitute the full wind vector probability density function. Subsequently, the 2DVAR Ambiguity Removal method, see, e.g., section 3.5.6, is applied with a much larger set of wind vector solutions. The output BUFR format can accommodate any number of wind solutions due to the use of the so-called delayed descriptor replication. Details on the KNMI inversion approach can be found in [*Portabella,* 2002]. For SeaWinds, MSS compares better to an independent NWP model reference and buoys than conventional two or four-solution schemes [*Portabella and Stoffelen,* 2004; *Vogelzang et al.*, 2008], but for ERS and ASCAT this needs to be investigated further.

Technical information on the KNMI inversion approach can be found in Chapter 5.

### 3.5.6 Ambiguity Removal

The Ambiguity Removal (AR) step of the wind retrieval is the selection of the most probable surface wind vector among the available wind vector solutions, the so-called ambiguities. Various methods have been developed for AR. More information on Ambiguity Removal is given in Chapter 6. The default method implemented in AWDP is the KNMI 2DVAR AR scheme. A description of its implementation can be found in section 6.4. The Multiple Solution Scheme (MSS) offers the possibility to postpone AR to the NWP step in order to treat all information from models and measurements in the same manner. Further details on the algorithms and their validation can be found in the reports [*de Vries and Stoffelen*, 2000; *de Vries, Stoffelen and Beysens*, 2005].

The performance of 2DVAR with meteorological balance constraints was tested and optimized for ERS data. It was found to be superior to other schemes.

### 3.5.7 Monitoring

For the automatic ingestion of observations into their NWP systems, meteorological centres require quality checks on the NRT products. For the ASCAT wind product a monitor flag is under development, analogous to the one developed for the SeaWinds Wind Product. This flag indicates that several measures on the level of corruption of the output BUFR files are above a specified threshold. Onset of the flag indicates that the input should be rejected for ingestion by the NWP system. Details on the monitor developed can be found in the NWP SAF document [*de Vries, Stoffelen and Beysens*, 2005].

## 3.6 Details of performance

AWDP is delivered with two example BUFR input files containing data from 26 April 2007. They are named `ascat_20070426_test_250.l1_bufr` (25 km cell spacing) and

`ascat_20070426_test_125.l1_bufr` (12.5 km cell spacing) and contain approximately half an orbit of data. Moreover, a set of ECMWF GRIB files containing the necessary NWP output is supplied. Table 3.2 gives the approximate times needed for processing these files under various options on a personal workstation with a 2.66 GHz Pentium 4 processor under Linux using the GNU g95 Fortran compiler.

| Cell spacing (m) | MSS? | Inversion (seconds) | AR (seconds) | BUFR IO (seconds) | GRIB IO (seconds) | Total (seconds) |
|---|---|---|---|---|---|---|
| 25000 | No | 33 | 8 | 3 | 1 | 46 |
| 12500 | No | 132 | 16 | 8 | 3 | 160 |
| 12500 | Yes | 156 | 91 | 8 | 3 | 263 |

**Table 3.2** Approximate times needed by AWDP to process example BUFR files under various input resolutions and options.

As can be seen from table 3.2, the use of MSS results in slightly larger processing times needed for inversion and in much larger processing times needed for AR. The computation time, of course, increases with increasing resolution.

The choice of platform, compiler and compiler settings will generate a large variation in the processing times. Using the Portland pgf90 compiler rather than the GNU g95 compiler in the examples in table 3.2 will result in processing times that are 25% to 50% smaller.

# Chapter 4

# Program Design

In this chapter, the design of the AWDP program is described in detail. Readers to whom only a summary will suffice are referred to the Top Level Design (TLD) in section 4.1. Readers who really want to know the very detail should not only read the complete chapter, but also the documentation within the code.

## 4.1 Top Level Design

### 4.1.1 Main program

The main program, AWDP, (file `awdp` in the `awdp/src` directory) is a Unix (Linux) executable which processes ASCAT BUFR or PFS or ERS BUFR input files. The main output consists of BUFR files. The output BUFR messages are always in the ASCAT BUFR format, for a list of descriptors see appendix C. The user may provide arguments and parameters according to Unix command line standards. The purpose of the different options is described in the User Manual (Chapter 2).

When executed, the AWDP program logs information on the standard output. The detail of this information may be set with the verbosity flag. The baseline of processing is described in Figure 4.1. A more detailed representation of the AWDP structure is given in Appendices A and B.

The first step is to process the arguments given at the command line using the genscat *Compiler_Features* module. Next, the AWDP program reads the input file specified in the arguments. The BUFR messages or PFS records are read and mapped onto the AWDP data structure, see subsection 4.1.3. As part of the pre-processing a similar AWDP data structure is created for the output. Subsequently, the input data are sorted with respect to data acquisition time, duplicate rows are merged and the output data structure is filled with level 1b ($\sigma_0$ related) data. Then, the NWP GRIB data (wind forecasts, land-sea mask and sea surface temperature) are read and the data are collocated with the Wind Vector Cells. The next steps are the inversion and the ambiguity removal. These steps are performed on the output data. The program ends with the post-processing step (which includes some conversions and the monitoring) and the mapping of the output data structure onto BUFR messages of the BUFR output file. The different stages in the processing correspond directly to specific modules of the code. These modules form the process layer, see section 4.3.

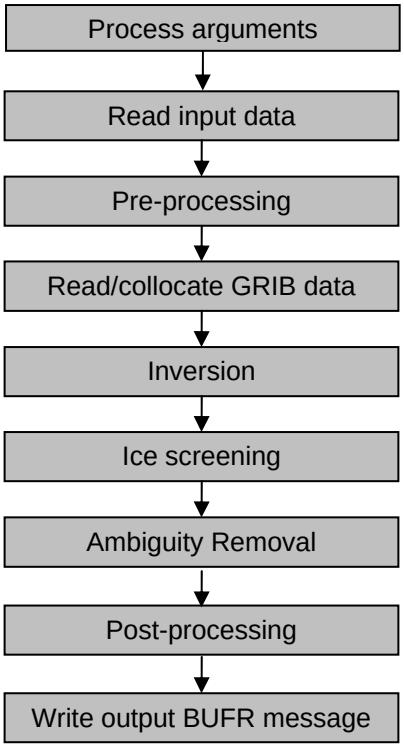


**Figure 4.1** Baseline of the ASCAT Wind Data Processor

### 4.1.2 Layered model structure

AWDP is a Fortran 90 program consisting of several Fortran 90 modules which are linked after their individual compilation. The AWPD program is set up from two layers of software modules. The purpose of the layer structure is to divide the code into generic scatterometer processing software and ASCAT specific software. Details on the individual modules can be found in sections 4.2 and 4.3.

The first layer (the process layer) consists of modules which serve the main steps of the process.

| Module name | Tasks | Comments |
|---|---|---|
| *awdp_data* | Definition of data structures | |
| *awdp_bufr* | BUFR file handling | Interface to `genscat/support/bufr` |
| *awdp_pfs* | PFS file handling | Interface to `genscat/support/pfs` |
| *awdp_prepost* | Sorting of input | Duplicate rows are merged |
| | Quality control | Usability of input data is determined |
| | Post processing | Setting of flags |
| | Monitoring | |
| | Clean up | Deallocation of used memory |
| *awdp_grib* | GRIB file handling | Interface to `genscat/support/grib` |
| | Collocation of GRIB data | NWP data are interpolated w.r.t. time and location |
| *awdp_inversion* | Inversion | Interface to `genscat/inversion` |

| Module name | Tasks | Comments |
|---|---|---|
| *awdp_ambrem* | Ambiguity Removal | Interface to `genscat/ambrem` |
| *awdp_icemodel* | Ice screening | Interface to `genscat/icemodel` |

**Table 4.1** AWDP process modules.

Each module contains code for performing one or more of the specific tasks. These tasks are shortly described in table 4.1. A more elaborate description is given in section 4.3. The first module listed, *awdp_data* is a general support module. This module is used by the other modules of the process layer for the inclusion of definitions of the data structures and the support routines.

The second module layer is the genscat layer. The genscat module classes (i.e., groups of modules) used in the AWDP program are listed in table 4.2. The genscat package is a set of generic modules which can be used to assemble processors as well as pre, and post-processing tools for different scatterometer instruments available to the user community. A short description of the main (interface) modules is given in section 4.2. The most important classes of modules are related to the inversion processing step (Chapter 5), the Ambiguity Removal step (Chapter 6), the BUFR file handling (Chapter 8), and the GRIB file handling (Chapter 9). The genscat modules are located in subdirectory `genscat`.

In addition, genscat contains a large support class to convert and transform meteorological, geographical, and time data, to handle file access and error messages, sorting, and to perform more complex numerical calculations on minimization and Fourier transformation. Many routines are co-developed for ERS, ASCAT and SeaWinds data processing.

| Module class | Tasks | Description |
|---|---|---|
| *Ambrem* | Ambiguity Removal | 2DVAR and other schemes, see Chapter 6 |
| *Inversion* | Wind retrieval | Inversion in one cell, see Chapter 5 |
| *IceModel* | Ice screening | Uses ice line and wind cone for ice discremination |
| *Support* | BUFR support | *BufrMod*, based on ECMWF library |
| | PFS support | Reading of PFS files |
| | GRIB support | *gribio_module*, based on ECMWF library |
| | FFT, minimization | Support for 2DVAR |
| | Error handling | Print error messages |
| | File handling | Finding, opening and closing free file units |
| | Conversion | Conversion of meteorological quantities |
| | Sorting | Sorting of ambiguities to their probability |
| | Date and time | General purpose |

**Table 4.2** genscat module classes.

### 4.1.3 Data Structure

Along track, the ASCAT swath is divided into rows. Within a row (across track), the ASCAT orbit is divided into cells, also called Wind Vector Cells (WVCs) or nodes. This division in rows and cells forms the basis of the main data structures within the AWDP package. In fact, both the input and the output structure are one dimensional arrays of the row data structure, *row_type*. These arrays represent just a part of the swath. Reading and writing (decoding and encoding) ASCAT BUFR files corresponds to the mapping of a BUFR message to an instance of the *row_type* and

vice versa.

The main constituent of the *row_type* is the cell data structure, *cell_type*, see figure 4.2. Since most of the processing is done on a cell-by-cell basis the *cell_type* is the pivot data structure of the processor.

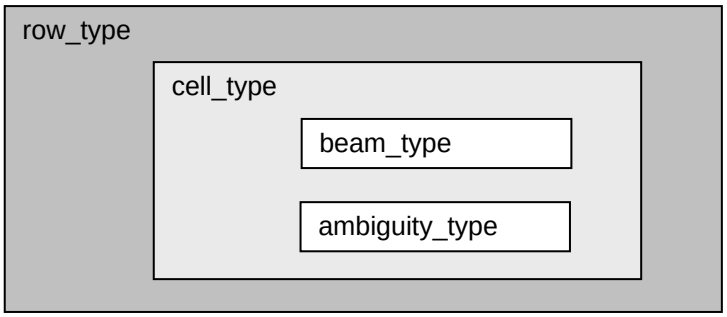


**Figure 4.2** Schematic representation of the nested data definitions in the *row_type* data structure.

The $\sigma_0$ related level 1b data of a cell are stored in a data structure called *beam_type*. Every cell contains three instances of the *beam_type,* corresponding to the fore, middle and aft beams.

A cell may also contain an array of instances of the *ambiguity_type* data structure. This array stores the results of a successful wind retrieval step, the wind ambiguities (level 2 data). Details of all the data structures and methods working on them are described in the next sections.

### 4.1.4 Quality flagging and error handling

Important aspects of the data processing are to check the validity of the data and to check the data quality. In the AWDP program two flags are set for every WVC, see table 4.3. The flags themselves do not address a single aspect of the data, but the flags are composed of several bits each addressing a specific aspect of the data. A bit is set to 0 (1) in case the data is valid (not valid) with respect to the corresponding aspect. In order to enhance the readability of the code, each flag is translated to a data type consisting of only booleans (false = valid, true = invalid). On input and output these data types are converted to integer values by *set* and *get* routines.

| Flag | Tasks | Description |
|---|---|---|
| wvc_quality | Quality checking | In BUFR output |
| process_flag | Range checking | Not in BUFR output |

**Table 4.3** Flags for every WVC (attributes of *cell_type*).

Apart from the flags on WVC level, also the beams contain quality indicators. Most of them are implemented as real values ranging from 0 to 1, where 0 stands for good quality and 1 for degraded quality. See section 4.3.1 for more information on this.

### 4.1.5 Verbosity

Every routine in a module may produce some data and statements for the log of the processor. To

control the size the log, several modules contain parameters for the level of verbosity. The verbosity of the AWDP program may be controlled by the verbosity command line option `-verbosity`. In general, there are three levels of verbosity specified:

≤ -1: be as quiet as possible;

0: only report top level processing information;

≥ 1: report additional information.

Of course, errors are logged in any case. Table 4.4 gives a (incomplete) list of verbosity parameters. They are not all set by the command line option as some of them serve testing and debugging purposes.

| Module | Verbosity parameter |
|---|---|
| *Ambrem2Dvar* | *TDVverbosity* |
| *AmbremBGclosest* | *BGverbosity* |
| *BatchMod* | *BatchVerbosity* |
| *Ambrem* | *AmbremVerbosity* |
| *awdp_bufr* | *BufrVerbosity* |
| *awdp_grib* | *GribVerbosity* |

**Table 4.4** Verbosity parameters.

## 4.2 Module design for genscat layer

### 4.2.1 Module *inversion*

The module *inversion* contains the *genscat* inversion code. Module *post-inversion* contains some routines specific for ERS and ASCAT inversion and quality control. The modules are located in subdirectory `genscat/inversion`.. Details of this module are described in Chapter 5. In the AWDP program, the inversion module is only used in the *awdp_inversion* module, see section 4.3.6.

### 4.2.2 Module *ambrem*

The module *ambrem* is the main module of the genscat Ambiguity Removal code. It is located in subdirectory `genscat/ambrem`. Details of this module are described in Chapter 6. In the AWDP program, the *ambrem* module is only used in the *awdp_ambrem* module, see section 4.3.7.

### 4.2.3 Module *icemodel*

The module *icemodel* contains the *genscat* ice screening code. It is located in subdirectory `genscat/icemodel`. In the AWDP program, the *icemodel* module is only used in the *awdp_icemodel* module, see section 4.3.8.

### 4.2.4 Module *Bufrmod*

Genscat contains several support modules. In particular, the *BufrMod* module is the Fortran 90 wrapper around the BUFR library used for BUFR input and output. It is located in subdirectory

`genscat/support/bufr`. Details of this module are described in Chapter 8. In the AWDP program, the *BufrMod* module is only used in the *awdp_bufr* module, see subsection 4.3.2.

### 4.2.5 Module *gribio_module*

The *gribio_module* module is the Fortran 90 wrapper around the GRIB library used for GRIB input and collocation of the NWP data with the scatterometer data. It is located in subdirectory `genscat/support/grib`. Details of this module are described in Chapter 9. In the AWDP program, the *gribio_module* module is only used in the *awdp_grib* module, see subsection 4.3.5.

### 4.2.6 Support modules

Subdirectory `genscat/support` contains more support modules besides *Bufrmod* and *gribio_module*. The KNMI 2DVAR Ambiguity Removal method requires minimization of a cost function and numerical Fourier transformation. These routines are located in subdirectories `BFGS` and `singletonfft`, respectively, and are discussed in more detail in section 6.4.

Subdirectory `Compiler_Features` contains module *Compiler_Features* for handling some compiler specific issues, mainly with respect to command line argument handling. The `Makefile` in this directory compiles on of the available source files, depending on the Fortran compiler used.

Subdirectory `convert` contains module *convert* for the conversion of meteorological and geographical quantities, e.g. the conversion of wind speed and direction into *u* and *v* components and vice versa..

Subdirectory `datetime` contains module *DateTimeMod* for date and time conversions. AWDP only uses routines *GetElapsedSystemTime* (for calculating the running time of the various processing steps), and *julian2ymd* and *ymd2julian* (for conversion between Julian day number and day, month and year). Module *DateTimeMod* needs modules *ErrorHandler* and *numerics*.

Subdirectory `ErrorHandler` contains module *ErrorHandler* for error management. This module is needed by module *DateTimeMod*.

Subdirectory `file` contains module *LunManager* for finding, opening and closing free logical units in Fortran. AWDP uses only routines *get_lun* and *free_lun* for opening and closing of a logical unit, respectively.

Subdirectory `num` contains module *numerics* for handling missing values, for instance in the BUFR library. This module is needed by module *DateTimeMod* and is used in the test program `test_modules`.

Subdirectory `pfs` contains module `pfs_ascat` for opening, reading and closing of files in PFS format.

Subdirectory `sort`, finally, contains module *SortMod* for sorting the rows according to their acquisition date and time, or the wind vector solutions according to their probability.

## 4.3 Module design for process layer

The process layer consists of the modules *awdp_data*, *awdp_bufr, awdp_pfs*, *awdp_prepost*,

*awdp_grib*, *awdp_inversion*, *awdp_icemodel* and *awdp_ambrem*. The routines present in these modules are described in the next sections.

### 4.3.1 Module *awdp_data*

The module *awdp_data* contains all the important data types relevant for the processing. Elementary data types are introduced for the most basic data structures of the processing. These are e.g. *wind_type* and *time_type*. Using these data types (and of course the standard types as integer, real etc.), more complex (composed) data types are derived. Examples are *beam_type*, *ambiguity_type*, *cell_type*, and *row_type*. A complete description of all types is given below. The attributes of all these types have intentionally self-documenting names.

**Ambiguity data**: The *ambiguity_type* data type contains information on an individual ambiguity (wind vector solution). The attributes are listed in table 4.5. The routine *init_ambiguity()* sets all ambiguity data to missing. The routine *print_ambiguity()* may be used to print all ambiguity data.

| Attribute | Type | Description |
|---|---|---|
| *wind* | *wind_type* | Wind vector solution |
| *prob* | real | Probability of wind vector solution |
| *conedistance* | real | Distance of solution to the GMF |

**Table 4.5** Ambiguity data structure.

**Beam data**: Every WVC contains three beams. The information of every beam is stored in the data type *beam_type*. The attributes are listed in table 4.6. Most of the attributes are explained in detail in [*Wilson, Figa-Saldaña and O'Clerigh*, 2004]. The routine *init_beam()* sets all beam data to missing and the routine *test_beam* checks if the data in the beam are within valid ranges. The routine *print_beam()* may be used to print all beam data.

| Attribute | Type | Description |
|---|---|---|
| *identifier* | integer | Beam number: 1 = fore, 2 = mid, 3 = aft |
| *incidence* | real | Incidence angle (degrees, 0 is vertical, 90 is horizontal) |
| *azimuth* | real | Radar look angle (degrees, counted clockwise from the south) |
| *sigma0* | real | Radar backscatter ($\sigma_0$) in dB |
| *noise_val* | real | Noise value in % |
| *kp_estim_qual* | *kp_estim_qual_type* | Flag related to the quality of the Kp estimate |
| *s0_usability* | integer | Usability of $\sigma_0$: 0 = good, 1 = usable, 2 = bad |
| *synt_data_quantity* | real | Amount of synthetic data in $\sigma_0$ (0..1) |
| *synt_data_quality* | real | Quality of used synthetic data in $\sigma_0$ (0..1) |
| *orbit_quality* | real | Satellite orbit and attitude quality (0..1) |
| *solar_reflec* | real | Solar array reflection contamination in $\sigma_0$ (0..1) |
| *telemetry* | real | Telemetry quality (0..1) |
| *extrapol_ref_pres* | real | Presence of extrapolated reference functions (0..1) |
| *land_frac* | real | Land fraction in $\sigma_0$ (0..1) |

**Table 4.6** Beam data structure.

**Cell Data**: The *cell_type* data type is a key data type in the AWDP program, because many processing steps are done on a cell by cell basis. The attributes are listed in table 4.7. The routine

*init_cell()* sets the cell data to missing values. Also the flags are set to missing. The routine *test_cell()* tests the validity of data. This routine sets the cell process flag. The routine *print_cell()* may be used to print the cell data.

| Attribute | Type | Description |
|---|---|---|
| *centre_id* | integer | Identification of originating/generating centre |
| *sub_centre_id* | integer | Identification of originating/generating sub-centre |
| *software_id_l1b* | integer | Software identification of level 1 processor |
| *satellite_id* | integer | Satellite identifier |
| *sat_instruments* | integer | Satellite instrument identifier |
| *sat_motion* | real | Direction of motion of satellite |
| *time* | *time_type* | Date and time of data acquisition |
| *lat* | real | Latitude of WVC |
| *lon* | real | Longitude of WVC |
| *pixel_size_hor* | real | Distance between WVCs (meters) |
| *orbit_nr* | integer | Orbit number |
| *node_nr* | integer | Across track cell number |
| *height_atmosphere* | real | Height of atmosphere used |
| *loss_unit_lenght* | real | Loss per unit length of atmosphere |
| *beam_collocation* | *beam_collocation_type* | Beam collocation flag |
| *beam (3)* | *beam_type* | Beam data |
| *full_res* | *full_res_type* | Averaged full resolution data |
| *software_id_sm* | integer | Soil moisture information |
| *database_id* | integer | Soil moisture information |
| *surface_sm* | real | Soil moisture information |
| *surface_sm_err* | real | Soil moisture information |
| *sigma0_40* | real | Soil moisture information |
| *sigma0_40_err* | real | Soil moisture information |
| *slope_40* | real | Soil moisture information |
| *slope_40_err* | real | Soil moisture information |
| *sm_sensitivity* | real | Soil moisture information |
| *dry_backscatter* | real | Soil moisture information |
| *wet_backscatter* | real | Soil moisture information |
| *mean_surface_sm* | real | Soil moisture information |
| *rain_fall_detect* | real | Soil moisture information |
| *sm_corr_flag* | integer | Soil moisture information |
| *sm_proc_flag* | integer | Soil moisture information |
| *sm_quality* | real | Soil moisture information |
| *snow_cov_frac* | real | Soil moisture information |
| *froz_land_frac* | real | Soil moisture information |
| *inund_wet_frac* | real | Soil moisture information |
| *topo_complexity* | real | Soil moisture information |
| *software_id_wind* | integer | Software identification of level 2 wind processor |
| *generating_app* | integer | Generating application of model information |
| *model_wind* | *wind_type* | Model wind used for Ambiguity Removal |
| *ice_prob* | real | Probability of ice |
| *ice_age* | real | Ice age A-parameter |
| *wvc_quality* | *wvc_quality_type* | WVC quality flag |
| *num_ambigs* | integer | Number of ambiguities present in WVC |
| *selection* | integer | Index of selected wind vector |
| *skill* | real | Parameter used for PreScat Ambiguity Removal |
| *ambig (0..144)* | *ambiguity_type* | Array of wind ambiguities |
| *ice* | *icemodel_type* | Ice information |
| *stress_param* | *nwp_stress_param_type* | Wind stress information |

| Attribute | Type | Description |
|---|---|---|
| *process_flag* | *process_flag_type* | Processing flag |
| *level_of_input* | integer | Level of input data (1 or 2) |

**Table 4.7** Cell data structure.

All soil moisture information is read from the input BUFR file into the cell data structure and not used within the program. It is written to the output BUFR file at the end of the processing.

**Full resolution data**: The *full_res_type* contains average full resolution data, read from a PFS file, which are used to replace the 25-km or 12.5-km beam data. The attributes are listed in table 4.8. The routine *init_full_res()* sets the full resolution averaged data to zero. The routine *print_full_res()* may be used to print the full resolution data.

| Attribute | Type | Description |
|---|---|---|
| *count_tot* | integer | Number of full res measurements used |
| *lat* | real | Mean value of full res lats |
| *lon* | real | Mean value of full res lons |
| *count_fore* | integer | Number of full res fore beams used |
| *incidence_fore* | real | Mean value of full res values |
| *azimuth_fore* | real | Mean value of full res values |
| *sigma0_fore* | real | Mean value of full res values |
| *land_frac_fore* | real | Mean value of full res values |
| *count_mid* | integer | Number of full res mid beams used |
| *incidence_mid* | real | Mean value of full res values |
| *azimuth_mid* | real | Mean value of full res values |
| *sigma0_mid* | real | Mean value of full res values |
| *land_frac_mid* | real | Mean value of full res values |
| *count_aft* | integer | Number of full res aft beams used |
| *incidence_aft* | real | Mean value of full res values |
| *azimuth_aft* | real | Mean value of full res values |
| *sigma0_aft* | real | Mean value of full res values |
| *land_frac_aft* | real | Mean value of full res values |

**Table 4.8** Full res data structure.

**Ice model data**: The *icemodel_type* contains information related to the ice screening. The attributes are listed in table 4.9. The routine *init_icemodel()* sets the ice model data to missing values. The routine *print_icemodel()* may be used to print the ice data.

| Attribute | Type | Description |
|---|---|---|
| *class* | integer | Code for WVC being ice or wind |
| *ii* | integer | Coordinate on the ice map |
| *jj* | integer | Coordinate on the ice map |
| *b* | real | Ice coordinate |
| *c* | real | Ice coordinate |
| *dIce* | real | Distance to the ice line |

**Table 4.9** Ice model data structure.

**NWP stress parameter data**: The *nwp_stress_param_type* data type contains information

relevant for the ice screening and wind stress calculations (stress calculation is not yet implemented in AWDP). The attributes are listed in table 4.10. The routine *init_nwp_stress_param()* sets the NWP stress parameter data to missing values. The routine *print_nwp_stress_param ()* may be used to print the stress data.

| Attribute | Type | Description |
|---|---|---|
| *u* | real | Eastward (zonal) wind component |
| *v* | real | Northward (meridional) wind component |
| *t* | real | Air temperature |
| *q* | real | Specific humidity |
| *sst* | real | Sea surface temperature |
| *chnk* | real | Charnok parameter |
| *sp* | real | Surface pressure |

**Table 4.10** NWP stress parameter data structure.

**Row data**: The data of a complete row of the swath is stored in the data type *row_type*, see table 4.11. A complete row corresponds to a single BUFR message in the AWDP output. The level 1 BUFR data may contain more than one row per BUFR message..

| Attribute | Type | Description |
|---|---|---|
| *time_stamp* | integer | Time stamp of row data in seconds, used for sorting |
| *num_cells* | integer | Actual number of WVC's |
| *Cell(82)* | *cell_type* | Array of Wind Vector Cells |

**Table 4.11** Row data structure.

**Time data**: The *time_type* data type contains a set of 6 integers representing both the date and the time, see table 4.12. The routine *init_time()* sets the time entries to missing values. The routine *test_time()* tests the validity of the date and time specification (see also the cell process flag). The routine *print_time()* can be used to print the time information.

| Attribute | Type | Description |
|---|---|---|
| *year* | integer | 19XX or 20XX |
| *month* | integer | 1 – 12 |
| *day* | integer | 1 – 31 |
| *hour* | integer | 0 – 23 |
| *minute* | integer | 0 – 59 |
| *second* | integer | 0 – 59 |

**Table 4.12** Time data structure.

**Wind Data**: The *wind_type* data type contains the wind speed and wind direction, see table 4.13. The routine *init_wind()* sets the wind vector to missing. The routine *print_wind()* may be used to print the wind vector. The routine *test_wind()* tests the validity of the wind specification, see also the cell process flag.

| Attribute | Type | Description |
|---|---|---|
| *speed* | real | Wind speed |
| *dir* | real | Wind direction |

**Table 4.13** Wind data structure.

Some special data types are introduced for the data (quality) flags. These are discussed below.

**Beam collocation flag**: The *beam_collocation_type* data type is used to indicate whether data of the three beams is originating from a single ground station or from multiple ground stations (collocated data). This is relevant for so-called direct readout data from different ground stations which maybe merged into one single product. In a WVC, e.g. the fore beam information from one ground station may be combined with the mid and aft beam information from another ground station, in order to make a complete WVC. The attributes are listed in table 4.14. The routine *get_beam_collocation()* converts an integer value to the logical beam collocation structure. The routine *set_beam_collocation ()* converts a logical beam collocation structure to an integer value.

| Attribute | Bit | $2^{Bit}$ | Description |
|---|---|---|---|
| *missing* | | | Flag not set (all bits on) |
| *collocation* | 0 | 1 | Beam information originates from different ground stations |

**Table 4.14** Beam collocation flag bits.

**$K_p$ estimate quality flag**: The *kp_estim_qual_type* data type contains the flag indicating the quality of the $K_p$ estimate. Each one of the three beams in a WVC contain an instance of this flag. The attributes are listed in table 4.15. The function *get_kp_estim_qual()* interprets an integer flag (BUFR input) to an instance of *kp_estim_qual_type*. The function s*et_kp_estim_qual()* transforms an instance of *kp_estim_qual_type* to an integer flag.

| Attribute | Bit | $2^{Bit}$ | Description |
|---|---|---|---|
| *missing* | | | Flag not set (all bits on) |
| *estim_qual* | 0 | 1 | Inferior quality of $K_p$ estimate |

**Table 4.15** $K_p$ estimate quality flag bits (Fortran).

**Wind Vector Cell quality flag**: Every WVC contains a flag for its quality. Therefore the *cell_type* contains an instance of the *wvc_quality_type*. Table 4.16 gives an overview of its attributes. The function *get_wvc_quality()* interprets an integer flag (BUFR input) to an instance of *wvc_quality_type*. The function *get_wvc_quality()* transforms an instance of *wvc_quality_type* to an integer flag. The routine *print_wvc_quality()* may be used to print the bit values of the flag.

| Attribute | Bit | $2^{Bit}$ | Description |
|---|---|---|---|
| *missing* | | | Flag not set (all bits on) |
| *qual_sigma0* | 22 | 4194304 | Not enough good $\sigma_0$ available for wind retrieval |
| *azimuth* | 21 | 2097152 | Poor azimuth diversity among $\sigma_0$ |
| *kp* | 20 | 1048576 | Any beam noise content above threshold |

| Attribute | Bit | $2^{Bit}$ | Description |
|---|---|---|---|
| *monflag* | 19 | 524288 | Product monitoring not used |
| *monvalue* | 18 | 262144 | Product monitoring flag |
| *knmi_qc* | 17 | 131072 | KNMI quality control fails |
| *var_qc* | 16 | 65536 | Variational quality control fails |
| *land* | 15 | 32768 | Some portion of wind vector cell is over land |
| *ice* | 14 | 16384 | Some portion of wind vector cell is over ice |
| *inversion* | 13 | 8192 | Wind inversion not successful |
| *large* | 12 | 4096 | Reported wind speed is greater than 30 m/s |
| *small* | 11 | 2048 | Reported wind speed is less than or equal to 3 m/s |
| *rain_fail* | 10 | 1024 | Rain flag not calculated |
| *rain_detect* | 9 | 512 | Rain detected |
| *no_background* | 8 | 256 | No meteorological background used |
| *redundant* | 7 | 128 | Data are redundant |
| *gmf_distance* | 6 | 64 | Distance to GMF too large |

**Table 4.16** Wind Vector Cell quality flag bits (Fortran).

**Cell process flag**: Besides a cell quality flag, every WVC contains a process flag. The process flag checks on aspects that are important for a proper processing, but are not available as a check in the cell quality flag. The cell process flag is set by the routine *test_cell*, which calls routines *test_time*, *test_beam* and *test_wind*.

Table 4.17 lists the attributes of the *process_flag_type*. The process flag is only available internally in AWDP. The routine *print_process_flag()* may be used to print the bit values of the flag.

| Attribute | Description |
|---|---|
| *satellite_id* | Invalid satellite id |
| *sat_instruments* | Invalid satellite instrument id |
| *sat_motion* | Invalid satellite direction of motion |
| *time* | Invalid date or time specification |
| *latlon* | Invalid latitude or longitude |
| *pixel_size_hor* | Invalid cell spacing |
| *node_nr* | Invalid across track cell number |
| *beam (3)* | Invalid data in one of the beams |
| *model_wind* | Invalid background wind |
| *ambiguity* | Invalid ambiguities |
| *selection* | Invalid wind selection |

**Table 4.17** Cell process flag bits (Fortran).

Table 4.18 provides an overview of all routines and their calls in module *awdp_data*.

| Routine | Call | Description |
|---|---|---|
| *copy_cell* | | Copy all information from one cell into another |
| *get_beam_collocation* | *init_cell* | Convert integer beam collocation to logical structure |
| *get_kp_estim_qual* | *init_beam* | Convert integer $K_p$ estimate quality to logical structure |
| *get_wvc_quality* | *init_cell* | Convert integer WVC quality to logical structure |
| *init_ambiguity* | | Initialise ambiguity structure |
| *init_beam* | *init_cell* | Initialise beam structure |
| *init_cell* | | Initialise cell structure |
| *init_full_res* | *init_cell* | Initialise full resolution structure |

| Routine | Call | Description |
|---|---|---|
| *init_icemodel* | *init_cell* | Initialise ice model structure |
| *init_nwp_stress_param* | *init_cell* | Initialise NWP stress parameters structure |
| *init_process_flag* | *init_cell* | Initialise process flag structure |
| *init_time* | *init_cell* | Initialise time structure |
| *init_wind* | *init_cell* | Initialise wind structure |
| *print_ambiguity* | | Print ambiguity structure |
| *print_beam* | | Print beam structure |
| *print_cell* | | Print cell structure |
| *print_full_res* | | Print full resolution structure |
| *print_icemodel* | | Print ice model structure |
| *print_nwp_stress_param* | | Print NWP stress parameters structure |
| *print_process_flag* | | Print process flag structure |
| *print_time* | | Print time structure |
| *print_wind* | | Print wind structure |
| *print_wvc_quality* | | Print quality flag structure |
| *set_beam_collocation* | | Convert logical beam collocation to integer |
| *set_kp_estim_qual* | | Convert logical $K_p$ estimate quality to integer |
| *set_wvc_quality* | | Convert logical WVC quality to integer |
| *test_beam* | *test_cell* | Test validity of beam data |
| *test_cell* | | Test validity of cell data |
| *test_time* | *test_cell* | Test validity of time data |
| *test_wind* | *test_cell* | Test validity of wind data |

**Table 4.18** Routines in module *awdp_data*

### 4.3.2 Module *awdp_bufr*

The module *awdp_bufr* maps the AWDP data structure on BUFR messages and vice versa. A list of the BUFR data descriptors can be found in appendix C. Satellite and instrument identifiers are listed in tables 4.19 and 4.20. Note that the first MetOp mission is MetOp 2, which is also known as MetOp A. The *awdp_bufr* module uses the genscat module *BufrMod*, see subsection 4.2.3 for the interface with the BUFR routine library.

| Satellite | Value |
|---|---|
| ERS-1 | 1 |
| ERS-2 | 2 |
| MetOp 1 = MetOp B | 3 |
| MetOp 2 = MetOp A | 4 |
| MetOp 3 = MetOp C | 5 |

**Table 4.19** BUFR satellite identifiers.

| Instrument | Parameter | Value |
|---|---|---|
| AMI/scatt | *sat_instr_ers* | 142 |
| ASCAT | *sat_instr_ascat* | 190 |

**Table 4.20** BUFR instrument identifiers.

Table 4.21 provides an overview of the different routines and their calls in this module. The genscat support routines *ymd2julian()* and *julian2ymd()* are used to provide each row in AWDP

with a date/time stamp that can be used for sorting easily.

| Routine | Call | Description |
|---|---|---|
| *ascat_bufr_to_row_data* | *read_bufr_file* | ASCAT BUFR message into one or more *row_types* |
| *ers_bufr_to_row_data* | *read_bufr_file* | ERS BUFR message into 19 *row_types* |
| *init_bufr_processing* | *read_bufr_file, write_bufr_file* | Initialise module |
| *read_bufr_file* | AWDP | Read a complete BUFR file into *row_types* |
| *row_to_bufr_data* | *write_bufr_file* | AWDP *row_type* into ASCAT BUFR message |
| *write_bufr_file* | AWDP | Write all *row_types* into a complete BUFR file |

**Table 4.21** Routines in module *awdp_bufr*

Note that the acquisition date and time of ERS data are modified when they are read in routine *ers_bufr_to_row_data*. An ERS BUFR message contains 19 rows of data which all have the same date and time of acquisition. This would cause problems in AWDP when the rows are sorted with respect to the acquisition date and time. Therefore, the date and time of each ERS row are recalculated assuming that the 10$^{th}$ (middle) row of the ERS BUFR message contains the 'true' acquisition time and that subsequent rows are 3.766 seconds apart. The time corrections are rounded to an integer number of seconds. Hence, in the first row, 34 seconds are subtracted from the acquisition time, in the second row 30 seconds, et cetera, until in the last (19$^{th}$) row, 34 seconds are added to the acquisition time.

### 4.3.3 Module *awdp_pfs*

The module *awdp_pfs* maps the records in a PFS file on the AWDP data structure. It also contains a routine to read in a full resolution PFS file and use the data to calculate averaged beam data which are used to replace 25/12.5-km row data.

Table 4.22 provides an overview of the different routines and their calls in this module. Several routines from the *pfs_ascat* module in genscat are called from this module to handle the PFS data. Appendix B5 shows the calling trees of the routines in module *pfs_ascat* that are used in AWDP.

| Routine | Call | Description |
|---|---|---|
| *ascat_pfs_to_row_data* | *read_pfs_file* | ASCAT PFS record into one *row_type* |
| *read_full_res_data* | AWDP | Read full resolution PFS data and replace beam data |
| *read_pfs_file* | AWDP | Read a complete PFS level 1b file into *row_types* |

**Table 4.22** Routines in module *awdp_pfs*

### 4.3.4 Module *awdp_prepost*

Module *awdp_prepost* contains the routines to do all the pre and post processing. Pre processing consists of the procedures between the reading of the BUFR input and the wind retrieval for the output product. This includes sorting and merging, and assessments of the quality of the input data. Post processing consists of the procedure between the ambiguity removal step and the BUFR encoding of the output. The post processing includes the monitoring of the wind data and the setting of some of the flags in the output product.

| Routine | Call | Description |
|---|---|---|
| *calibrate_s0* | *preprocess, postprocess* | Apply $\sigma_0$ calibration during the cal/val period |
| *merge_rows* | *preprocess* | Merge the data of two input rows |
| *monitoring* | *postprocess* | Monitoring |
| *postprocess* | AWDP | Main routine of the post processing |
| *pre_inversion_qc* | *preprocess* | Perform quality checks on input data |
| *preprocess* | AWDP | Main routine of the pre processing |
| *process_cleanup* | AWDP | Memory management |
| *write_binary_output* | *postprocess* | Write WVC data to a binary output file |
| *write_properties* | *postprocess* | Write some properties of the data into a text file |

**Table 4.23** Routines of module *awdp_prepost*.

Table 4.23 lists the tasks of the individual routines. AWDP calls *preprocess()* to sort the rows with respect to the acquisition data and time. It also checks on the appearance of double rows, that is, rows which are less than half the nominal cell distance (pixel size on horizontal in the input data) apart. If *preprocess()* finds a double row it merges the two rows into one row. In that case the number of input rows will be reduced. Once the input rows are sorted and merged, an output row structure is allocated ant the input data are copied into the output rows.

The routine *pre_inversion_qc()* which is called by *preprocess()* performs land flagging and checks the setting of flags in the level 1b beam information. If the input data is of inferior quality, the *qual_sigma0* flag in the *wvc_quality* is set, which prevents further processing of this WVC. Also the land fractions present in the beam information in the level 1b product are considered: if any land fraction in the fore, mid or aft beam exceeds 0.02, the *qual_sigma0* flag in *wvc_quality* is set, as well. The *land* flag in *wvc_quality* is set whenever any level 1b land fraction is above zero.

The next step is the calibration of the $\sigma_0$'s in *calibrate_s0*. This is done during the cal/val period of ASCAT. Once the level 1b products are fully calibrated, this step can be removed. Note that the calibration is done again in the reverse order after the post processing in order to write the $\sigma_0$'s to output as plain copies of the input $\sigma_0$'s. More information about the calibration can be found in [*Verspeek*, 2007].

The monitoring, which is performed as part of the post processing, calculates some statistics from the wind product and writes them to an ASCII file called *monitoring_report.txt*. The monitoring parameters are listed in table 4.24. They are calculated separately for three different regions of each swath (left and right). Note that the monitoring is invoked only if the `–mon` command line option is set.

| Parameter | Description |
|---|---|
| *observation* | Number of Wind Vector Cells in output = *N1* |
| *land* | Fraction of WVCs with land flag set |
| *ice* | Fraction of WVCs with ice flag set |
| *background* | Fraction of WVCs containing model winds |
| *backscatter_info* | Fraction of WVCs containing sufficient valid $\sigma_0$'s for inversion =*N2* |
| *knmi_flag* | Ratio number of WVCs with KNMI QC flag set / *N2* |
| *wind_retrieval* | Fraction of *N2* that actually contains wind solutions = *N3* |
| *wind_selection* | Fraction of *N3* that actually contains a wind selection = *N4* |

| Parameter | Description |
|---|---|
| *big_mle* | Number of WVCs containing a wind solution but no MLE value |
| *avg_mle* | Averaged (over *N4*) MLE value of 1st wind selection |
| *var_qc* | Fraction of *N4* that has the Variational QC flag set |
| *rank_1_skill* | Fraction of *N4* where the first wind solution is the chosen one |
| *avg_wspd_diff* | Averaged (over *N4*) difference between observed and model wind speeds |
| *rms_diff_wspd* | RMS (over *N4*) difference between observed and model wind speeds |
| *wspd_ge_4* | Fraction of *N4* where the selected wind speed is ≥ 4 m/s = *N5* |
| *rms_diff_dir* | RMS (over *N5*) difference between observed and model wind directions |
| *rms_diff_u* | RMS (over *N5*) difference between observed and model wind *u* components |
| *rms_diff_v* | RMS (over *N5*) difference between observed and model wind *v* components |
| *rms_diff_vec_len* | RMS (over *N5*) vector length between observed and model winds |
| *ambiguity* | Fraction of *N5* where the chosen solution is *not* the one closest to the model wind |

**Table 4.24** Parameters in monitoring output.

### 4.3.5 Module *awdp_grib*

The module *awdp_grib* reads in ECMWF GRIB files and collocates the model data with the scatterometer measurements. The *awdp_grib* module uses the genscat module *gribio_module*, see subsection 4.2.5 for the interface with the GRIB routine library.

Table 4.25 provides an overview of the routines and their calls in this module. The genscat support routines *uv_to_speed()* and *uv_to_dir()* are used to convert NWP wind components into wind speed and direction.

| Routine | Call | Description |
|---|---|---|
| *get_grib_data* | AWDP | Get land mask, ice mask and background winds using GRIB data |
| *init_grib_processing* | *get_grib_data* | Initialise module |

**Table 4.25** Routines in module *awdp_grib*

NWP model sea surface temperature and land-sea mask data are used to provide information about possible ice or land presence in the WVCs. WVCs with a sea surface temperature below 272.16 K (-1.0 °C) are assumed to be covered with ice and the *ice* and *qual_sigma0* flags in *wvc_quality* are set. Note that this step is omitted if the ice screening is used; see section 4.3.7.

Land presence within each WVC is determined using the land-sea mask available from the model data. The weighted mean value of the land fractions of all model grid points within 80 km of the WVC centre is calculated and if this mean value exceeds a threshold of 0.02, the *qual_sigma0* flag in *wvc_quality* is set. The *land* flag in *wvc_quality* is set if the calculated land fraction is above zero.

NWP forecast wind data are necessary in the ambiguity removal step of the processing. Wind forecasts with forecast time steps of +3h, +6h, …, +36h can be read in. The model wind data are linearly interpolated with respect to time and location and put into the *model_wind* part of each WVC.

### 4.3.6 Module *awdp_inversion*

Module *awdp_inversion* serves the inversion step in the wind retrieval. The inversion step is done

cell by cell. The actual inversion algorithm is implemented in the genscat modules *inversion* and *post_inversion*, see subsection 4.2.1. Table 4.26 provides an overview of the routines and their calls in this module.

| Routine | Call | Description |
|---|---|---|
| *init_inversion* | *invert_wvcs* | Initialisation |
| *invert_node* | *invert_wvcs* | Call to the genscat inversion routines |
| *invert_wvcs* | AWDP | Loop over all WVCs and perform inversion |

**Table 4.26** Routines of module *awpd_inversion*.

### 4.3.7 Module *awdp_ambrem*

Module *awdp_ambrem* controls the ambiguity removal step of the AWDP program. The actual ambiguity removal schemes are implemented in the genscat module *ambrem*, see section 4.2.2. The default method is the KNMI 2DVAR scheme. Table 4.27 lists the tasks of the individual routines.

| Routine | Call | Description |
|---|---|---|
| *fill_batch* | *remove_ambiguities* | Fill a batch with observations |
| *remove_ambiguities* | AWDP | Main routine of ambiguity removal |
| *select_wind* | *remove_ambiguities* | Final wind selection |

**Table 4.27** Routines of module *awpd_ambrem*.

The ambiguity removal scheme works on a so-called batch. The batch is defined in the *fill_batch()* routine. For the AWDP program a batch is just a set of rows. The size of the batch is determined by the resolution of the structure functions and the number of FFT. The genscat routine *remove_ambiguities()* performs the actual ambiguity removal. Finally *select_wind()* passes the selection to the output WVCs.

### 4.3.8 Module *awdp_icemodel*

Module *awdp_icemodel* performs the ice screening of the wind product. The ice screening works on the principle that WVCs over water yield wind solutions which are close to the GMF ('cone'). If a WVC is over ice, the $\sigma_0$ triplets from fore, mid and aft beam will be close to the so-called ice line. Hence, there is a possibility to discriminate between water (wind) and ice WVCs. The implementation of this principle is described in more detail in [*Verspeek*, 2006]. The ice screening is done directly after the ambiguity removal step. Table 4.28 provides an overview of the routines and their calls in this module.

| Routine | Call | Description |
|---|---|---|
| *calcIcelineParms* | *nonbayesianIceModel*<br>*calcIceCoord* | Calculate distance to ice line from given $\sigma_0$'s |
| *iceGMF* | not used | Calculate the $\sigma_0$ values from the ice coordinates |
| *iceLine* | *iceGMF* (not used) | Calculate the ice line origin and slope |
| *generalisedIcemodel* | not used | Instrument independent ice model |

| Routine | Call | Description |
|---|---|---|
| *n2s_n1s* | *calcIcelineParms* | Convert from two-sided to one-sided node numbering |
| *n1s_n2s* | not used | Convert from one-sided to two-sided node numbering |
| *coordTransform* | *bayesianIcemodel* | Calculate coordinates on an SSM/I grid |
| *scat2iceMap* | *bayesianIcemodel* | Update the ice map with the information in cell data |
| *iceMap2scat* | *bayesianIcemodel* | Update cell data structure with information in ice map |
| *calcIceCoord* | *bayesianIcemodel* | Calculate ice coordinates and distance to ice line |
| *bayesianIcemodel* | *awdpIcemodel* | Implementation of the Bayesian ice model |
| *nonbayesianIceModel* | *awdpIcemodel* | Implementation of the basic ice model without history |
| *awdpIcemodel* | AWDP | Main routine of ice screening |

**Table 4.28** Routines of module *awdp_icemodel*.

### 4.3.9 Module *awdp*

Module *awdp* is the main program of AWDP. It processes the command line options and controls the flow of the wind processing by calling the subroutines performing the subsequent processing steps. If any process step returns with an error code, the processing will be terminated.

# Chapter 5

# Inversion module

## 5.1 Background

In the inversion step of the wind retrieval, the radar backscatter observations in terms of the normalized radar cross-sections ($\sigma_0$'s) are converted into a set of ambiguous wind vector solutions. In fact, a Geophysical Model Function (GMF) is used to map a wind vector (specified in term of wind speed and wind direction) to the $\sigma_0$ values. The GMF further depends not only on wind speed and wind direction, but also on the measurement geometry (relative azimuth and incidence angle), and beam parameters (frequency, polarisation). A maximum likelihood estimator (MLE) is used to select a set of wind vector solutions that optimally match the observed $\sigma_0$'s. The wind vector solutions correspond to local minima of the MLE function

$$MLE = \frac{1}{N}\sum_{i=1}^{N}\frac{\left(\sigma_0^{obs}(i) - \sigma_0^{GMF}(i)\right)^2}{K_p} \quad , \tag{5.1}$$

With $N$ the number of independent $\sigma_0$ measurements available within the wind vector cell, and $K_p$ the covariance of the measurement error. Following a Bayesian approach, $K_p$ is a constant representing the noise in all three ERS or ASCAT beams together [*Stoffelen and Portabella*, 2006]. This selection depends on the number of independent $\sigma_0$ values available within the wind vector cell. The MLE can be regarded upon as the distance between an actual scatterometer measurement and the GMF in N-dimensional measurement space. The MLE is related to the probability $P$ that the GMF at a certain wind speed and direction represents the measurement by

$$P \propto e^{-MLE} \quad . \tag{5.2}$$

Therefore, wind vectors with low MLE have a high probability of being the correct solution. On the other hand, wind vectors with high MLE are not likely represented by any point on the GMF.

Details on the inversion problem can be found in [*Stoffelen and Portabella*, 2006; *Portabella,* 2002]. The AWDP program includes the Multiple Solution Scheme (MSS), see [*Portabella and Stoffelen*, 2001].

## 5.2 Routines

The inversion module class contains two modules named *inversion* and *post_inversion*. They are located in subdirectory `genscat/inversion`. Tables 5.1 and 5.2 list all routines in the modules. Appendix B.1 shows the calling tree for the inversion routines.

| Routine | Call | Routine | Call |
|---|---|---|---|
| *invert_one_wvc* | AWDP | *INTERPOLATE* | generic |
| *fill_wind_quality_code* | *invert_one_wvc* | *interpolate1d* | *calc_sigma0* |
| *save_inv_input* | not used | *interpolated2d* | *calc_sigma0* |
| *read_inv_input* | not used | *interpolate2dv* | *calc_sigma0* |
| *save_inv_output* | not used | *interpolate3d* | *calc_sigma0* |
| *do_parabolic_winddir_search* | *invert_one_wvc* | *read_LUT* | *calc_sigma0* |
| *calc_normalisation* | *invert_one_wvc* | *create_LUT_C_VV* | *calc_sigma0* |
| *calc_sign_MLE* | *invert_one_wvc* | *test_for_identical_LUTs* | *calc_sigma0* |
| *print_message* | see B.1 | *my_mod* | not used |
| *init_inv_input* | AWDP | *my_min* | see B.1 |
| *init_inv_output* | *invert_one_wvc* | *my_max* | see B.1 |
| *init_inv_settings_to_default* | AWDP | *my_average* | see B.1 |
| *write_inv_settings_to_file* | not used | *get_indices_lowest_local_minimum* | *invert_one_wvc* |
| *get_inv_settings* | AWDP | *my_index_max* | see B.1 |
| *set_inv_settings* | AWDP | *my_exit* | see B.1 |
| *check_input_data* | *invert_one_wvc* | *print_wind_quality_code* | see B.1 |
| *find_minimum_cone_dist* | *invert_one_wvc* | *print_input_data_of_inversion* | *check_input_data* |
| *get_parabolic_minimum* | *do_parabolic_winddir_search* | *print_output_data_of_inversion* | see B.1 |
| *calc_cone_distance* | *find_minimum_cone_dist* | *print_in_out_data_of_inversion* | not used |
| *calc_dist_to_cone_center* | not used | *calc_sigma0_cmod4* | *create_LUT_C_VV* |
| *convert_sigma_to_zspace* | *invert_one_wvc* | *f1* | *calc_sigma0_cmod4* |
| *get_ers_noise_estimate* | *calc_var_s0* | *Get_Br_from_Look_Up_Table* | *calc_sigma0_cmod4* |
| *calc_var_s0* | *calc_normalisation* | *calc_sigma0_cmod5* | *create_LUT_C_VV* |
| *get_dynamic_range* | not used | *calc_sigma0_cmod5_5* | *create_LUT_C_VV* |
| *get_GMF_version_used* | not used | *calc_sigma0_cmod5_n* | *create_LUT_C_VV* |
| *calc_sigma0* | see B.1 | *calc_sigma0_cmod6* | *create_LUT_C_VV* |

**Table 5.1** Routines in module *inversion*.

| Routine | Call |
|---|---|
| *normalise_conedist_ers_ascat* | AWDP |
| *calc_kp_ers_ascat* | *normalise_conedist_ers_ascat* |
| *calc_geoph_noise_ers_ascat* | *calc_kp_ers_ascat* |
| *normalise_conedist_prescat_mode* | AWDP |
| *get_ers_noise_estimate* | *normalise_conedist_prescat_mode* |
| *check_ers_ascat_inversion_data* | see B.1 |
| *check_wind_solutions_ers_ascat* | AWDP |
| *remove_one_solution* | *check_wind_solutions_ers_ascat* |
| *calc_probabilities* | AWDP |

**Table 5.2** Routines of module *post_inversion*.

To establish the MLE function (1), the radar cross section according to the GMF, $\sigma_0^{GMF}$, must be calculated. This is done in routine *calc_sigma0*. The GMF used is read as a Look Up Table (LUT) from a binary file. The value for $\sigma_0^{GMF}$ is obtained from interpolation of this table. The

interpolation is done via symbolic routine *INTERPOLATE* which is set to *interpolate1d*, *interpolate2d*, *interpolate2dv*, or *interpolate3d*, depending on the type of interpolation needed.

For C-band at VV polarization the GMF (CMODx, see [*Hersbach, Stoffen and de Haan, 2007*]) is given in analytical form (routines *calc_sigma0_cmodxxx*). If a C-band LUT is not present it will be created by routine *create_LUT_C_VV*. This routine calls one of the routines *calc_sigma0_cmodxxx* that contain the analytical expressions of the CMOD4 or CMOD5 algorithm. There is a parameter in the inversion settings type that is used to determine which CMOD function is to be used. Routines *get_lun* and *free_lun* from module *LunManager* in subdirectory `genscat/support/file` are needed when reading and creating the LUTs.

Note that module *post_inversion* uses some tables for the normalisation of MLEs and noise values. These tables are read from ASCII files which are present in direction `genscat/inversion`. The environment variable `$INVERSION_LUTSDIR` should contain the proper directory name.

## 5.3 Antenna direction

The output wind direction of inversion routines are generally given in the meteorological convention, see table 5.3. The inversion routine uses a wind direction that is relative to the antenna direction. The convention is that if the wind blows towards the antenna then this relative wind direction equals to 0. Therefore, it is important to be certain about the convention of your antenna (azimuth) angle.

For ERS and ASCAT, the radar look angle (antenna angle or simply azimuth) equals 0 if the antenna is orientated towards the south. The radar look angle increases clockwise. Therefore, the antenna angle needs a correction of 180 degrees.

| Meteorological | Oceanographic | Mathematical | *u* | *v* | Description |
|---|---|---|---|---|---|
| 0 | 180 | 270 | 0 | -1 | Wind blowing from the north |
| 90 | 270 | 180 | -1 | 0 | Wind blowing from the east |
| 180 | 0 | 90 | 0 | 1 | Wind blowing from the south |
| 270 | 90 | 0 | 1 | 0 | Wind blowing from the west |

**Table 5.3** Conventions for the wind direction.

# Chapter 6

# Ambiguity Removal module

## 6.1 Ambiguity Removal

Ambiguity Removal (AR) schemes select a surface wind vector among the different surface wind vector solutions per cell for the set of wind vector cells in consideration. The goal is to set a unique, meteorological consistent surface wind field. The surface wind vector solutions per cell, simply called ambiguities, result from the wind retrieval process step.

Whenever the ambiguities are ranked, a naive scheme would be to select the ambiguity with the first rank (e.g., the highest probability, the lowest distance to the wind cone). In general, such a persistent first rank selection will not suffice to create a realistic surface wind vector field: scatterometer measurements tend to generate ambiguous wind solutions with approximately equal likelihood (mainly due to the ~180° invariance of stand alone scatterometer measurements). Therefore additional spatial constraints and/or additional (external) information are needed to make sensible selections.

A common way to add external information to a WVC is to provide a background surface wind vector. The background wind acts as a first approximation for the expected mean wind over the cell. In general, a NWP model wind is interpolated for this purpose. Whenever a background wind is set for the WVC, a second naive Ambiguity Removal scheme is at hand: the Background Closest (BC) scheme. The selected wind vector is just the minimizer of the distance (e.g., in the least squares sense) to the background wind vector. This scheme may produce far more realistic wind vector fields than the first rank selection, since the background surface wind field is meteorologically consistent.

However, background surface winds have their own uncertainty. Therefore, sophisticated schemes for Ambiguity Removal take both the likelihood of the ambiguities and the uncertainty of the background surface wind into account. Examples are the KNMI Two-Dimensional Variational (2DVAR) scheme and the PreScat scheme.

The implementation of these schemes is described in sections 6.4 and 6.5.

## 6.2 Module *ambrem*

Module *Ambrem* is the interface module between the various ambiguity removal methods and the different scatterometer data processors. Table 6.1 provides an overview of the different routines

and their calls. A dummy method and the first rank selection method are implemented as part of *ambrem.* More elaborate Ambiguity Removal methods have an interface module, see table 6.2. Figure 6.1 shows schematically the interdependence of the various modules for Ambiguity Removal.

| Routine | Call | Description |
|---|---|---|
| *InitAmbremModule* | AWDP | Initialization of module *Ambrem* |
| *InitAmbremMethod* | AWDP | Initialization of specified AR scheme |
| *DoAmbrem* | AWDP | Execution of specified AR scheme |
| *Ambrem1stRank* | *DoAmbrem* | First rank selection method |
| *DoDummyMeth* | *DoAmbrem* | Dummy AR scheme for testing |
| *SetDummyMeth* | *DoAmbrem* | Batch definition of dummy method |
| *InitDummyMeth* | *DoAmbrem* | Initialization of dummy method |
| *InitDummyBatch* | not used | |
| *ExitAmbremMethod* | AWDP | Deallocation of memory |

**Table 6.1** Routines of module *Ambrem*.

| Routine | Description | Documentation |
|---|---|---|
| *Ambrem2DVAR* | Interface to KNMI 2DVAR method | Section 6.4 |
| *AmbremBGClosest* | Interface to Background Closest method | Section 6.1 |
| *AmbremPrescat* | Interface to Prescat method | Section 6.5 |

**Table 6.2** Interface modules for different Ambiguity Removal schemes.

## 6.3 Module *BatchMod*

After the wind retrieval step, the Ambiguity Removal step is performed on selections of the available data. In general, these selections are just a compact part of the swath or a compact part of the world ocean. The batch module *BatchMod* facilitates these selections of data. In fact, a batch data structure is introduced to create an interface between the swath related data and the data structures of the different AR methods. Consequently, the attributes of the batch data structures are a mixture of swath items and AR scheme items. Figure 6.2 gives a schematic overview of the batch data structure. Descriptions of the attributes of the individual batch data components are given in table 6.3.

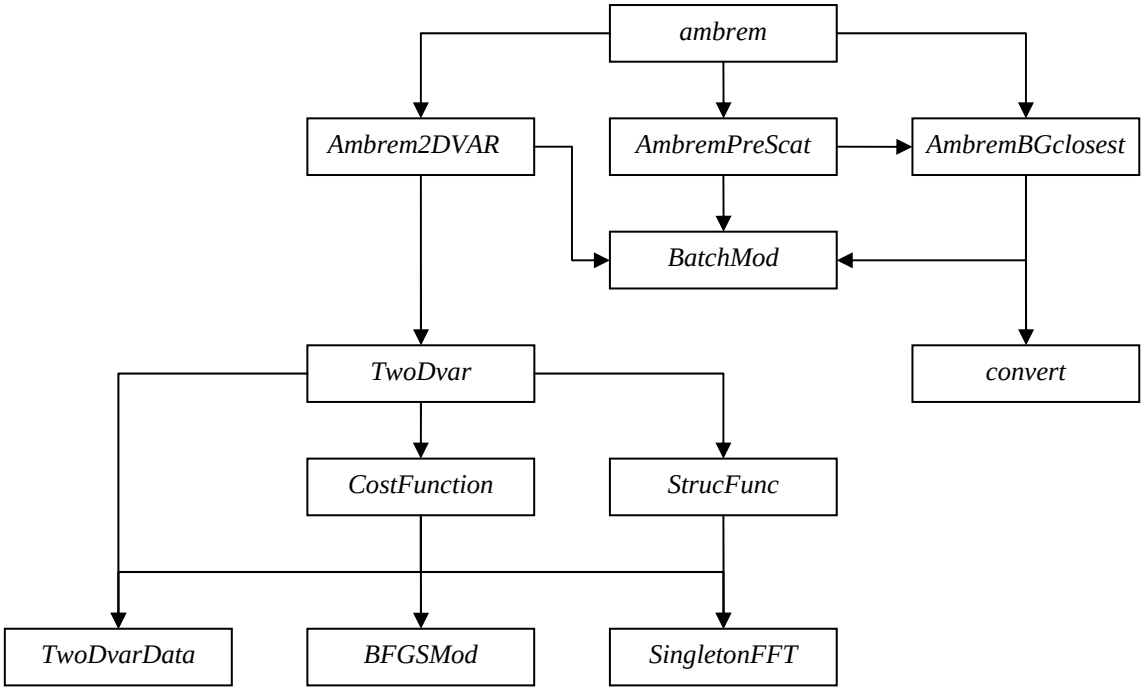


**Figure 6.1** Interdependence of the modules for Ambiguity Removal. The connections from module *ambrem* to module *BatchMod* and from module *Ambrem2DVAR* to *convert* are not drawn.

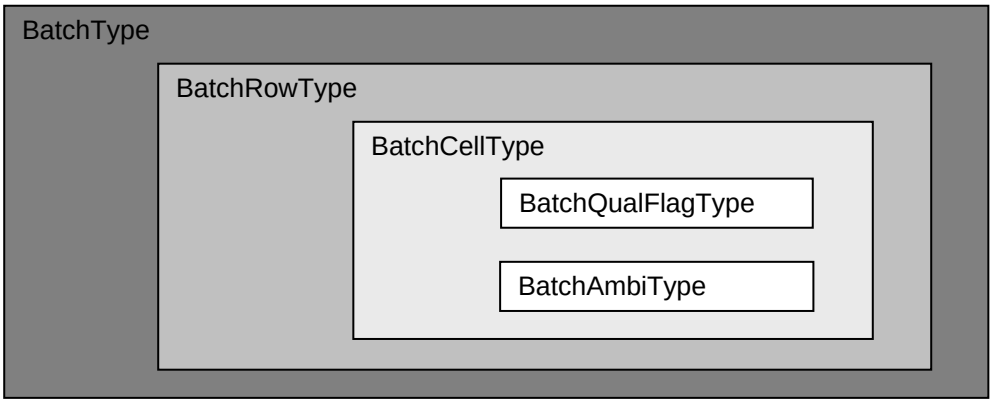


**Figure 6.2** Schematic representation of the batch data structure.

***BatchType***

| Attribute | Type | Description |
|---|---|---|
| *NrRows* | Integer | Number of rows in batch |
| *Row* | *BatchRowType* | Array of rows |

***BatchRowType***

| Attribute | Type | Description |
|---|---|---|
| *RowNr* | Integer | Row number within orbit |

| | | |
|---|---|---|
| *NrCells* | Integer | Number of cells in batch (max 76) |
| *Cell* | *BatchCellType* | Array of cells within row |

***BatchCellType***

| Attribute | Type | Description |
|---|---|---|
| *NodeNr* | Integer | Node number within orbit row |
| *lat* | Real | Latitude |
| *lon* | Real | Longitude |
| *ubg* | Real | u-component of background wind |
| *vbg* | Real | v-component of background wind |
| *NrAmbiguities* | Integer | Number of ambiguities |
| *Ambi* | *BatchAmbiType* | Array of ambiguities |

***BatchAmbiType***

| Attribute | Type | Description |
|---|---|---|
| *selection* | Integer | Index of selected ambiguity |
| *uana* | Real | u-component of analysis wind |
| *vana* | Real | v-component of analysis wind |
| *f* | Real | Contribution of this cell to cost function |
| *gu* | Real | Derivative of *f* to *u* |
| *gv* | Real | Derivative of *f* to *v* |
| *qualflag* | *BatchQualFlagType* | Quality control flag |

**Table 6.3** Batch data structures.

To check the quality of the batch a quality flag is introduced for instances of the *BatchCellType*. The flag is set by routine *TestBatchCell()*. The attributes of this flag of type *BatchQualFlagType* are listed in table 6.4.

Module *BatchMod* contains a number of routines to control the batch structure. The calls and tasks of the various routines are listed in table 6.5. The batch structure is allocatable because it is only active between the wind retrieval and the ambiguity removal step.

| Attribute | Description |
|---|---|
| *Missing* | Quality flag not set |
| *Node* | Incorrect node number specification |
| *Lat* | Incorrect latitude specification |
| *Lon* | Incorrect longitude specification |
| *Ambiguities* | Invalid ambiguities |
| *Selection* | Invalid selection indicator |
| *Background* | Incorrect background wind specification |
| *Analysis* | Incorrect analysis |
| *Threshold* | Threshold overflow |
| *Cost* | Invalid cost function value |
| *Gradient* | Invalid gradient value |

**Table 6.4** Batch quality flag attributes.

| Routine | Call | Description |
|---|---|---|
| *AllocRowsAndCellsAndInitBatch* | Processor | Allocation of batch |
| *AllocAndInitBatchRow* | *AllocRowsAndCellsAndInitBatch* | Allocation of batch rows |
| *AllocAndInitBatchCell* | *AllocAndInitBatchRow* | Allocation of batch cells |

| Routine | Call | Description |
|---|---|---|
| *AllocRowsOnlyAndInitBatch* | not used | |
| *InitBatchModule* | *Ambrem* | Initialization module |
| *InitBatch* | *AllocRowsAndCellsAndInitBatch* | Initialization of batch |
| *InitBatchRow* | *InitBatch* | Initialization of batch rows |
| *InitBatchCell* | *InitBatchRow* | Initialization of batch cells |
| *InitbatchAmbi* | *InitBatchCell* | Initialization of batch ambiguities |
| *DeallocBatch* | Processor | Deallocation of batch |
| *DeallocBatchRows* | *DeallocBatch* | Deallocation of batch rows |
| *DeallocBatchCells* | *DeallocBatchRows* | Deallocation of batch cells |
| *DeallocBatchAmbis* | *DeallocBatchCells* | Deallocation of batch ambiguities |
| *TestBatch* | Processor | Test complete batch |
| *TestBatchRow* | *TestBatch* | Test complete batch row |
| *TestBatchCell* | *TestBatchRow* | Test batch cell |
| *TestBatchQualFlag* | Processor | Print the quality flag |
| *getBatchQualFlag* | not used | |
| *setBatchQualFlag* | not used | |
| *PrnBatchQualFlag* | not used | |

**Table 6.5** Routines of module *BatchMod*.

## 6.4 The KNMI 2DVAR scheme

### 6.4.1 Introduction

The purpose of the KNMI 2DVAR scheme is to make an optimal selection provided the (modelled) likelihood of the ambiguities and the (modelled) uncertainty of the background surface wind field. First, an optimal estimated surface wind vector field (analysis) is determined based on variational principles. This is a very common method originating from the broad discipline of Data Assimilation. The optimal surface wind vector field is called the analysis. Second, the selected wind vector field (the result of the 2DVAR scheme) consists of the wind vector solutions that are closest to the analysis wind vector. For details on the KNMI 2DVAR scheme formulation the reader is referred to [*Vogelzang,* 2007]. Information on 2DVAR can also be found in [*Stoffelen, de Haan, Quilfen and Schyberg,* 2000; *de Vries, Stoffelen and Beysens,* 2005; *de Vries and Stoffelen,* 2000].

The calculation of the cost function and its gradient is rather complex matter. The reader who is only interested in how the 2DVAR scheme is assembled into the genscat module class *ambrem* is referred to subsection 6.4.2. Readers interested in the details of the cost function calculations and the minimization should also read the subsequent subsections. Subsection 6.4.3 forms an introduction to the cost function. It is recommended to first read this section, because it provides necessary background information to understand the code. Subsection 6.4.7 on the actual minimization and subsection 6.4.8 on Fast Fourier Transforms are in fact independent of the cost function itself. The reader might skip these subsections.

### 6.4.2 Data structure, interface and initialisation

The main module of the 2DVAR scheme is *TwoDvar*. Within the genscat ambiguity removal module class, the interface with the 2DVAR scheme is set by module *Ambrem2DVAR*. Table 6.6 lists its routines that serve the interface with *TwoDvar*.

| Routine | Call | Description |
|---|---|---|
| *Do2DVARonBatch* | *DoAmbrem* | Apply 2DVAR scheme on batch |
| *BatchInput2DVAR* | *Do2DVARonBatch* | Fills the 2DVAR data structure with input |
| *BatchOutput2DVAR* | *Do2DVARonBatch* | Fills the batch data structure with output |
| *Set_WVC_Orientations* | *BatchInput2DVAR* | Sets the observation orientation |
| *GetBatchSize2DVAR* | | Determine maximum size of batch |

**Table 6.6** Routines of module *Ambrem2DVAR*.

These routines are sufficient to couple the 2DVAR scheme to the processor. The actual 2DVAR processing is done by the routines of module *TwoDvar* itself. These routines are listed in table 6.7. Figures B2.1-B2.6 show the complete calling tree of the AR routines.

| Routine | Call | Description |
|---|---|---|
| *InitTwodvarModule* | | Initialization of module *TwoDvar* |
| *Do2DVAR* | *Do2DVARonBatch* | Cost function minimization |
| *PrintObs2DVAR* | *BatchInput2DVAR* | Print a single 2DVAR observation |
| *ExitTwodvarModule* | *ExitAmbremMethod* | Deallocation of module *TwoDvar* |

**Table 6.7** Routines of module *TwoDvar*.

The *Obs2dvarType* data type is the main data structure for the observed winds. Its attributes are listed in table 6.8. The *TDV_Type* data type contains all parameters that have to do with the 2DVAR batch grid: dimensions, sizes, and derived parameters. These data structures are defined in module *TwoDvarData* and the routines in this module are listed in table 6.10.

| Attribute | Type | Description |
|---|---|---|
| *alpha* | Real | Rotation angle |
| *cell* | Integer | Store batch cell number |
| *row* | Integer | Store batch row number |
| *igrid* | Integer | Row index |
| *jgrid* | Integer | Node index |
| *lat* | Real | Latitude to determine structure function |
| *Wll* | Real | Weight lower left |
| *Wlr* | Real | Weight lower right |
| *Wul* | Real | Weight upper left |
| *Wur* | Real | Weight upper right |
| *ubg* | Real | Background EW wind component |
| *vbg* | Real | Background NS wind component |
| *NrAmbiguities* | Integer | Number of ambiguities |
| *incr()* | *AmbiIncrType* | Ambiguity increments |
| *uAnaIncr* | Real | Analysis increment |
| *vAnaIncr* | Real | Analysis increment |
| *selection* | Integer | Selection flag |
| *QualFlag* | *TwoDvarQualFlagType* | Quality control flag |
| *f* | Real | Cost function at observation |
| *gu* | Real | d*f*/d*u* |
| *gv* | Real | d*f*/d*v* |

**Table 6.8** The *Obs2dvarType* data structure.

| Attribute | Type | Description |
|---|---|---|
| *delta* | Real | 2DVAR grid size in position domain |
| *delta_p* | Real | 2DVAR grid size in frequency domain |
| *delta_q* | Real | 2DVAR grid size in frequency domain |
| *N1* | Integer | Dimension 1 of 2DVAR grid |
| *H1* | Integer | *N1/2* |
| *K1* | Integer | *H1+1;*number of nonnegative frequencies |
| *N2* | Integer | Dimension 2 of 2DVAR grid |
| *H2* | Integer | *N2/2* |
| *K2* | Integer | *H2+1;*number of nonnegative frequencies |
| *Ncontrol* | Integer | Size of control vector |

**Table 6.9** The *TDV_Type* data structure.

| Routine | Call | Description |
|---|---|---|
| *TDV_Init* | *InitTwodvarModule* | Initialization of 2DVAR grid and preparations |
| *Set_HelmholzCoefficients* | *TDV_Init* | Set Helmholz transformation coefficients |
| *Set_CFW* | *TDV_Init* | Set cost function weights |
| *TDV_Exit* | *ExitTwodvarmodule* | Deallocate memory |
| *InitObs2dvar* | *BatchInput2DVAR, BatchOutput2DVAR* | Allocation of observations array |
| *DeallocObs2dvar* | *BatchOutput2DVAR* | Deallocation of observations array |
| *InitOneObs2dvar* | *InitObs2dvar* | Initialization of single observation |
| *TestObs2dvar* | *Do2DVAR* | Test single observation |
| *Prn2DVARQualFlag* | *Do2DVAR* | Print observation quality flag |
| *set2DVARQualFlag* | *TestObs2DVAR* | Convert observation quality flag to integer |
| *get2DVARQualFlag* | not used | Convert integer to observation quality flag |

**Table 6.10** Routines in module *TwoDvarData*.

The quality status of an instance of *Obs2dvarType* is indicated by the attribute *QualFlag* which is an instance of *TwoDvarQualFlagType*. The attributes of this flag are listed in table 6.11.

| Attribute | Description |
|---|---|
| *missing* | Flag values not set |
| *wrong* | Invalid 2DVAR process |
| *Lat* | Invalid latitude |
| *Background* | Invalid background wind increment |
| *Ambiguities* | Invalid ambiguity increments |
| *Selection* | Invalid selection |
| *Analyse* | Invalid analysis wind increment |
| *Cost* | Invalid cost function specification |
| *gradient* | Invalid gradient specification |
| *weights* | Invalid interpolation weights |
| *grid* | Invalid grid indices |

**Table 6.11** Attributes of 2DVAR observation quality flag.

### 6.4.3 Reformulation and transformation

The minimization problem to find the analysis surface wind field (the 2D Variational Data Assimilation problem) may be formulated as

$$\min_{v} J(v) \quad , \quad J(v) = J_{obs}(v) + J_{bg}(v), \tag{6.1}$$

where $v$ is the surface wind field in consideration and $J$ the total cost function consisting of the observational term $J_{obs}$ and the background term $J_{bg}$. The solution, the analysis surface wind field, may be denoted as $v_a$. Being just a weighted least squares term, the background term may be further specified as

$$J_{bg}(v) = [v - v_{bg}]^T B^{-1} [v - v_{bg}], \tag{6.2}$$

where $B$ is the background error covariance matrix. The $J_{obs}$ term of the 2DVAR scheme is not simply a weighted least squares term.

Such a formulation does not closely match the code of the 2DVAR scheme. In fact, for scientific and technical reasons several transformations are applied to reformulate the minimization problem. Description of these transformations is essential to understand the different procedures within the code. The interested reader is referred to [*Vogelzang* 2007].

### 6.4.4 Module *CostFunction*

Module *CostFunction* contains the main procedure for the calculation of the cost function and its gradient. It also contains the minimization procedure. Table 6.12 provides an overview of the routines.

| Routine | Call | Description |
|---|---|---|
| *Jt* | *Minimise* | Total cost function and gradient |
| *Jb* | *Jt* | Background term of cost function |
| *Jo* | *Jt* | Observational term of cost function |
| *JoScat* | *Jo* | Single observation contribution to the cost function |
| *Unpack_ControlVector* | *Jo* | Unpack of control vector |
| *Pack_ControlVector* | *Jo* | Pack of control vector (or its gradient) |
| *Uncondition* | *Jo* | Several transformations of control vector |
| *Uncondition_adj* | *Jo* | Adjoint of *Uncondition.* |
| *Minimise* | *Do2DVAR* (*TwoDvar*) | Minimization |
| *DumpAnalysisField* | *Do2DVAR* | Write analysis field to file |

**Table 6.12** Routines of module *CostFunction.*

### 6.4.5 Adjoint method

The minimization of cost function is done with a quasi-Newton method. Such a method requires an accurate approximation of the gradient of the cost function. The adjoint method is just a very economical manner to calculate this gradient. For introductory texts on the adjoint method and adjoint coding, see, e.g., [*Talagrand*, 1991; *Giering,* 1997]. For detailed information on the adjoint

model in 2DVAR see [*Vogelzang* 2007].

### 6.4.6 Structure Functions

Module *StrucFunc* contains the routines to calculate the covariance matrices for the stream function, $\psi$, and the velocity potential, $\chi$. Its routines are listed in table 6.13.

| Routine | Call | Description |
|---|---|---|
| *SetCovMat* | *Do2DVAR* | Calculate the covariance matrices |
| *InitStrucFunc* | *SetCovMat* | Initialize the structure functions |
| *StrucFuncPsi* | *SetCovMat* | Calculate $\psi$ |
| *StrucFuncChi* | *SetCovMat* | Calculate $\chi$ |

**Table 6.13** Routines of module *StrucFunc*.

Routine *InitStrucFunc* sets the structure function parameters to a default value.

### 6.4.7 Minimization

The minimization routine used is *LBFGS*. This is a quasi Newton method with a variable rank for the approximation of the Hessian written by J. Nocedal. A detailed description of this method is given by [*Liu and Nocedal* 1989]. Routine LBFGS is freeware and can be obtained from web page http://www.netlib.org/opt/index.html, file `lbfgs_um.shar`. The original Fortran 77 code has been adjusted to compile under Fortran 90 compilers. Routine LBFGS and its dependencies are located in module `BFGSMod.F90` in directory `genscat/support/BFGS`. Table 6.14 provides an overview of the routines in this module.

Routine LBFGS uses reverse communication. This means that the routine returns to the calling routine not only if the minimization process has converged or when an error has occurred, but also when a new evaluation of the function and the gradient is needed. This has the advantage that no restrictions are imposed on the form of routine *Jt* calculating the cost function and its gradient.

The formal parameters of *LBFGS* have been extended to include all work space arrays needed by the routine. The work space is allocated in the calling routine *minimise*. The rank of *LBFGS* affects the size of the work space. It has been fixed to 3 in routine *minimise*, because this value gave the best results (lowest values for the cost function at the final solution).

| Routine | Call | Description |
|---|---|---|
| *LBFGS* | *minimise* | Main routine |
| *LB1* | *LBFGS* | Printing of output (switched off) |
| *daxpy* | *LBFGS* | Sum of a vector times a constant plus another vector with loop unrolling. |
| *ddot* | *LBFGS* | Dot product of two vectors using loop unrolling. |
| *MCSRCH* | *LBFGS* | Line search routine. |
| *MCSTEP* | *MCSRCH* | Calculation of step size in line search. |

**Table 6.14** Routines in module *BFGSMod*.

Some of the error returns of the line search routine *MCSRCH* have been relaxed and are treated as a normal return. Further details can be found in the comment in the code itself.

Routines *daxpy* and *ddot* were rewritten in Fortran 90. These routines, originally written by J. Dongarra for the Linpack library, perform simple operations but are highly optimized using loop unrolling. Routine *ddot*, for instance, is faster than the equivalent Fortran 90 intrinsic function *dot_product.*

### 6.4.8 SingletonFFT_Module

Module *SingletonFFT_Module* in directory `genscat/support/singletonfft` contains the multi-variate complex Fourier routines needed in the 2DVAR scheme. A mixed-radix Fast Fourier Transform algorithm based on the work of R.C. Singleton is implemented.

| Routine | Call | Description |
|---|---|---|
| *SingletonFFT2d* | *SetCovMat, Uncondition, Uncondition_adj* | 2D Fourier transform |
| *fft* | *SingletonFFT2d* | Main FFT routine |
| *SFT_Permute* | *fft* | Permute the results |
| *SFT_PermuteSinglevariate* | *SFT_Permute* | Support routine |
| *SFT_PermuteMultivariate* | *SFT_Permute* | Support routine |
| *SFT_PrimeFactors* | *fft* | Get the factors making up *N* |
| *SFT_Base2* | *fft* | Base 2 FFT |
| *SFT_Base3* | *fft* | Base 3 FFT |
| *SFT_Base4* | *fft* | Base 4 FFT |
| *SFT_Base5* | *fft* | Base 5 FFT |
| *SFT_BaseOdd* | *fft* | General odd-base FFT |
| *SFT_Rotate* | *fft* | Apply rotation factor |

**Table 6.15** Fourier transform routines.

Table 6.15 gives an overview of the available routines. The figures in Appendix B2 shows the calling tree of the FT routines relevant for 2DVAR.

Remark: the 2DVAR implementation can be made more efficient by using a real-to-real FFT routine rather than a complex-to-complex one as implemented now. Since AWDP satisfies the requirements in terms of computational speed, this has low priority.

## 6.5 The PreScat scheme

The PreScat ambiguity removal scheme can be invoked within AWDP by the use of command line option `–armeth prescat`. More information on this scheme can be found in [*Stoffelen, de Haan, Quilfen and Schyberg,* 2000]. Currently, the PreScat scheme can be used only in combination with ERS data.

# Chapter 7

# Module *iceModelMod*

Module *iceModelMod* is part of the genscat support modules. It contains all the Bayesian statistics routines, including the routines for spatial and temporal averaging. It also contains all the routines for initialising and printing of the SSM/I grids for the North Pole and South Pole region.

## 7.1 Background

The `-icemodel` option in AWDP basically fills the fields Ice Probability (BUFR item 87) and Ice Age (BUFR item 88). Also it can output graphical maps of ice model related parameters on an SSM/I grid for the North Pole and for the South Pole region.

Each time the MetOp satellite passes over the pole region the corresponding ice map is updated with the new ASCAT data. A spatial and temporal averaging is performed in order to digest the new information. After the overpass, at the end of processing an entire BUFR file, the updated information on the ice map is put back into the BUFR structure. Optionally graphical maps are plotted, which can be controlled by optional input parameters for routine printIceMap. The graphical filenames have encoded the North Pole/South Pole, the date/time as well as the parameter name. The most important ones are:

print_a: file `[N|S][yyyymmddhhmmss].ppm` contains the ice subclass and the a-ice parameter on a grey-scale for points classified as ice.

print_t: file `[N|S][yyyymmddhhmmss]t.ppm` contains the ice class.

print_sst: file `[N|S][yyyymmddhhmmss]sst.ppm` contains the sea surface temparature

print_postprob: file `[N|S][yyyymmddhhmmss]postprob.ppm` contains the a-posteriori ice probability.

Typically at least two days of ASCAT data are needed to entirely fill the ice map with data and give meaningful ice model output. Because AWDP handles only one BUFR file at a time, a script is needed that calls AWDP several times. After each AWDP-run a binary restart file is written to disk containing the information of an icemap (`latestIceMapN.rst` for the North Pole and `latestIceMapS.rst` for the South Pole). With the next call of awdp, these restart files are read in again. Environment variable `$RESTARTDIR` contains the directory for the ice model restart files.

Optionally sea surface temperature (SST) data from GRIB files can be used to further improve the

quality of the ice algorithm (the use_sst logical must be turned on).

Processing l1b input with the use of NWP data and SST data can be done with the following command line options:

```
awdp –f <bufr file> -nwpfl <gribfilelist> -icemodel 2 –mon -handleall
```

Reprocessing of level 2 input with only running the ice model on top of it can be done with the following command line options:

```
>awdp –f <bufr file> -icemodel 2 –noinv –noamb –mon -handleall
```

The SSM/I grids are widely used for representation of ice related parameters. A good description as well as some software routines can be found on the website of the National Snow and Ice Data Centre (NSIDC): http://www.nsidc.org/data/docs/daac/ae_si25_25km_tb_and_sea_ice.gd.html.

A more detailed description of the Bayesian statistics method and ice model is given in [*Verspeek*, 2006].

## 7.2 Routines

Table 7.1 provides an overview of the routines in module *iceModelMod*.

| Routine | Call | Description |
|---|---|---|
| *calcAave* | *iceMapWeighted* | Calculate the space-time averaged ice map parameters |
| *calcPoly3* | *calcIcelineParms, iceLine* | Calculate a 3rd order polynomial |
| *calcSubClass* | *iceMapWeighted* | Calculate the ice subclass |
| *getClass* | *updateIcePixel, nonbayesianIceModel* | Get the ice class (sea or ice) |
| *printClass* | not used | Print the ice class (sea or ice) |
| *getLatest* | *iceMapWeighted, calcAave, calcSubclass, updateIcePixel, printIcePixel* | Get the indices of the latest measurement |
| *getPrevious* | *iceMapWeighted, calcAave* | Get the indices of the previous measurement |
| *GetQxGivenIce* | *updateIcePixel* | Quotient of wind probability and ice probability |
| *iceMapWeighted* | *bayesianIcemodel* | Calculate the ice a posteriori probability |
| *initIceMap* | *bayesianIcemodel* | Initialise ice map |
| *ExpandDateTime* | *iceMapWeighted, calcAave, updateIcePixel* | Converts a date/time to a real |
| *MAPLL* | *coordTransform* | Convert from lat/lon to polar stereographic coordinates |
| *MAPXY* | not used | Convert from polar stereographic to lat/lon coordinates |
| *wT* | *iceMapWeighted, calcAave* | Calculate the moving time average function |
| *logit* | not used | Calculate the logit of p: ln(p/(1-p)) |
| *inv_logit* | not used | Calculate the inverse of the logit of p: 1/(1+exp(-p)) |
| *printIcePixel* | *updateIcePixel, scat2iceMap* | Print an ice pixel |
| *RW_IceMap* | *bayesianIcemodel* | Read or write an ice map from/to a binary restart file |
| *updateIcePixel* | *scat2iceMap* | Update an ice pixel with the contents of a BUFR message. |

| Routine | Call | Description |
|---|---|---|
| *printIceMap* | *bayesianIcemodel* | Print one or more ice map variables to graphical .ppm files |
| *printIce* | *printIceMap* | Print the ice parameter a and the ice classes to a .ppm file |
| *printppmcolor* | *printIceMap* | Print variable on ice map to .ppm file, using colour index |
| *printppmvar* | *printIceMap* | Print variable on ice map to .ppm file, mapped on gray scale |

**Table 7.1** Routines of module *iceModelMod*.

## 7.3 Data structures

There are two important data structures defined in this module. The first contains all relevant data of one pixel on the ice map (IcePixel). The second one contains basically a two-dimensional array of ice pixels and represents an entire ice map (IceMapType). This could be either an ice map of the North Pole region or the South Pole region.

| Attribute | Type | Description |
|---|---|---|
| *subClass* | integer | Ice subclass |
| *aIceAves* | real | Average of the a-iceparameter |
| *aSd* | real | a-iceparameter standard deviation |
| *nIce* | integer | Number of measurement |
| *class0* | integer | Ice class |
| *Pice* | real | a-priori ice probability |
| *Qice* | real | a-priori odds on ice |
| *pIceGivenX* | real | a-posteriori ice probability |
| *qIceGivenX* | real | a-posteriori odss on ice |
| *sumWeightST* | real | Sum of weight factors |
| *sst* | real | Sea surface temperature (K) |
| *aIce* | real(nhist) | a-iceparameter |
| *qXgivenIce* | real(nhist) | Odds on measurement (X) given ice |
| *timeIce* | DateTime(nhist) | Date/time of measurement |
| *class* | integer(nhist) | Ice class |

**Table 7.2** Attributes for the *IcePixel* data type.

| Attribute | Type | Description |
|---|---|---|
| *nPixels* | integer | Number of pixels for the ice map |
| *nLines* | integer | Number of lines for the ice map |
| *nHist* | integer | Number of historical measurements that is stored (=2) |
| *pole* | integer | Indicator for Northpole or Southpole |
| *timeIceNow* | DateTime | Expanded section 3 (data description) |
| *timeIcePrev* | DateTime | Expanded section 4 (data) |
| *xy* | IcePixel(nPixels, nLines) | Pointer to the ice map contents |

**Table 7.3** Attributes for the *IceMapType* data type.

## 7.4 Parameters

There are several parameters involved that control the Bayesian statistics. They have sensible default values but most of them are made public so that their value can be overridden in the main program.

| Parameter | Description |
|---|---|
| *writeRestartFile* | Logical controlling the writing of a restartfile |
| *useWindInfo* | Logical controlling whether wind information is used |
| *useLandpoints* | Logical controlling whether an ice probability is calculated for land points |
| *use_only_windponts* | Logical controlling whether only points with a valid wind solution are used |
| *use_sst* | Logical controlling wheter sea surface temperature is used in the ice model algorithm |
| *weightS* | Matrix given the weight factors for the spatial averaging |
| *decayTime* | Defines the decay time (hours) of the temporal weighting function |
| *cutoffTime* | Defined the cutoff time (hours) for the temporal weighting function |
| *pClimateIce* | This is the ice probability when no a-priori information is available |
| *qClimateIce* | This is the odds on ice related to *pClimateIce* |
| *Class_no_data* | Class: no data |
| *Class_sea* | Class: sea (wind) |
| *Class_ice* | Class: ice |
| *Class_sea_or_ice* | Class: sea or ice (indecisive) |
| *Class_no_sea_no_ice* | Class: unknown (outlier) |
| *SubClass_a2* | SubClass: sea |
| *SubClass_b1* | SubClass: sea or ice (weight < weightSTLimit) |
| *SubClass_b2* | SubClass: probably ice (SD(a) >= aSdLimit) |
| *SubClass_b3* | SubClass: ice |
| *SubClass_d* | SubClass: unknown (outlier) |
| *SubClass_no_data* | SubClass: no data |
| *nHist* | Number of historical measurement that are stored (=2) |
| *aSdLimit* | Upper limit for the standard deviation of the a-iceparameter |
| *pIceGivenXlimit* | Lower limit of p(ice\|X) for classifying a pixel as ice |
| *sumWeightSTLimit* | Lower limit for the total weight of all measurements involved in the temperal and spatial averaging |
| *dRefIce* | Upper limit for the distance to ice line |
| *dRefWind* | Upper limit for the distance to wind cone |
| *sstLowLimit* | Lower limit for sea surface temperature (K). Below this limit pixels are classified as ice |
| *sstHighLimit* | Upper limit for sea surface temperature (K). Above this limit pixels are classified as sea (wind) |

**Table 7.6** Parameters in the Bayesian statistics.

# Chapter 8

# Module *BufrMod*

Module *BufrMod* is part of the genscat support modules. The current version is a Fortran 90 wrapper around the ECMWF BUFR library (see http://www.ecmwf.int/). The goal of this support module is to provide a comprehensive interface to BUFR data for every Fortran 90 program using it. In particular, *BufrMod* provides all the BUFR functionality required for the scatterometer processor based on genscat. Special attention has been paid to testing and error handling.

## 8.1 Background

The acronym BUFR stands for Binary Universal Form for the Representation of data. BUFR is maintained by the World Meteorological Organization WMO and other meteorological centres. In brief, the WMO FM-94 BUFR definition is a binary code designed to represent, employing a continuous binary stream, any meteorological data. It is a self defining, table driven and very flexible data representation system. It is beyond the scope of this document to describe BUFR in detail. Complete descriptions are distributed via the websites of WMO (http://www.wmo.int/) and of the European Centre for Medium-range Weather Forecasts ECMWF (http://www.ecmwf.int/).

Module *BufrMod* is in fact an interface. On the one hand it contains (temporary) definitions to set the arguments of the ECMWF library functions. On the other hand, it provides self explaining routines to be incorporated in the wider Fortran 90 program. Section 8.2 describes the routines in module *BufrMod*. The public available data structures are described in section 8.3. *BufrMod* uses two libraries: the BUFR software library of ECMWF and *bufrio*, a small library in C for file handling at the lowest level. These libraries are discussed in some more detail in section 8.4.

## 8.2 Routines

Table 8.1 provides an overview of the routines in module *BufrMod*. The most important ones are described below.

| Routine | Call | Description |
| --- | --- | --- |
| *InitAndSetNrOfSubsets* | AWDP | Initialization routine |
| *set_BUFR_fileattributes* | AWDP | Initialization routine |
| *open_BUFR_file* | AWDP | Opens a BUFR file |
| *get_BUFR_nr_of_messages* | AWDP | Inquiry of BUFR file |
| *get_BUFR_message* | AWDP | Reads instance of *BufrDataType* from file |
| *get_expected_BUFR_msg_size* | *get_BUFR_message* | Inquiry of BUFR file |
| *ExpandBufrMessage* | *get_BUFR_message* | Convert from *BufrMessageType* to *BufrSectionsType* |

| Routine | Call | Description |
|---|---|---|
| *PrintBufrErrorCode* | *ExpandBufrMessage*, *EncodeBufrData* | |
| *CheckBufrTables* | *ExpandBufrMessage* | Data check |
| *get_file_size* | *CheckBufrTables* | Determine size of BUFR file |
| *get_bufrfile_size_c* | *get_file_size* | Support routine in C |
| *encode_table_b* | *CheckBufrTables* | |
| *encode_table_d* | *CheckBufrTables* | |
| *FillBufrSecData* | *ExpandBufrMessage* | Convert from *BufrSectionsType* to *BufrDataType* |
| *close_BUFR_file* | AWDP | Closes a BUFR file |
| *BufrReal2Int* | AWDP | Type conversion |
| *BufrInt2Real* | AWDP | Type conversion |
| *save_BUFR_message* | AWDP | Saves instance of *BufrDataType* to file |
| *EncodeBufrData* | *save_BUFR_message* | Convert from *BufrSectionsType* to *BufrMessageType* |
| *CheckBufrData* | *EncodeBufrData* | Data check |
| *FillBufrData* | *EncodeBufrData* | Convert from *BufrDataType* to *BufrSectionsType* |
| *bufr_msg_is_valid* | not used | |
| *set_bufr_msg_to_invalid* | not used | |
| *PrintBufrData* | not used | |
| *GetPosBufrData* | not used | |
| *GetRealBufrData* | not used | |
| *GetIntBufrData* | not used | |
| *GetRealBufrDataArr* | not used | |
| *GetIntBufrDataArr* | not used | |
| *GetRealAllBufrDataArr* | not used | |
| *CloseBufrHelpers* | not used | |
| *missing_real* | not used | |
| *missing_int* | not used | |
| *int2real* | not used | |
| *do_range_check_int* | not used | |
| *do_range_check_real* | not used | |
| *AddRealDataToBufrMsg* | not used | |
| *AddIntDataToBufrMsg* | not used | |
| *PrintBufrModErrorCode* | not used | |
| *GetFreeUnit* | *encode_table_b*, *encode_table_d* | Get free file unit |

**Table 8.1** Routines of module *BufrMod*.

**Reading (decoding)**: Routine *get_BUFR_message()* reads a single BUFR message from the BUFR file and creates an instance of *BufrDataType*.

**Writing (encoding)**: Routine *save_BUFR_message()* saves a single BUFR message to the BUFR file. The data should be provided as an instance of *BufrDataType*.

**Checking and Printing**: The integer parameter *BufrVerbosity* controls the extent of the log statements while processing the BUFR file. The routines *PrintBufrData()* and *CheckBufrData()* can be used to respectively print and check instances of *BufrDataType*.

**Open and Close BUFR files**: The routine *open_BUFR_file()* opens the BUFR file for either reading (*writemode*=.false.) or writing (*writemode*=.true.). Routine *set_BUFR_fileattributes()* determines several aspects of the BUFR file and saves these data in an instance of *bufr_file_attr_data*, see table 8.5. Routine *get_BUFR_nr_of_messages()* is used to determine the number of BUFR messages in the file. Finally, routine *close_BUFR_file()* closes the BUFR file.

As said before, the underlying encoding and decoding routines originate from the ECMWF BUFR library. Appendix B3 shows the calling trees of the routines in module *BufrMod* that are used in AWDP.

## 8.3 Data structures

The data type closest to the actual BUFR messages in the BUFR files is the *BufrMessageType*, see table 8.2. These are still encoded data. Every BUFR message consists of 5 sections and one supplementary section. After decoding (expanding) the BUFR messages, the data are transferred into an instance of *BufrSectionsType*, see table 8.3, which contains the data and meta data in integer values subdivided in these sections.

| Attribute | Type | Description |
|---|---|---|
| *buff* | integer array | BUFR message, all sections |
| *size* | integer | Size in bytes of BUFR message |
| *nr_of_words* | integer | Idem, now size in words |

**Table 8.2** Attributes for the *BufrMessageType* data type.

| Attribute | Type | Description |
|---|---|---|
| *ksup*(9) | integer | Supplementary info and items selected from the other sections |
| *ksec*(3) | integer | Expanded section 0 (indicator) |
| *ksec1*(40) | integer | Expanded section 1 (identification) |
| *ksec2*(4096) | integer | Expanded section 2 (optional) |
| *ksec3*(4) | integer | Expanded section 3 (data description) |
| *ksec4*(2) | integer | Expanded section 4 (data) |

**Table 8.3** Attributes for the *BufrSectionsType* data type.

| Attribute | Type | Description |
|---|---|---|
| *Nsec0* | integer | ksup ( 9) dimension section 0 |
| *nsec0size* | integer | ksec0( 1) size section 0 |
| *nBufrLength* | integer | ksec0( 2) length BUFR |
| *nBufrEditionNumber* | integer | ksec0( 3) |
| *Nsec1* | integer | ksup ( 1) dimension section 1 |
| *nsec1size* | integer | ksec1( 1) size section 1 |
| *kEditionNumber* | integer | ksec1( 2) |
| *Kcenter* | integer | ksec1( 3) |
| *kUpdateNumber* | integer | ksec1( 4) |
| *kOptional* | integer | ksec1( 5) |
| *ktype* | integer | ksec1( 6) |
| *ksubtype* | integer | ksec1( 7) local use |
| *kLocalVersion* | integer | ksec1( 8) |
| *kyear* | integer | ksec1( 9) century year |
| *kmonth* | integer | ksec1(10) |
| *kday* | integer | ksec1(11) |
| *khour* | integer | ksec1(12) |
| *kminute* | integer | ksec1(13) |
| *kMasterTableNumber* | integer | ksec1(14) |

| Attribute | Type | Description |
|---|---|---|
| *kMasterTableVersion* | integer | ksec1(15) |
| *ksubcenter* | integer | ksec1(16) |
| *klocalinfo()* | integer | ksec1(17:40) |
| *Nsec2* | integer | ksup ( 2) dimension section 2 |
| *nsec2size* | integer | ksec2( 1) size section 2 |
| *key*(46) | integer | ksec2( 2: ) key |
| *Nsec3* | integer | ksup ( 3) dimension section 3 |
| *nsec3size* | integer | ksec3( 1) size section 3 |
| *Kreserved3* | integer | ksec3( 2) reserved |
| *ksubsets* | integer | ksec3( 3) number of reserved subsets |
| *kDataFlag* | integer | ksec3( 4) compressed (0,1) observed (0,1) |
| *Nsec4* | integer | ksup ( 4) dimension section 4 |
| *nsec4size* | integer | ksec4( 1) size section 4 |
| *kReserved4* | integer | ksec4( 2) reserved |
| *nelements* | integer | ksup ( 5) actual number of elements |
| *nsubsets* | integer | ksup ( 6) actual number of subsets |
| *nvals* | integer | ksup ( 7) actual number of values |
| *nbufrsize* | integer | ksup ( 8) actual size of BUFR message |
| *ktdlen* | integer | Actual number of data descriptors |
| *ktdexl* | integer | Actual number of expanded data descriptors |
| *ktdlst()* | integer array | List of data descriptors |
| *ktdexp()* | integer array | List of expanded data descriptors |
| *values()* | real array | List of values |
| *cvals()* | character array | List of CCITT IA no. 5 elements |
| *cnames()* | character array | List of expanded element names |
| *cunits()* | character array | List of expanded element units |

**Table 8.4** Attributes of the BUFR message data type *BufrDataType*.

The next step is to bring the section data to actual dimensions, descriptions and values of data which can be interpreted as physical parameters. Therefore, instances of *BufrSectionsType* are transferred to instances of *BufrDataType*, see table 8.4. The actual data for input or output in a BUFR message should be an instance of the *BufrDataType* data type. Some meta information on the BUFR file is contained in the self explaining *bufr_file_attr_data* data type, see table 8.5.

| Attribute | Type | Description |
|---|---|---|
| *nr_of_BUFR_mesasges* | integer | Number of BUFR messages |
| *bufr_filename* | character | BUFR file |
| *bufr_fileunit* | integer | Fortran unit of BUFR file |
| *file_size* | integer | Size of BUFR file |
| *file_open* | logical | Open status of BUFR file |
| *writemode* | logical | Reading or writing mode of BUFR file |
| *is_cray_blocked* | integer | Cray system blocked? |
| *list_of_BUFR_startpointers()* | integer | Pointers to BUFR messages |
| *message_is_valid()* | logical | Validity of BUFR messages |

**Table 8.5** Attributes of the *bufr_file_attr_data* data type for BUFR files.

## 8.4 Libraries

Module *BufrMod* uses two libraries: the BUFR software library of ECMWF and *bufrio*, a small

library in C for file handling at the lowest level.

The BUFR software library of ECMWF is used as a basis to encode and decode BUFR data. This software library is explained in [*Dragosavac,* 1994].

Library *bufrio* contains routines for BUFR file handling at the lowest level. Since this is quite hard to achieve in Fortran, these routines are coded in C. The routines of *bufrio* are listed in table 8.6. The source file (`bufrio.c`) is located in subdirectory `genscat/support/bufr`.

| Routine | Call | Description |
|---|---|---|
| *bufr_open* | *open_BUFR_file* | Open file |
| *bufr_split* | *open_BUFR_file* | Find position of start of messages in file |
| *bufr_read_allsections* | *get_BUFR_message* | Read *BufrMessageType* from BUFR file |
| *bufr_get_section_sizes* | *get_BUFR_message* | |
| *bufr_swap_allsections* | *get_BUFR_message, save_BUFR_message* | Optional byte swapping |
| *bufr_write_allsections* | *save_BUFR_message* | Write *BufrMessageType* to BUFR file |
| *bufr_close* | *close_BUFR_file* | |
| *bufr_error* | see appendix B.3 | Error handling |

**Table 8.6** Routines in library *bufrio*.

## 8.5 BUFR table routines

BUFR tables are used to define the data descriptors. The presence of the proper BUFR tables is checked before calling the reading and writing routines. If absent, it is tried to create the needed BUFR tables from the text version, available in genscat.

## 8.6 Centre specific modules

BUFR data descriptors are integers. These integers consist of class numbers and numbers for the described parameter itself. These numbers are arbitrary. To establish self documenting names for the BUFR data descriptors for a Fortran 90 code several centre specific modules are created. These modules are listed in table 8.7. Note that these modules are just cosmetic and not essential for the encoding or decoding of the BUFR data. They are not used in AWDP.

| Module | Description |
|---|---|
| *WmoBufrMod* | WMO standard BUFR data description |
| *KnmiBufrMod* | KNMI BUFR data description |
| *EcmwfBufrMod* | ECMWF BUFR data description |

**Table 8.7** Fortran 90 BUFR modules.

# Chapter 9

# Module *gribio_module*

Module *gribio_module* is part of the genscat support modules. The current version is a Fortran 90 wrapper around the ECMWF GRIB library (see http://www.ecmwf.int/). The goal of this support module is to provide a comprehensive interface to GRIB data for every Fortran 90 program using it. In particular, *gribio_module* provides all the GRIB functionality required for the scatterometer processor based on genscat. Special attention has been paid to testing and error handling.

## 9.1 Background

The acronym GRIB stands for GRIdded Binary. GRIB is maintained by the World Meteorological Organization WMO and other meteorological centres. In brief, the WMO FM-92 GRIB definition is a binary format for efficiently transmitting gridded meteorological data. It is beyond the scope of this document to describe GRIB in detail. Complete descriptions are distributed via the websites of WMO (http://www.wmo.int/) and of the European Centre for Medium-range Weather Forecasts ECMWF (http://www.ecmwf.int/).

Module *gribio_module* is in fact an interface. On the one hand it contains (temporary) definitions to set the arguments of the ECMWF library functions. On the other hand, it provides self explaining routines to be incorporated in the wider Fortran 90 program. Section 9.2 describes the routines in module *gribio_module*. The available data structures are described in section 9.3. The *gribio_module* uses two libraries: the GRIB software library of ECMWF and *gribio*, a small library in C for file handling at the lowest level. These libraries are discussed in some more detail in section 9.4.

## 9.2 Routines

Table 9.1 provides an overview of the routines in module *gribio_module*. The most important ones are described below.

| Routine | Call | Description |
| --- | --- | --- |
| *init_GRIB_module* | AWDP | Initialization routine |
| *dealloc_all_GRIB_messages* | AWDP | Clear all GRIB info from memory and close GRIB files |
| *set_GRIB_filelist* | AWDP | Open all necessary GRIB files |
| *get_from_GRIB_filelist* | AWDP, *get_colloc_from_GRIB_filelist* | Retrieve GRIB data for a given lat and lon |
| *inquire_GRIB_filelist* | AWDP, | Inquiry of GRIB file list |

| Routine | Call | Description |
|---|---|---|
| | *get_analyse_dates_and_times*, *get_colloc_from_GRIB_filelist* | |
| *get_colloc_from_GRIB_filelist* | AWDP | Retrieve time interpolated GRIB data for a given lat and lon |
| *get_GRIB_msgnr* | *get_field_from_GRIB_file*, *get_from_GRIB_file*, *get_from_GRIB_filelist*, *inquire_GRIB_filelist* | Inquiry of GRIB file list |
| *display_req_GRIB_msg_properties* | *get_GRIB_msgnr*, *get_from_GRIB_filelist* | Prints GRIB message info |
| *display_GRIB_message_properties* | *get_GRIB_msgnr*, *get_from_GRIB_filelist* | Prints GRIB message info |
| *get_expected_GRIB_msg_size* | *decode_sec012_gribex*, *get_GRIB_message* | Inquiry of GRIB file |
| *open_GRIB_file* | *get_field_from_GRIB_file*, *get_from_GRIB_file*, *set_GRIB_filelist*, *add_to_GRIB_filelist* | Open GRIB file and get some header information from all messages in this file |
| *decode_sec012_gribex* | *open_GRIB_file* | Decode header part of a GRIB message |
| *extract_data_from_GRIB_message* | *get_from_GRIB_file*, *get_from_GRIB_filelist* | Interpolate data from four surrounding points for a given lat and lon |
| *decode_allsec_gribex* | *get_field_from_GRIB_file*, *get_from_GRIB_file*, *get_from_GRIB_filelist* | Decode GRIB message |
| *get_GRIB_message* | *open_GRIB_file*, *get_field_from_GRIB_file*, *get_from_GRIB_file*, *get_from_GRIB_filelist* | Read GRIB message into memory |
| *dealloc_GRIB_message* | *open_GRIB_file*, *dealloc_all_GRIB_messages*, *get_field_from_GRIB_file* | Clear GRIB message from memory |
| *get_analyse_dates_and_times* | *get_colloc_from_GRIB_filelist* | Helper routine |
| *check_proximity_to_analyse* | *get_colloc_from_GRIB_filelist* | Helper routine |
| *get_field_from_GRIB_file* | not used | |
| *get_from_GRIB_file* | not used | |
| *add_to_GRIB_filelist* | not used | |

**Table 9.1** Routines of module *gribio_module*.

**Reading:** Routine set_GRIB_filelist reads GRIB messages from a list of files, decodes them and makes the data accessible in a list of GRIB messages in memory.

**Retrieving**: Routine *get_from_GRIB_filelist()* returns an interpolated value (four surrounding grid points) from the GRIB data in the list of files/messages for a given GRIB parameter, latitude and longitude. It is also possible to get a weighted value of all grid points lying within a circle around the latitude and longitude of interest. This is used in the land fraction calculation in AWDP. The land fraction is calculated by scanning all grid points of the land-sea mask lying within 80 km from the centre of the WVC. Every grid point found yields a land fraction (between 0 and 1). The land fraction of the WVC is calculated as the average of the grid land fractions, where each grid land fraction has a weight of $1/r^2$ , $r$ being the distance between the WVC centre and the model grid point.

Routine *get_colloc_from_GRIB_filelist()* returns an interpolated value (four surrounding grid

points) from the GRIB data in the list of files/messages for a given GRIB parameter, latitude, longitude, and time. The list of messages must contain a sequence of forecasts (e.g. +3 hrs, +6 hrs, +9 hrs, et cetera). At least three forecasts need to be provided; ideally two lying before the sensing time and one after.

```
----|-----|-----|----
    1     2  ^  3
```

In this diagram, the `1`, `2`, and `3` mean the three forecast steps with intervals of three hours between them. The `^` is the sensing time. The software will perform a cubic time interpolation. Note that the `1`, `2` and `3` in the diagram may correspond to +3, +6 and +9 forecasts, but also e.g. to +9, +12 and +15. If more forecasts are provided, e.g. like this:

```
----|-----|-----|-----|-----|----
    1     2     3  ^  4     5
```

the software will use forecast steps `2`, `3`, and `4`, i.e., it will pick the most usable values by itself. If one forecast before, and two after are provided:

```
----|-----|-----|----
    1  ^  2     3
```

the software will still work, and use all three forecasts.

**Checking and Printing**: The integer parameter *GribVerbosity* controls the extent of the log statements while processing the GRIB data.

As said before, the underlying encoding and decoding routines originate from the ECMWF GRIB library. Appendix B4 shows the calling trees of the routines in module *gribio_module* that are used in AWDP.

## 9.3 Data structures

Some meta information on the GRIB file is contained in the self explaining grib*_file_attr_data* data type, see table 9.2.

The data type closest to the actual GRIB messages in the GRIB files is the *grib_message_data*, see table 9.3. These are both encoded and decoded data. Every GRIB message consists of 5 sections. After decoding (expanding) the GRIB messages, the data are transferred from the *message* attribute into the *section** attributes of *grib_message_data*, which contains the data and meta data in integer values subdivided in these sections.

| Attribute | Type | Description |
|---|---|---|
| *nr_of_GRIB_messages* | integer | Number of messages in this file |
| *grib_filename* | character array | Name of GRIB file |
| *grib_fileunit* | integer | Unit number in file table |
| *file_size* | integer | Size of GRIB file in bytes |
| *file_open* | logical | Status flag |
| *list_of_GRIB_startpointers* | integer array | Starting point of message in file |
| *list_of_GRIB_level* | integer array | Key to information in messages |

| Attribute | Type | Description |
|---|---|---|
| *list_of_GRIB_level_type* | integer array | Key to information in messages |
| *list_of_GRIB_date* | integer array | Key to information in messages |
| *list_of_GRIB_hour* | integer array | Key to information in messages |
| *list_of_GRIB_analyse* | integer array | Key to information in messages |
| *list_of_GRIB_derived_date* | integer array | Key to information in messages |
| *list_of_GRIB_derived_hour* | integer array | Key to information in messages |
| *list_of_GRIB_par_id* | integer array | Key to information in messages |
| *list_of_GRIB_sec4_sizes* | integer array | Size of section 4 |

**Table 9.2** Attributes for the *grib_file_attr_data* data type.

| Attribute | Type | Description |
|---|---|---|
| *date* | real | Date when data are valid |
| *time* | real | Time when data are valid |
| *derived_date* | real | date + time/24 |
| *derived_time* | real | mod(time/24) |
| *total_message_size* | integer | Size of message |
| *sec_4_size* | integer | Size of section 4 |
| *in_memory* | logical | Status flag |
| *is_decoded* | logical | Status flag |
| *contains_real_data* | logical | Flag, set if message contains real data |
| *nr_lon_points* | integer | Information about grid |
| *nr_lat_points* | integer | Information about grid |
| *nr_grid_points* | integer | Information about grid |
| *lat_of_first_gridpoint* | real | Information about grid |
| *lat_of_last_gridpoint* | real | Information about grid |
| *lon_of_first_gridpoint* | real | Information about grid |
| *lon_of_last_gridpoint* | real | Information about grid |
| *lat_step* | real | Information about grid |
| *lon_step* | real | Information about grid |
| *message* | character array, pointer | Raw GRIB data |
| *section0* | integer array | Expanded section 0 (data) |
| *section1* | integer array | Expanded section 1 (data) |
| *section2* | integer array | Expanded section 2 (data) |
| *section3* | integer array | Expanded section 3 (data) |
| *section4* | integer array | Expanded section 4 (data) |
| *real_section2* | real array | Expanded real data in section 2 (data) |
| *real_section3* | real array | Expanded real data in section 3 (data) |
| *real_section4* | real array, pointer | Expanded real data in section 4 (data) |

**Table 9.3** Attributes for the *grib_message_data* data type.

| Attribute | Type | Description |
|---|---|---|
| *which_grib_message* | integer | Position of message in file |
| *grib_message* | grib_message_data | GRIB message |

**Table 9.4** Attributes of the GRIB message data type *list_of_GRIB_msgs_type*.

| Attribute | Type | Description |
|---|---|---|

| Attribute | Type | Description |
|---|---|---|
| *filename* | character(256) | Name of GRIB file |
| *is_open* | logical | Status flag |
| *list_of_GRIB_msgs* | list_of_GRIB_msgs_type array | List of messages in file |

**Table 9.5** Attributes of the *list_of_grib_files_type* data type for GRIB files.

## 9.4 Libraries

Module *gribio_module* uses two libraries: the GRIB software library of ECMWF and *gribio*, a small library in C for file handling at the lowest level.

The GRIB software library of ECMWF is used as a basis to decode GRIB data. This software library is explained on http://www.ecmwf.int/.

Library *gribio* contains routines for GRIB file handling at the lowest level. Since this is quite hard to achieve in Fortran, these routines are coded in C. The routines of *gribio* are listed in table 9.6. The source file (`gribio.c`) is located in subdirectory `genscat/support/grib`.

| Routine | Call | Description |
|---|---|---|
| *grib_open* | *open_GRIB_file* | Open file |
| *grib_split* | *open_GRIB_file* | Find position of start of messages in file |
| *grib_read_allsections* | *get_GRIB_message* | Read *grib_message_data* from GRIB file |
| *grib_swap_allsections* | *get_GRIB_message* | Optional byte swapping |
| *grib_close* | *dealloc_all_GRIB_messages*, *get_field_from_GRIB_file* | Close file |
| *grib_error* | See appendix B.4 | Error handling |

**Table 9.6** Routines in library *gribio*.

# References


- Dragosavac, M., 1994,
  *BUFR User Guide and Reference Manual*. ECMWF. (Available on http://www.ecmwf.int/)
- Figa-Saldaña, J., and Wilson, J.J.W., 2005,
  *ASCAT Level 1 Product Format Specification*, Issue 6, Rev 5, EUMETSAT, EPS.MIS.SPE.97233 (Available on http://www.eumetsat.int/).
- Giering, R., 1997,
  *Tangent linear and Adjoint Model Compiler, Users manual*. Max-Planck- Institut fuer Meteorologie.
- Hersbach, H., Stoffelen, A. and de Haan, S., 2007,
  *An improved C-band scatterometer ocean geophysical model function: CMOD5*, Journal of Geophysical Research, **112**.
- Liu, D.C., and Nocedal, J., 1989
  *On the limited memory BFGS method for large scale optimization methods*. Mathematical Programming, 45, pp. 503-528.
- UK Met Office, 2001
  *ERS Products WMO FM94 BUFR Format*, ER-IS-UKM-GS-0001, Version 4, Issue 2.
- Portabella, M., 2002,
  *Wind field retrieval from satellite radar systems*, PhD thesis, University of Barcelona. (Available on http://www.knmi.nl/scatterometer/publications/).
- Portabella, M. and Stoffelen, A., 2001,
  *Rain Detection and Quality Control of SeaWinds*, Journal of Atm. Oceanic Technol., **18**, pp. 1171-1183.
- Portabella, M. and Stoffelen, A., 2004,
  *A probabilistic approach for SeaWinds Data Assimilation*, Quart. J. Royal Meteor. Soc., **130**, pp. 127-152.
- Stoffelen, A. and M. Portabella, 2006,
  *On Bayesian Scatterometer Wind Inversion*, IEEE Transactions on Geoscience and Remote Sensing, 44, 6, 1523-1533, doi:10.1109/TGRS.2005.862502.
- Stoffelen, A., de Haan, S., Quilfen, Y., and Schyberg, H., 2000,
  *ERS scatterometer ambiguity removal scheme comparison*, OSI SAF report. (Available on, http://www.knmi.nl/scatterometer/publications/).
- Stoffelen, A.C.M., 1998,

*Scatterometry*, PhD thesis, University of Utrecht, ISBN 90-393-1708-9. (Available on http://www.knmi.nl/scatterometer/publications/).

- Talagrand, O., 1991,
  *The use of adjoint equations in numerical modeling of the atmospheric circulation*. In: Automatic Differentiation of Algorithms: Theory, Implementation and Application, A. Griewank and G. Corliess Eds. pp. 169-180, Philadelphia, Penn: SIAM.
- Verhoef, A., Vogelzang, J., Verspeek, J. and Stoffelen, A., 2008,
  *AWDP Test Report*, Report NWPSAF-KN-TV-005, UKMO, UK.
- Verspeek, J., 2006,
  *Sea ice classification using Bayesian statistics*, OSI SAF report
- Verspeek, J., Portabella, M., Stoffelen, A. and Verhoef, A., 2007,
  *ASCAT Calibration and Validation*, Report SAF/OSI/CDOP/KNMI/TEC/TN/163, OSI SAF (Available on http://www.knmi.nl/scatterometer/publications/)
- Vogelzang, J., 2007,
  *Two dimensional variational ambiguity removal (2DVAR).* Report NWPSAF-KN-TR-004, UKMO, UK. (Available on http://www.knmi.nl/scatterometer/publications/).
- Vogelzang, J., Stoffelen, A., Verhoef, A., de Vries, J. and Bonekamp, H., 2008,
  *Validation of two-dimensional variational ambiguity removal on SeaWinds scatterometer data*, submitted to J. Atm. Oceanic Technol.
- de Vries, J. and Stoffelen, A., 2000,
  *2D Variational Ambiguity Removal*. KNMI, Feb 2000. (Available on http://www.knmi.nl/scatterometer/publications/).
- de Vries, J., Stoffelen, A., and Beysens, J., 2005,
  *Ambiguity Removal and Product Monitoring for SeaWinds*. KNMI. (Available on http://www.knmi.nl/scatterometer/publications/).
- Wilson, J.J.W, Figa-Saldaña, J., and O'Clerigh, E., 2004,
  *ASCAT Product Generation Function Specification*, Issue 6, Rev 5, EUMETSAT, EUM.EPS.SYS.SPE.990009 (Available on http://www.eumetsat.int/).
- WMO, 2007,
  *Additions to BUFR/CREX Tables for pre-operational implementation endorsed by CBS for full operational status on 7 November 2007* (updated 04/01/07), pages 55-60 (available on http://www.wmo.int/web/www/WMOCodes/Updates/BUFRCREX/Preoperational050107.doc)

# Appendix A

# Calling tree for AWDP

The figures in this appendix show the calling tree for the AWDP program. Routines in white boxes are part of the AWDP process layer. Routines in black boxes are part of genscat. An arrow (→) before a routine name indicates that this part of the calling tree is a continuation of a branch in a previous figure. The same arrow after a routine name indicated that this branch will be continued in a following figure.

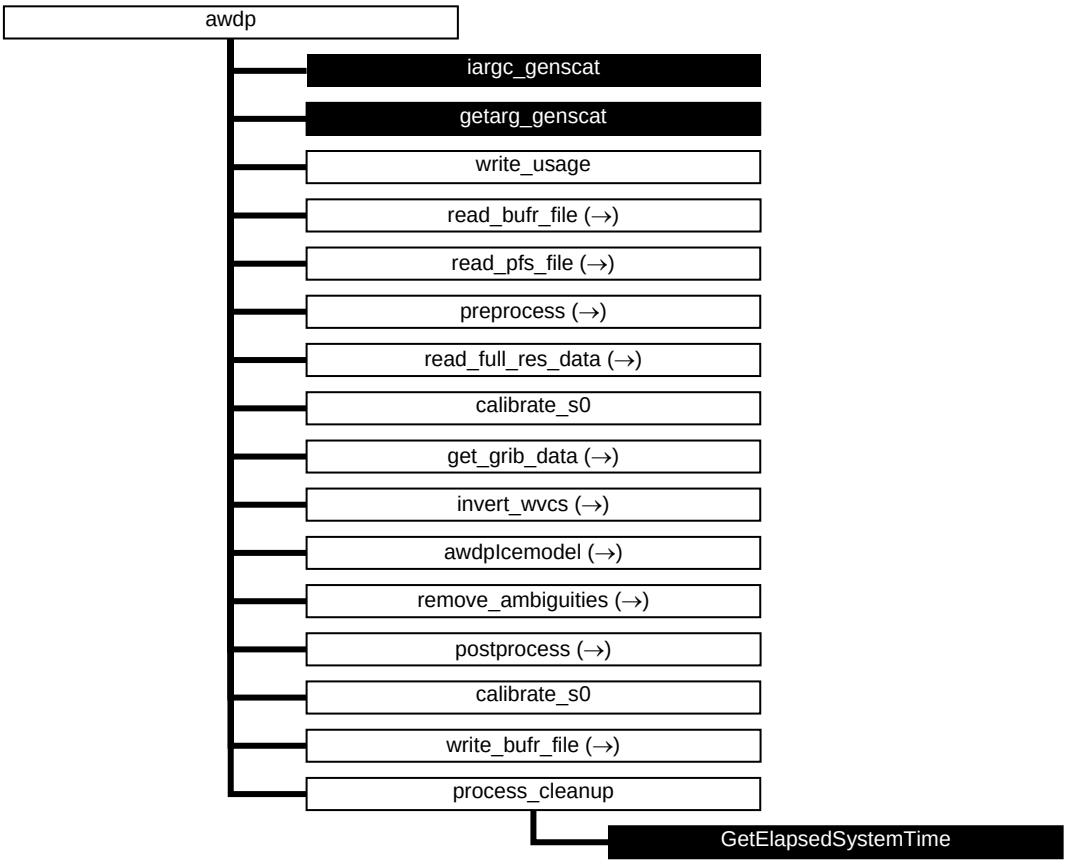


**Figure A.1** Calling tree for program *awdp* (top level). White boxes are cut here and will be continued in one of the first level or second level calling trees in the next figures. Black boxes with light text indicate genscat routines.

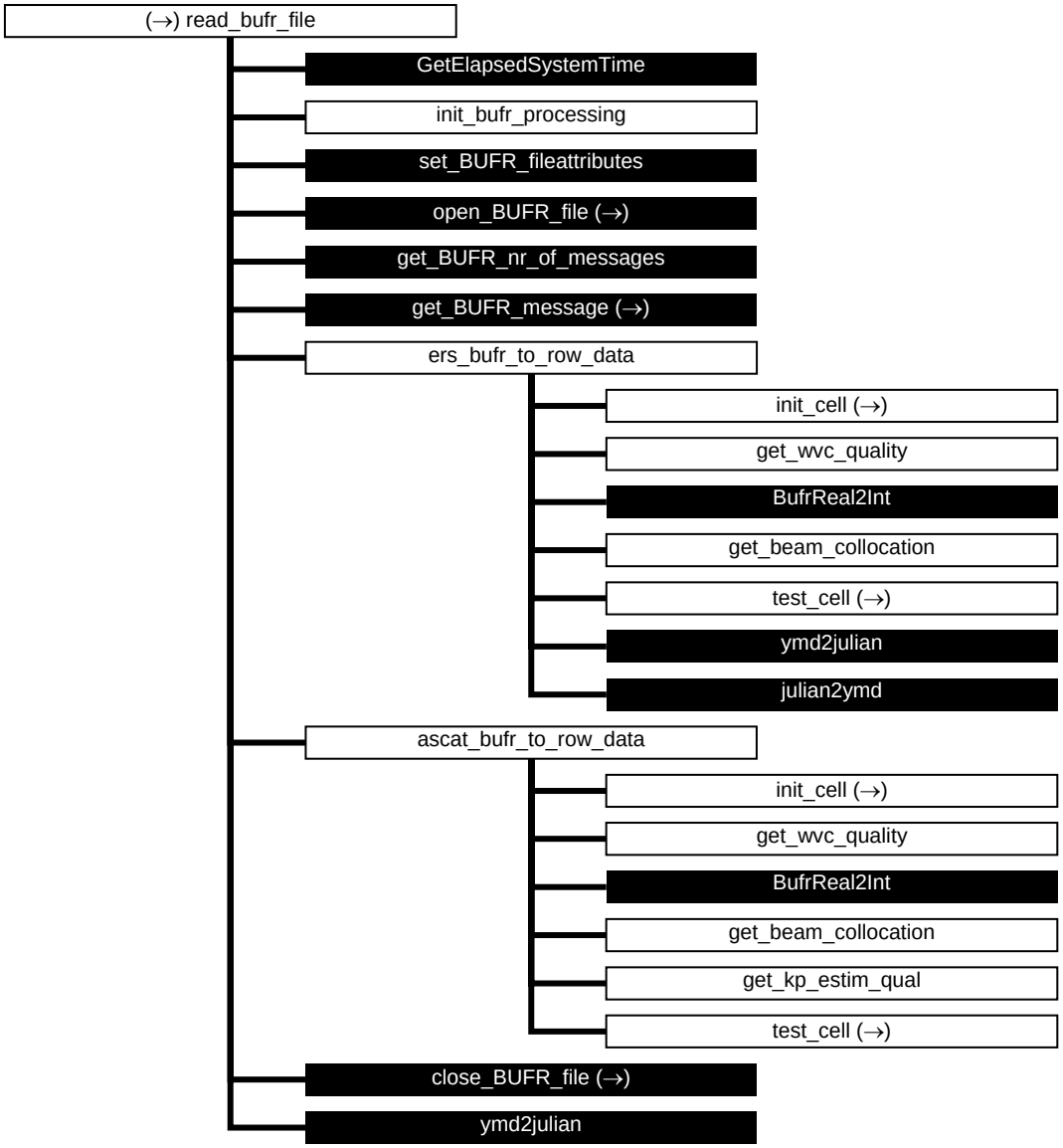


**Figure A.2** Calling tree for routine *read_bufr_file* (first level).

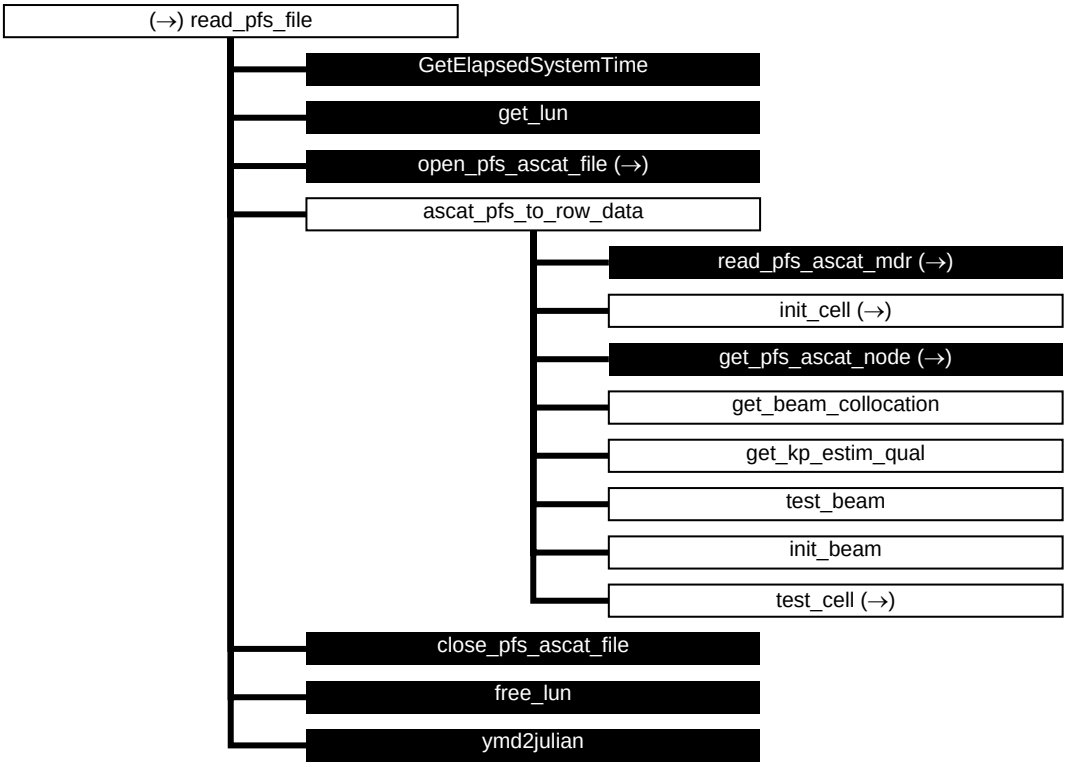


**Figure A.3** Calling tree for routine *read_pfs_file* (first level).

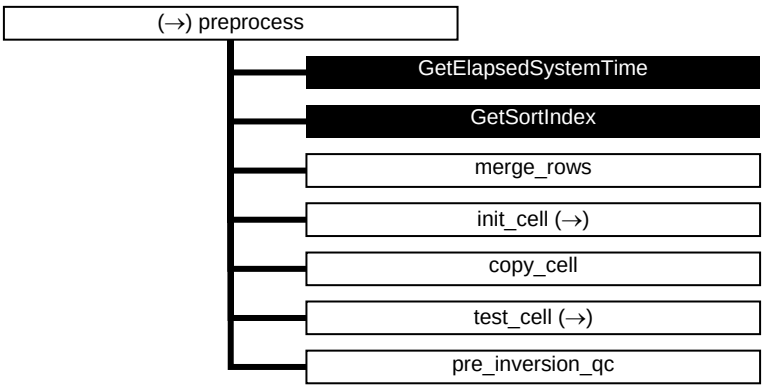


**Figure A.4** Calling tree for routine *preprocess* (first level).

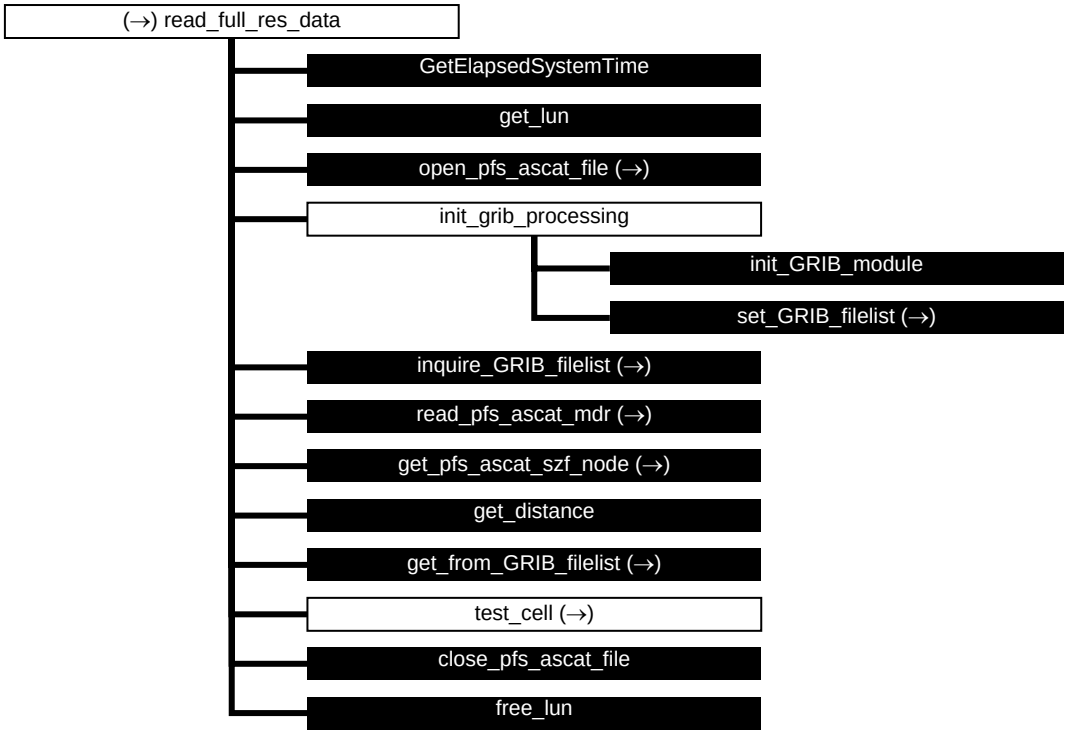


**Figure A.5** Calling tree for routine *read_full_res_data* (first level).

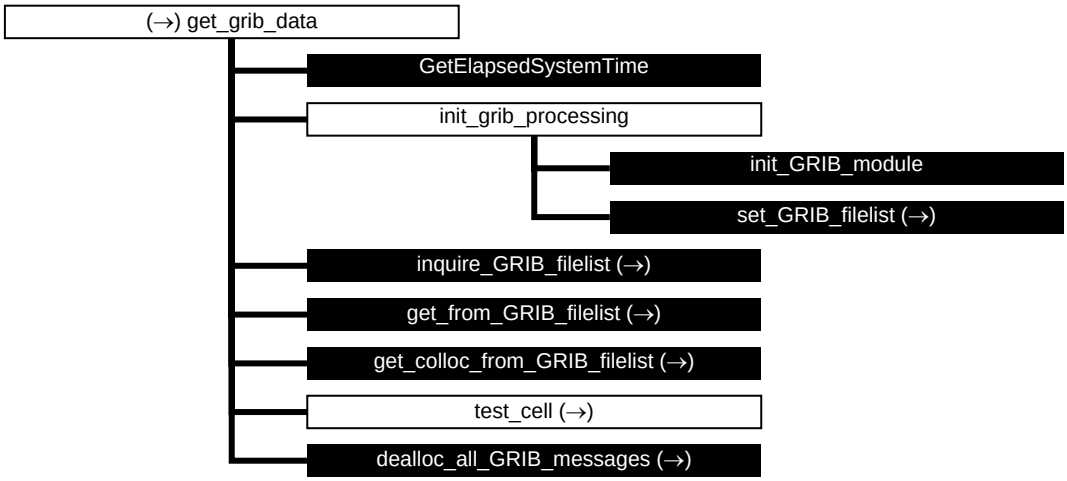


**Figure A.6** Calling tree for routine *get_grib_data* (first level).

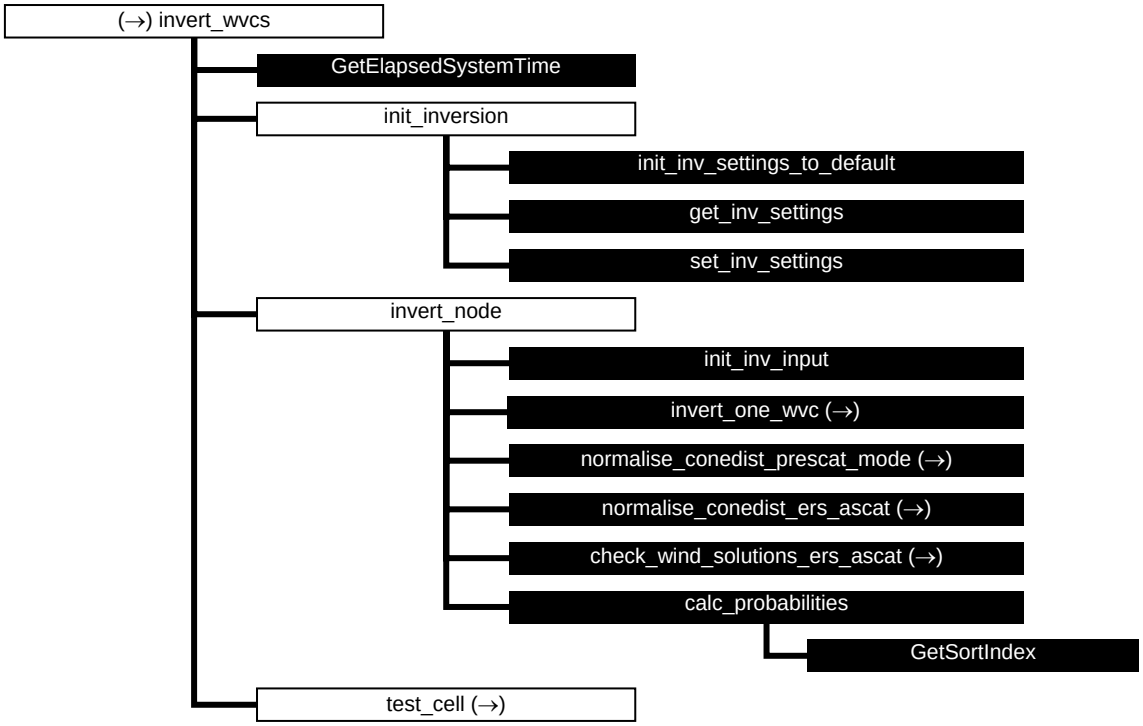


**Figure A.7** Calling tree for routine *invert_wvcs* (first level).

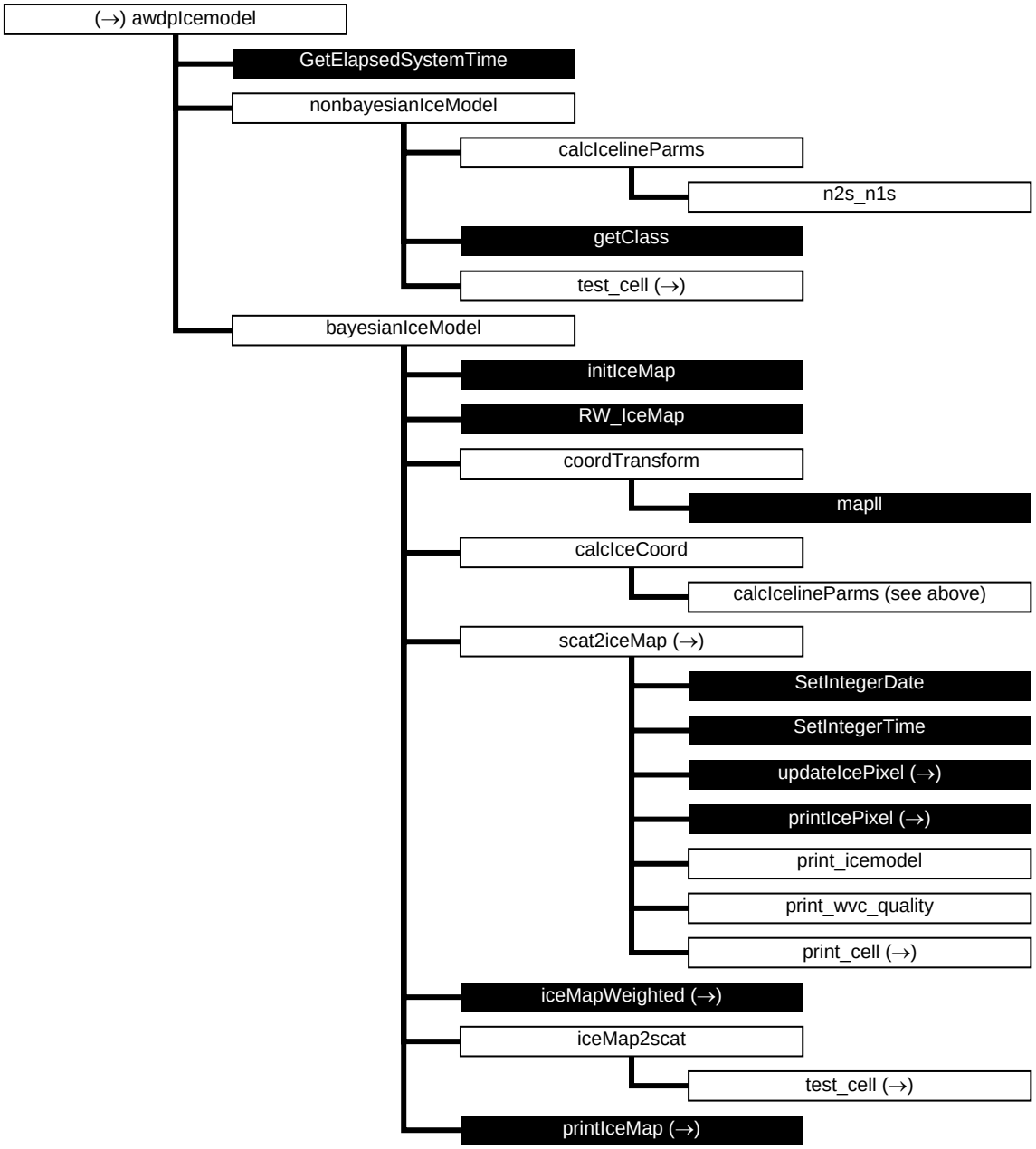


**Figure A.8** Calling tree for routine *awdpIcemodel* (first level).

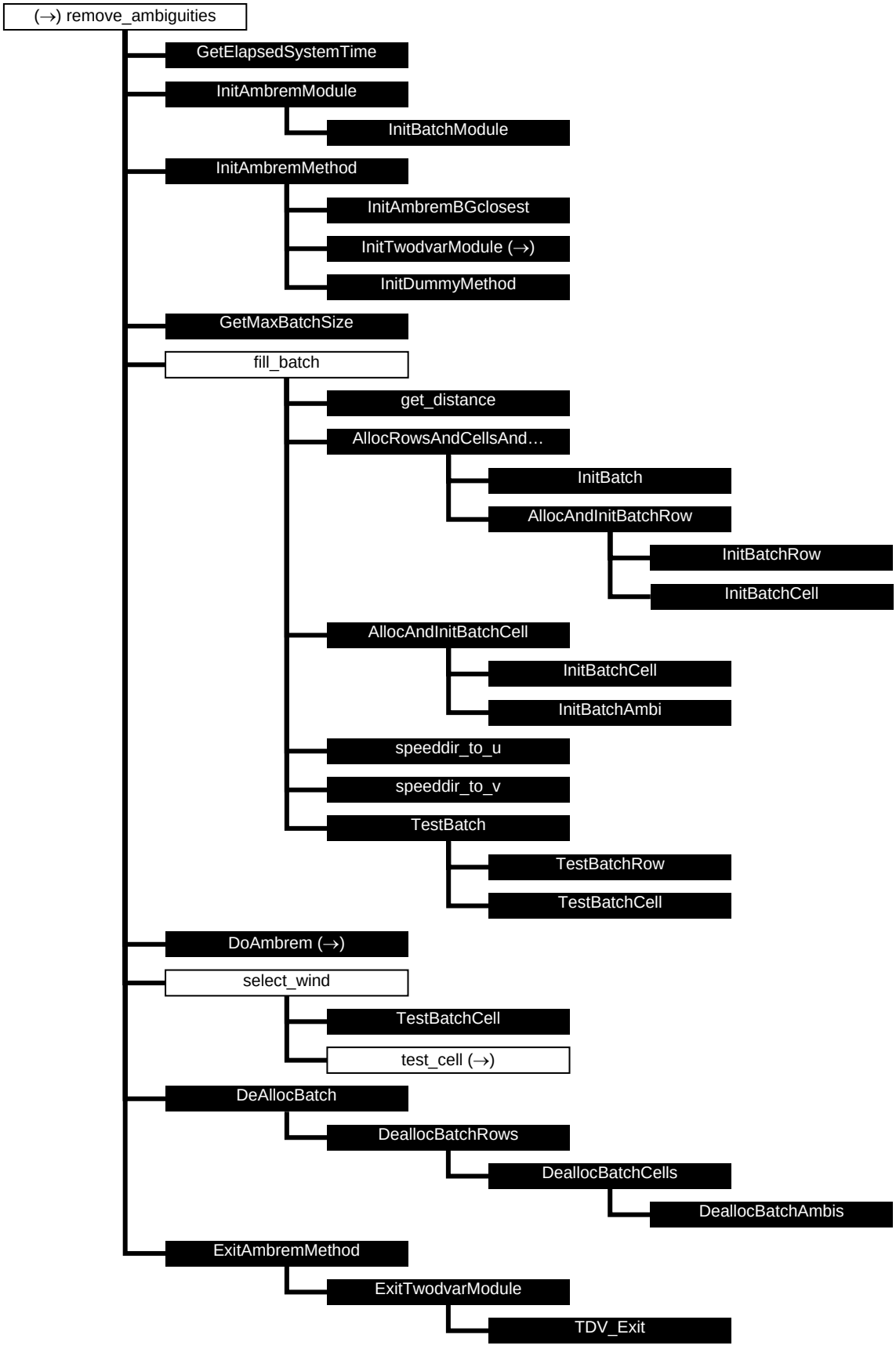


**Figure A.9** Calling tree for routine *remove_ambiguities* (first level). The full name of the 12th routine is *AllocRowsAndCellsAndInitBatch*.

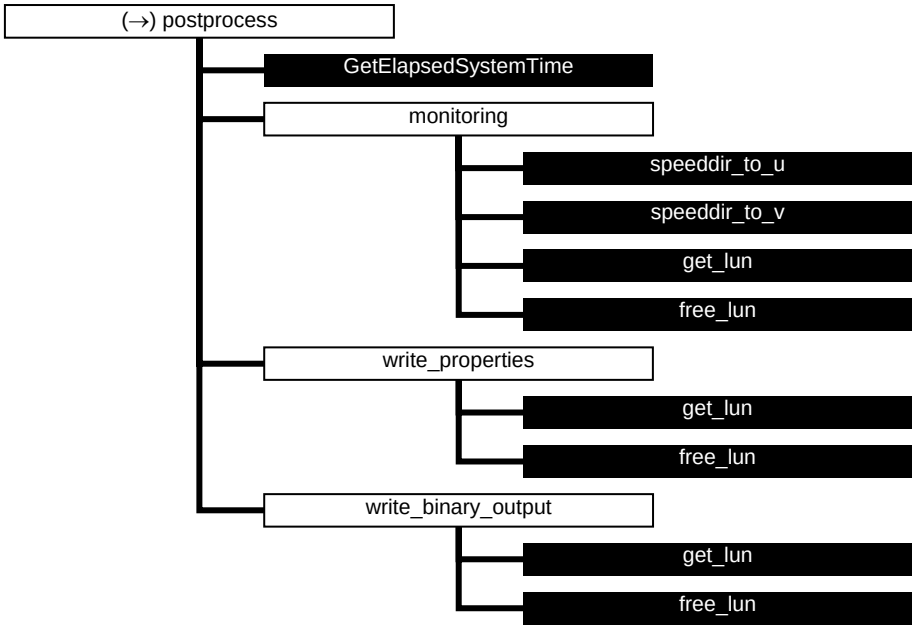


**Figure A.10** Calling tree for routine *postprocess* (first level).

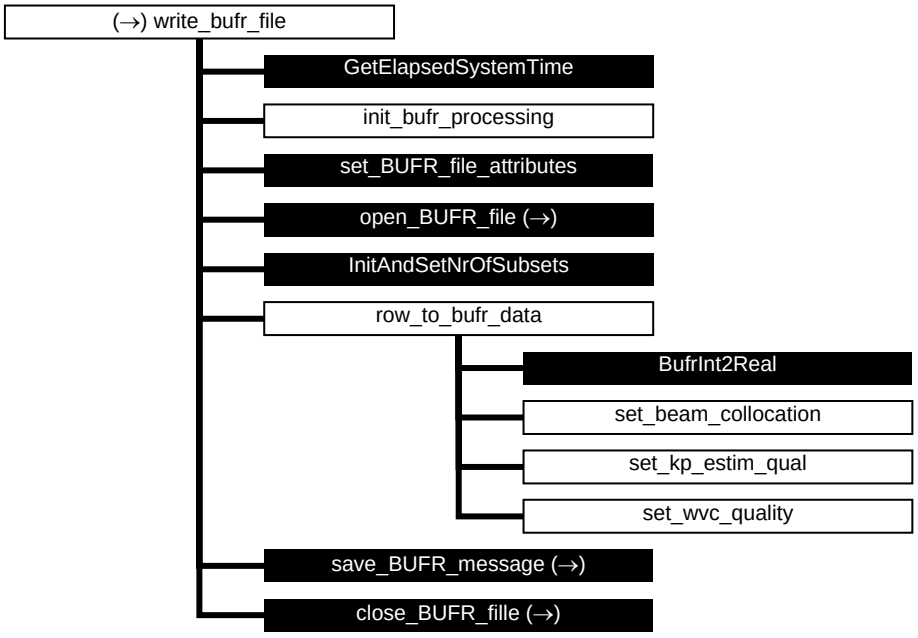


**Figure A.11** Calling tree for routine write_bufr_file (first level).

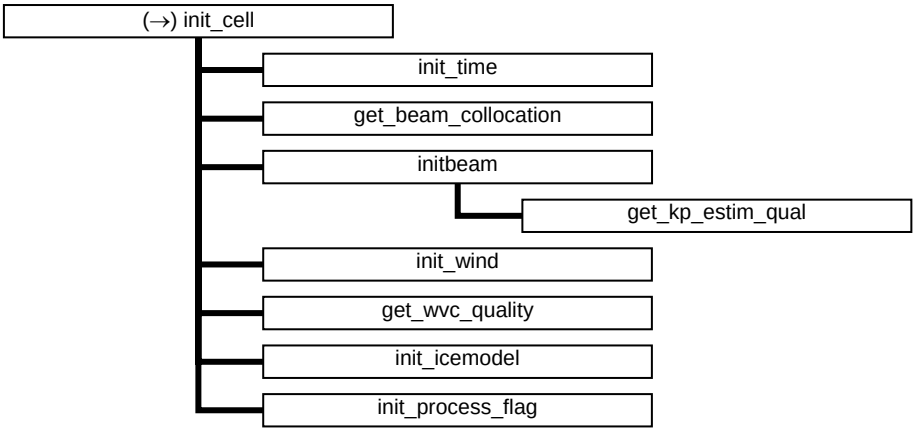


**Figure A.12** Calling tree for routine *init_cell* (second level).

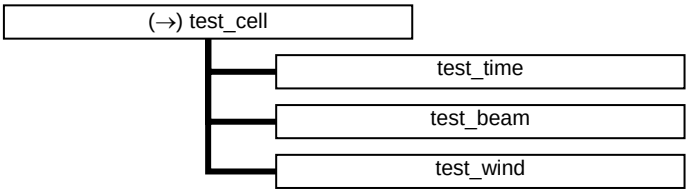


**Figure A.13** Calling tree for routine *test_cell* (second level).

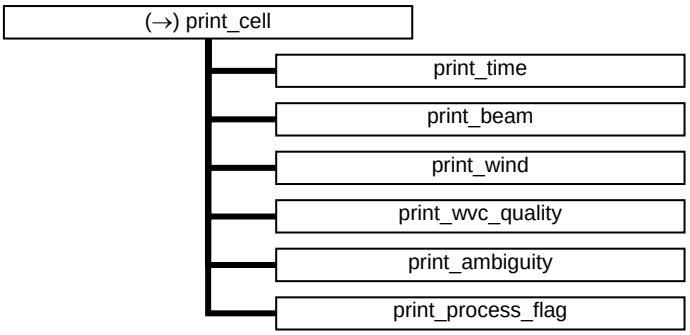


**Figure A.14** Calling tree for routine *PrintCell* (second level).

# Appendix B1

# Calling tree for inversion routines

The figures in this appendix show the calling tree for the inversion routines in genscat. All routines are part of genscat, as indicated by the black boxes. An arrow (→) before a routine name indicates that this part of the calling tree is a continuation of a branch in a previous figure. The same arrow after a routine name indicates that this branch will be continued in a following figure.

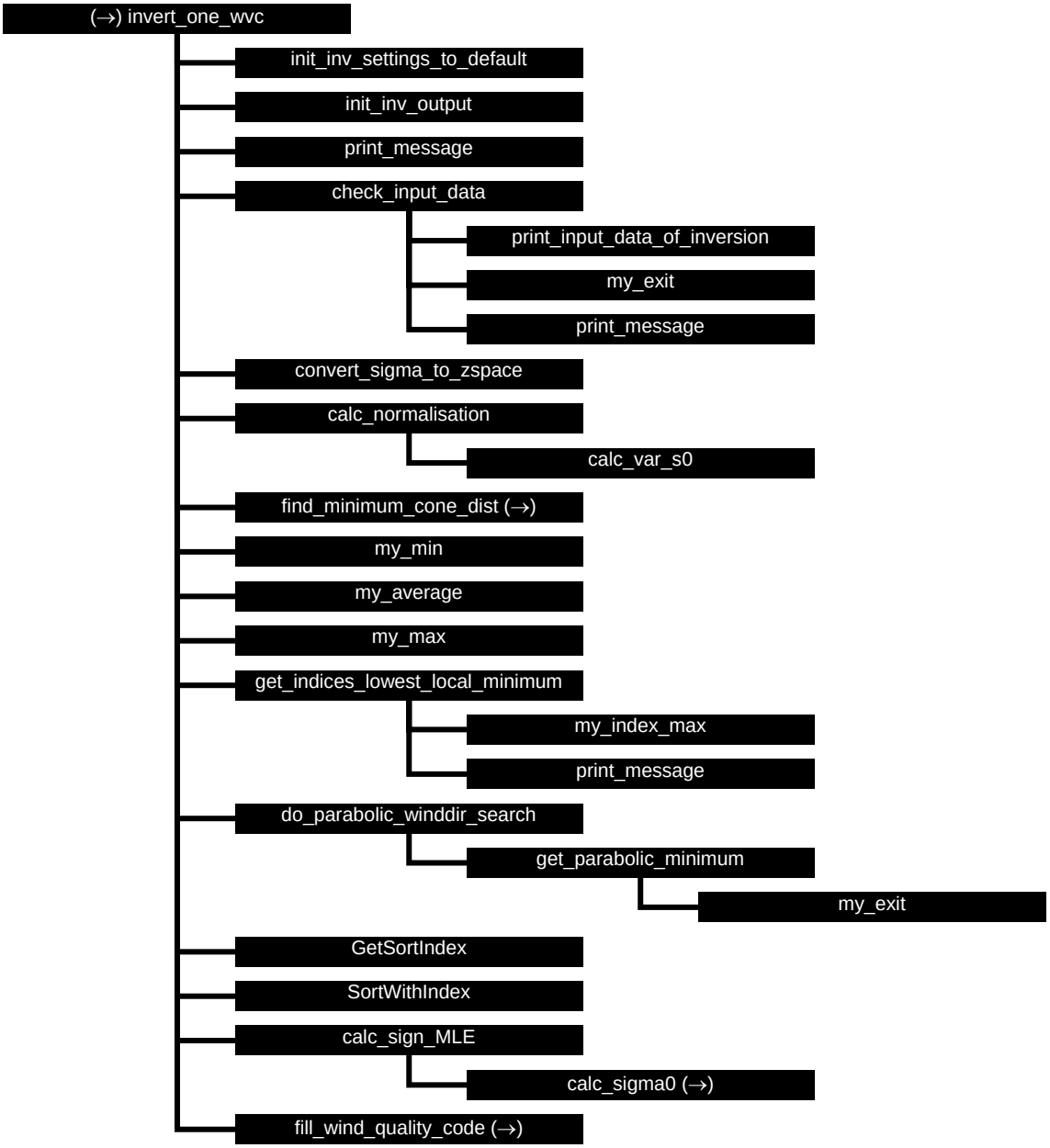


**Figure B1.1** Calling tree for inversion routine *invert_one_wvc*.

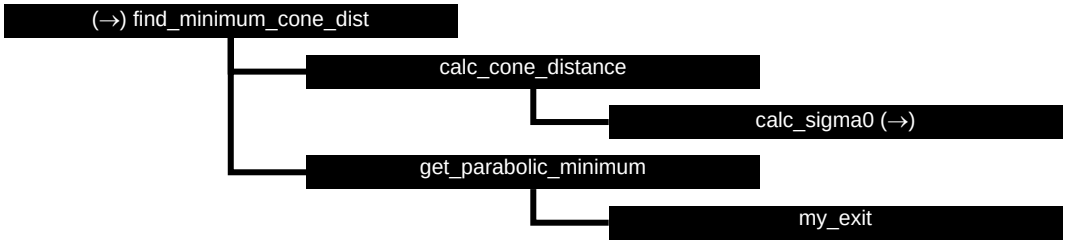


**Figure B1.2** Calling tree for inversion routine *find_minimum_cone_dist*

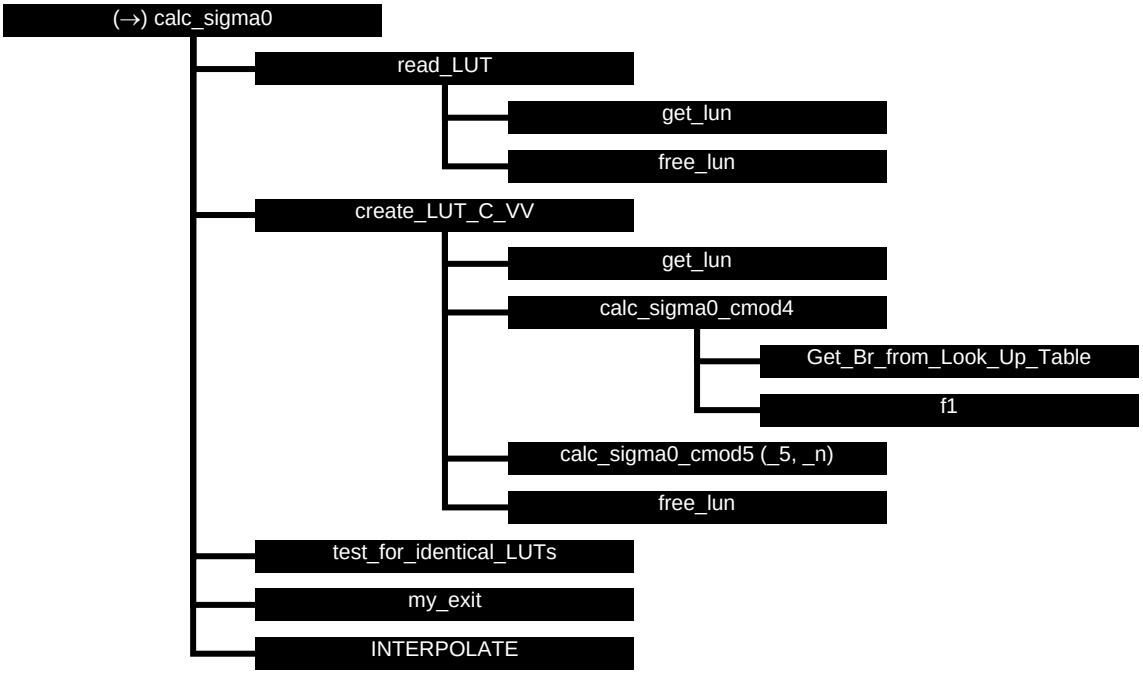


**Figure B1.3** Calling tree for inversion routine *calc_sigma0*. Routine *INTERPOLATE* is an interface that can have the values *interpolate1d*, *interpolate2d*, *interpolate2dv* or *interpolate3d*. There are several equivalent routines to calculate the CMOD backscatter, like *calc_sigma0_cmod5*, *calc_sigma0_cmod5_5*, *calc_sigma0_cmod5_n*.

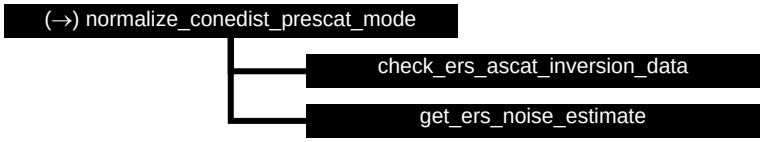


**Figure B1.4** Calling tree for inversion routine *normalize_conedist_prescat_mode*.

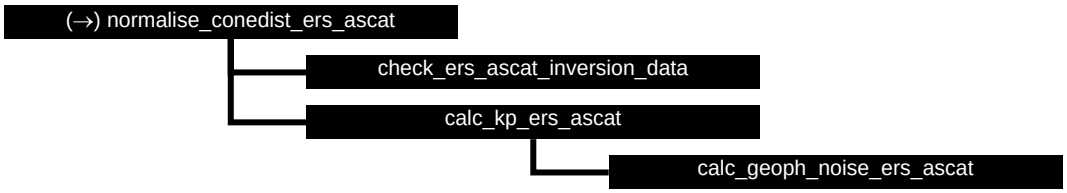


**Figure B1.5** Calling tree for inversion routine *normalize_conedist_ers_ascat*.

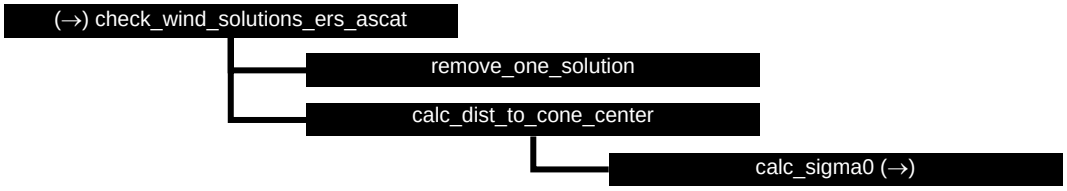


**Figure B1.6** Calling tree for inversion routine *check_wind_solutions_ers_ascat*.

# Appendix B2

# Calling tree for AR routines

The figures in this appendix show the calling tree for the Ambiguity Removal routines in genscat. All routines are part of genscat, as indicated by the black boxes. An arrow (→) before a routine name indicates that this part of the calling tree is a continuation of a branch in a previous figure. The same arrow after a routine name indicates that this branch will be continued in a following figure.

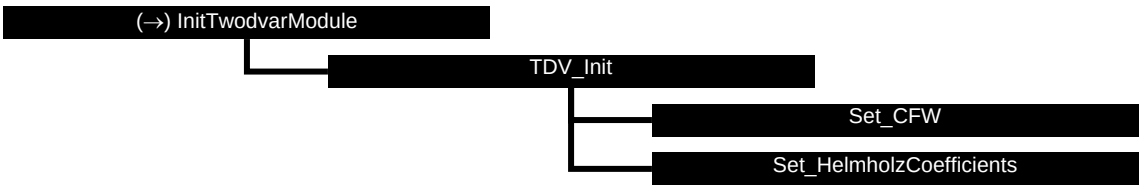


**Figure B2.1** Calling tree for AR routine *InitTwodvarModule*.

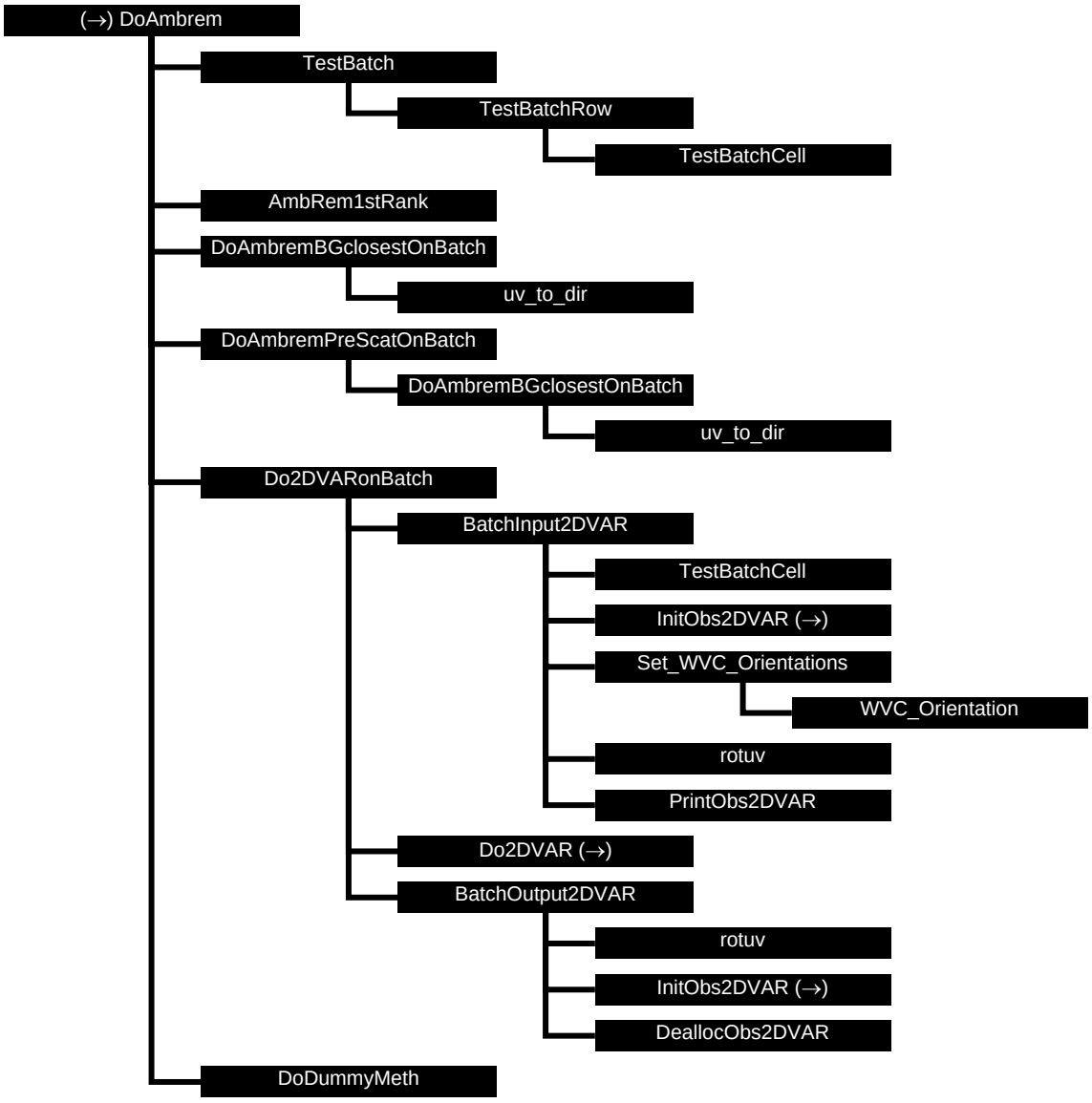


**Figure B2.2** Calling tree for AR routine *DoAmbrem*.


**Figure B2.3** Calling tree for AR routine *InitObs2dvar*.

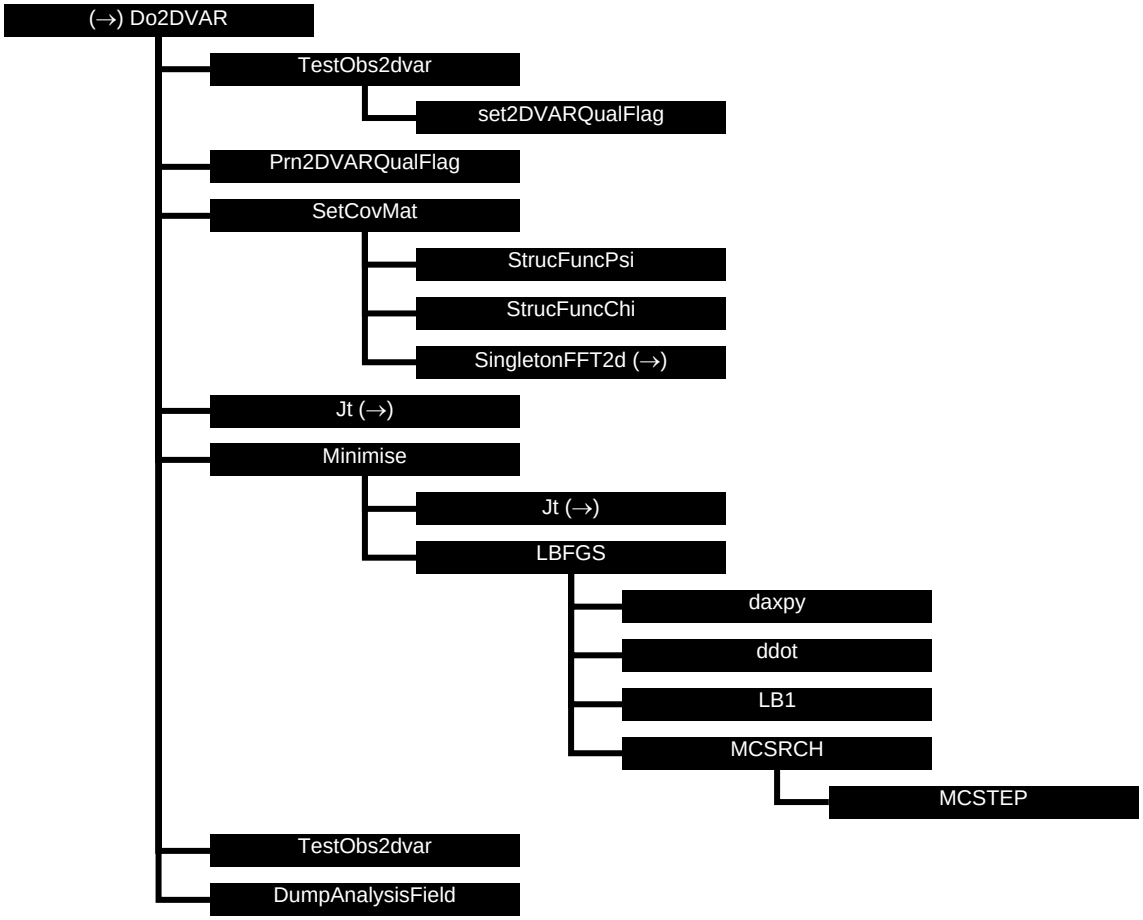


**Figure B2.4** Calling tree for AR routine *Do2DVAR*.

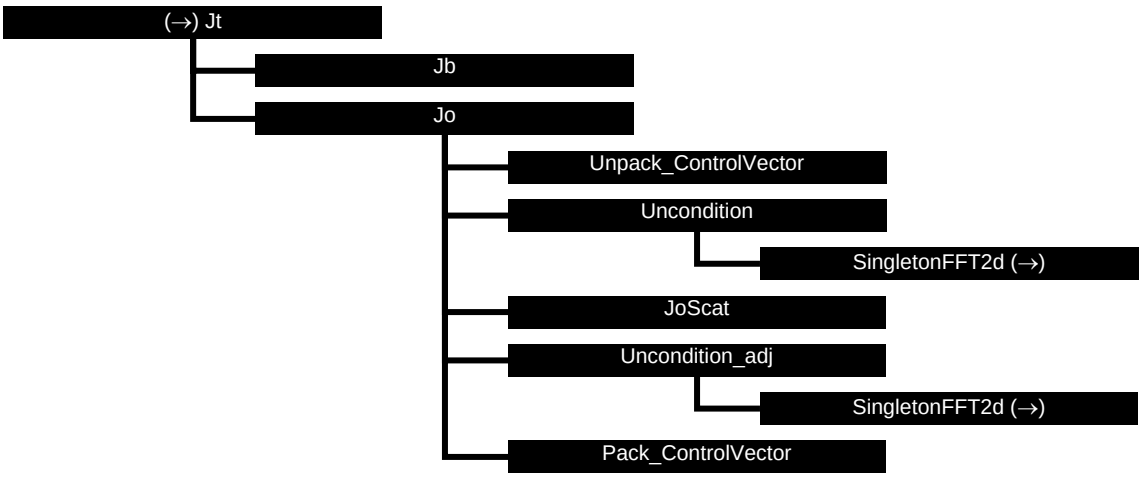


**Figure B2.5** Calling tree for AR routine *Jt* (calculation of cost function).

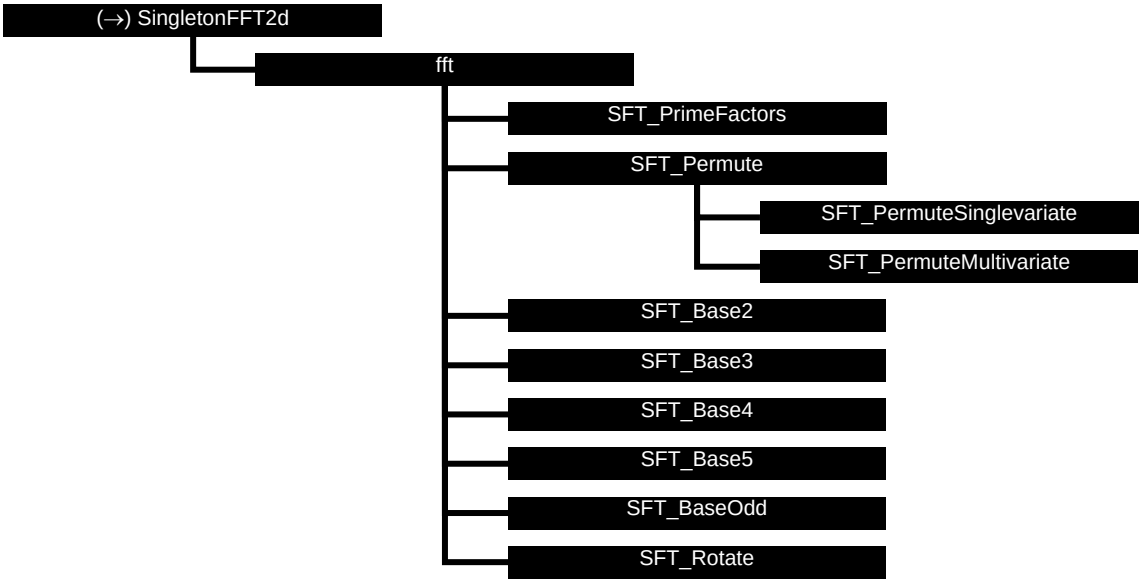


**Figure B2.6** Calling tree for AR routine *SingletonFFT2D*.

# Appendix B3

# Calling tree for BUFR routines

The figures in this appendix show the calling tree for the BUFR file handling routines in genscat. Routines in black boxes are part of genscat. Routines in grey boxes with names completely in capitals belong to the ECMWF BUFR library. Other routines in grey boxes belong to the *bufrio* library (in C). An arrow (→) before a routine name indicates that this part of the calling tree is a continuation of a branch in a previous figure. The same arrow after a routine name indicates that this branch will be continued in a following figure.

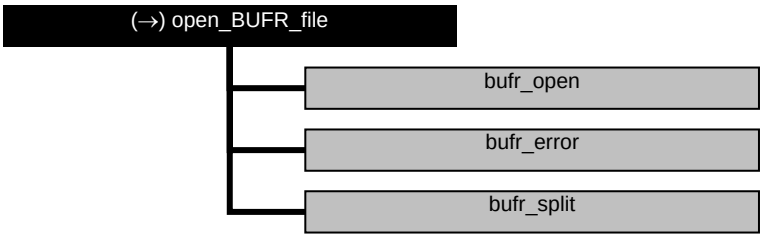


**Figure B3.1** Calling tree for BUFR file handling routine *open_BUFR_file*.

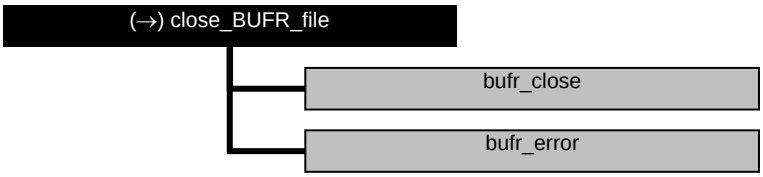


**Figure B3.2** Calling tree for BUFR handling routine *close_BUFR_file*.

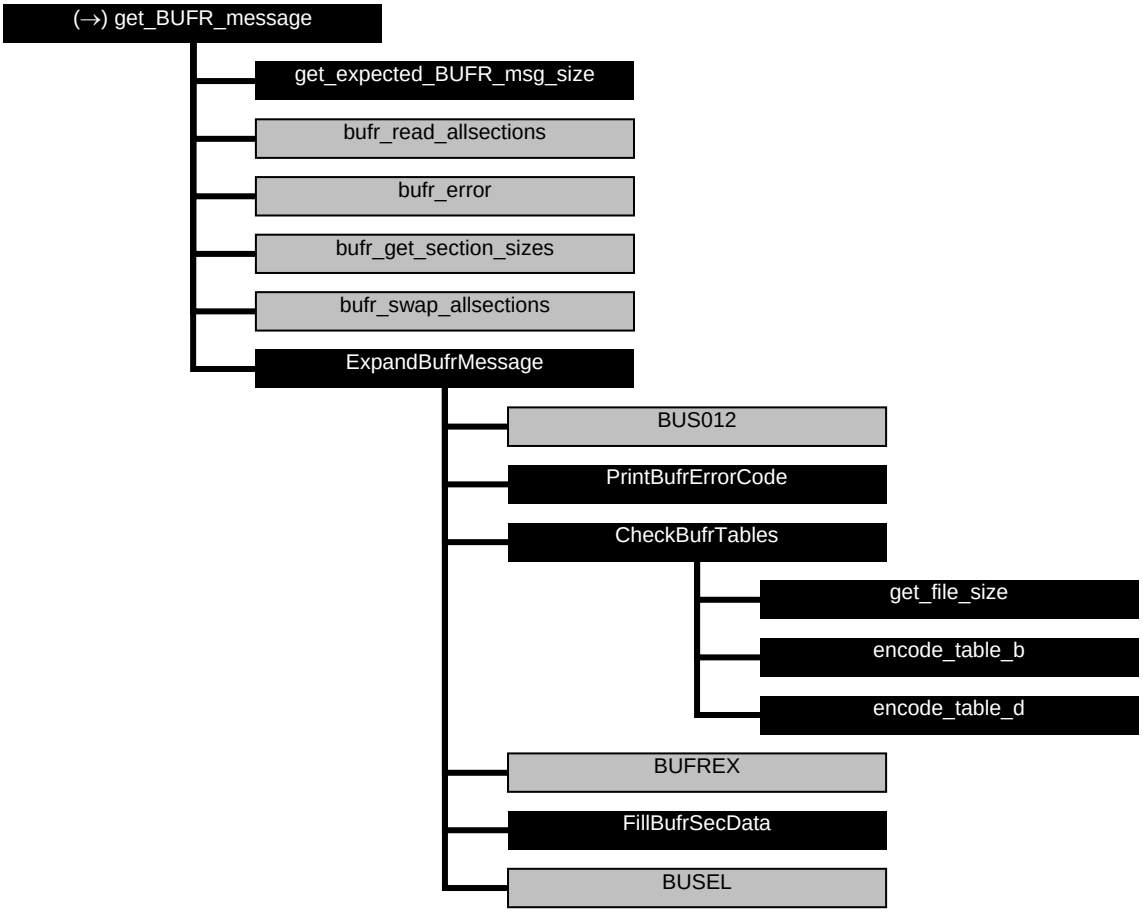


**Figure B3.3** Calling tree for BUFR handling routine *get_BUFR_message*.

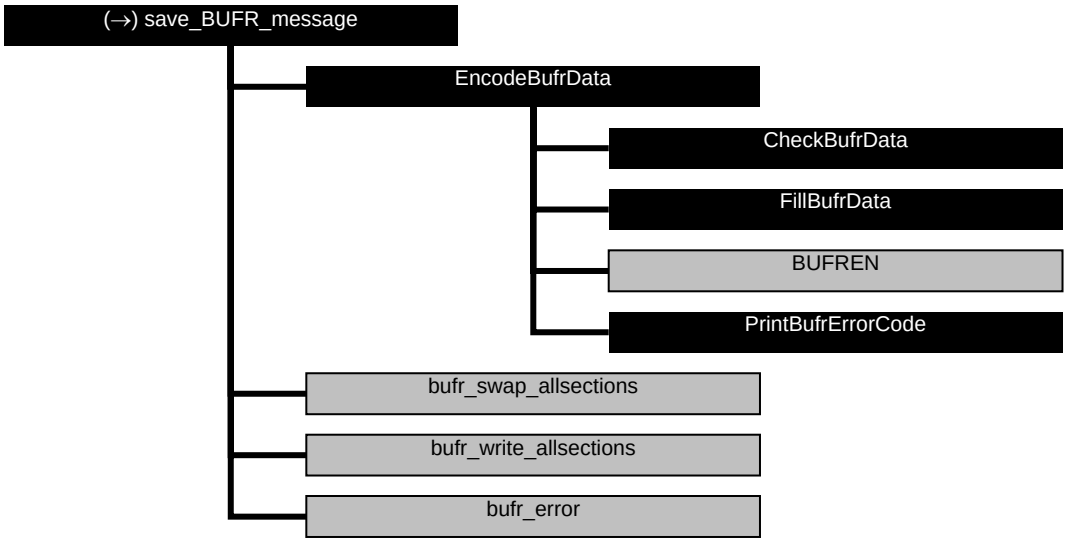


**Figure B3.4** Calling tree for BUFR file handling routine *save_BUFR_file*.

# Appendix B4

## Calling tree for GRIB routines

The figures in this appendix show the calling tree for the GRIB file handling routines in genscat. Routines in black boxes are part of genscat. Routines in grey boxes with names completely in capitals belong to the ECMWF GRIB library. Other routines in grey boxes belong to the *gribio* library (in C). An arrow (→) before a routine name indicates that this part of the calling tree is a continuation of a branch in a previous figure. The same arrow after a routine name indicates that this branch will be continued in a following figure.

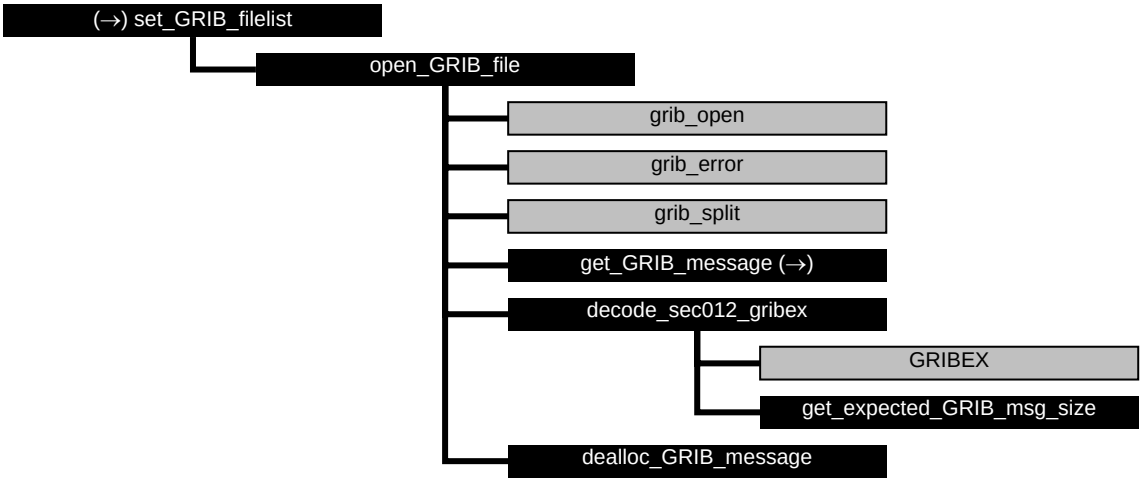


**Figure B4.1** Calling tree for GRIB file handling routine *set_GRIB_filelist*.

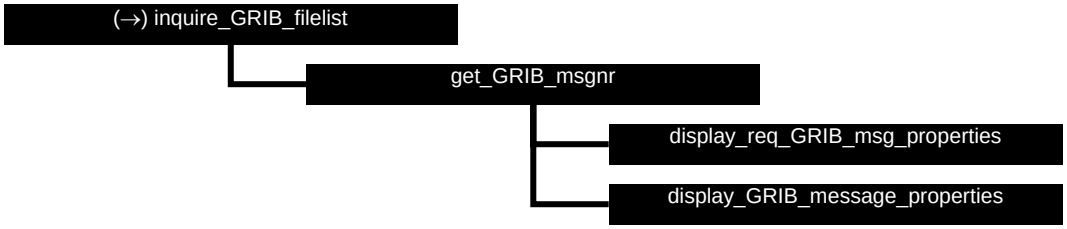


**Figure B4.2** Calling tree for GRIB file handling routine *inquire_GRIB_filelist*.

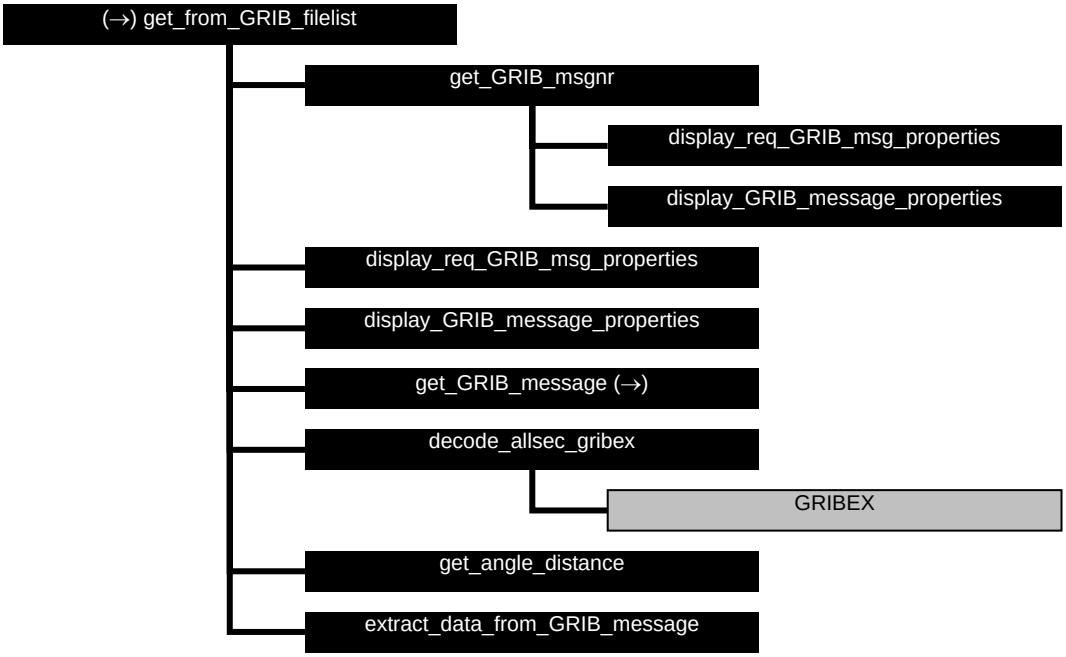


**Figure B4.3** Calling tree for GRIB file handling routine *get_from_GRIB_filelist*.

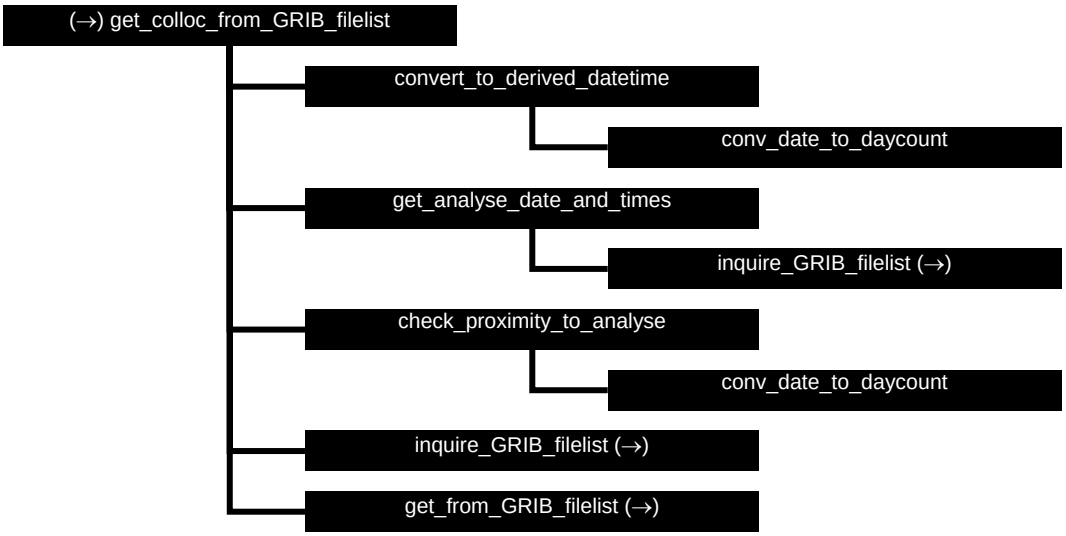


**Figure B4.4** Calling tree for GRIB file handling routine *get_colloc_from_GRIB_filelist*.

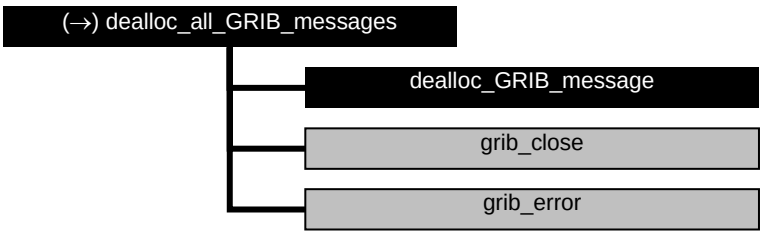


**Figure B4.5** Calling tree for GRIB file handling routine *dealloc_all_GRIB_messages*.

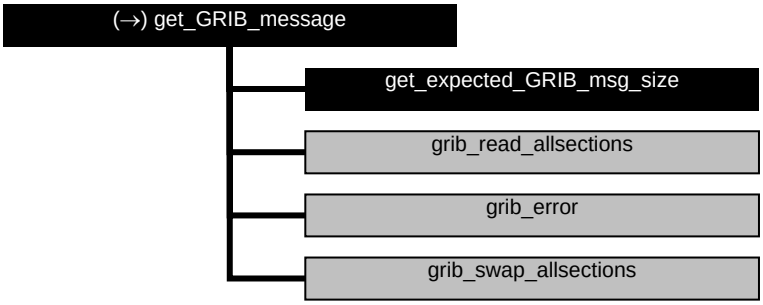


**Figure B4.6** Calling tree for GRIB file handling routine *get_GRIB_message*.

# Appendix B5

## Calling tree for PFS routines

The figures in this appendix show the calling tree for the PFS (native MetOp format) file handling routines in genscat. All routines are part of genscat, as indicated by the black boxes. An arrow (→) before a routine name indicates that this part of the calling tree is a continuation of a branch in a previous figure. The same arrow after a routine name indicates that this branch will be continued in a following figure.

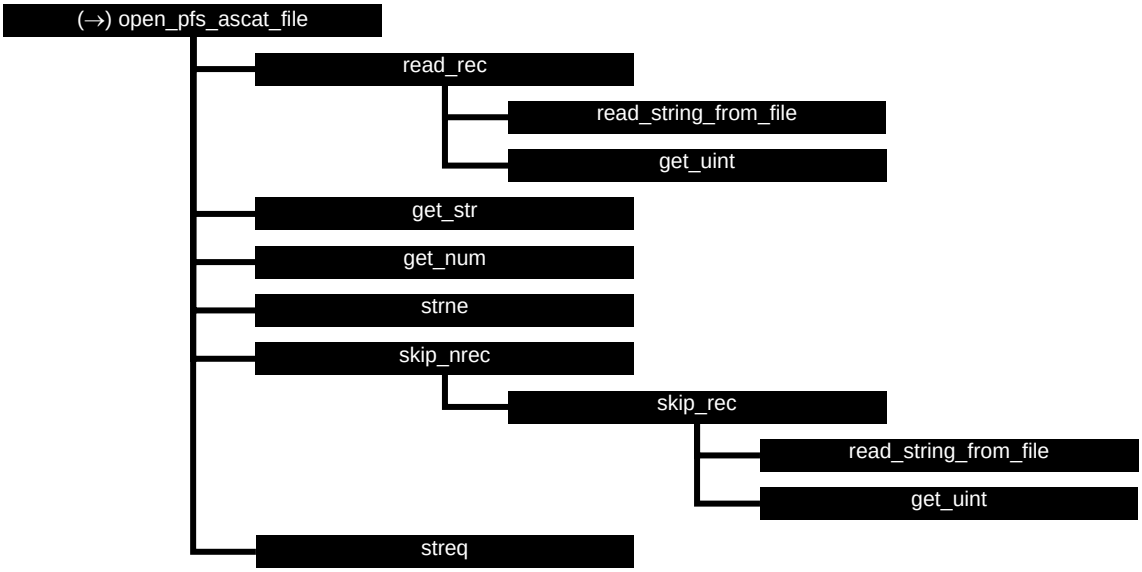


**Figure B5.1** Calling tree for PFS file handling routine *open_pfs_ascat_file*.


**Figure B5.2** Calling tree for PFS file handling routine *read_pfs_ascat_mdr*.

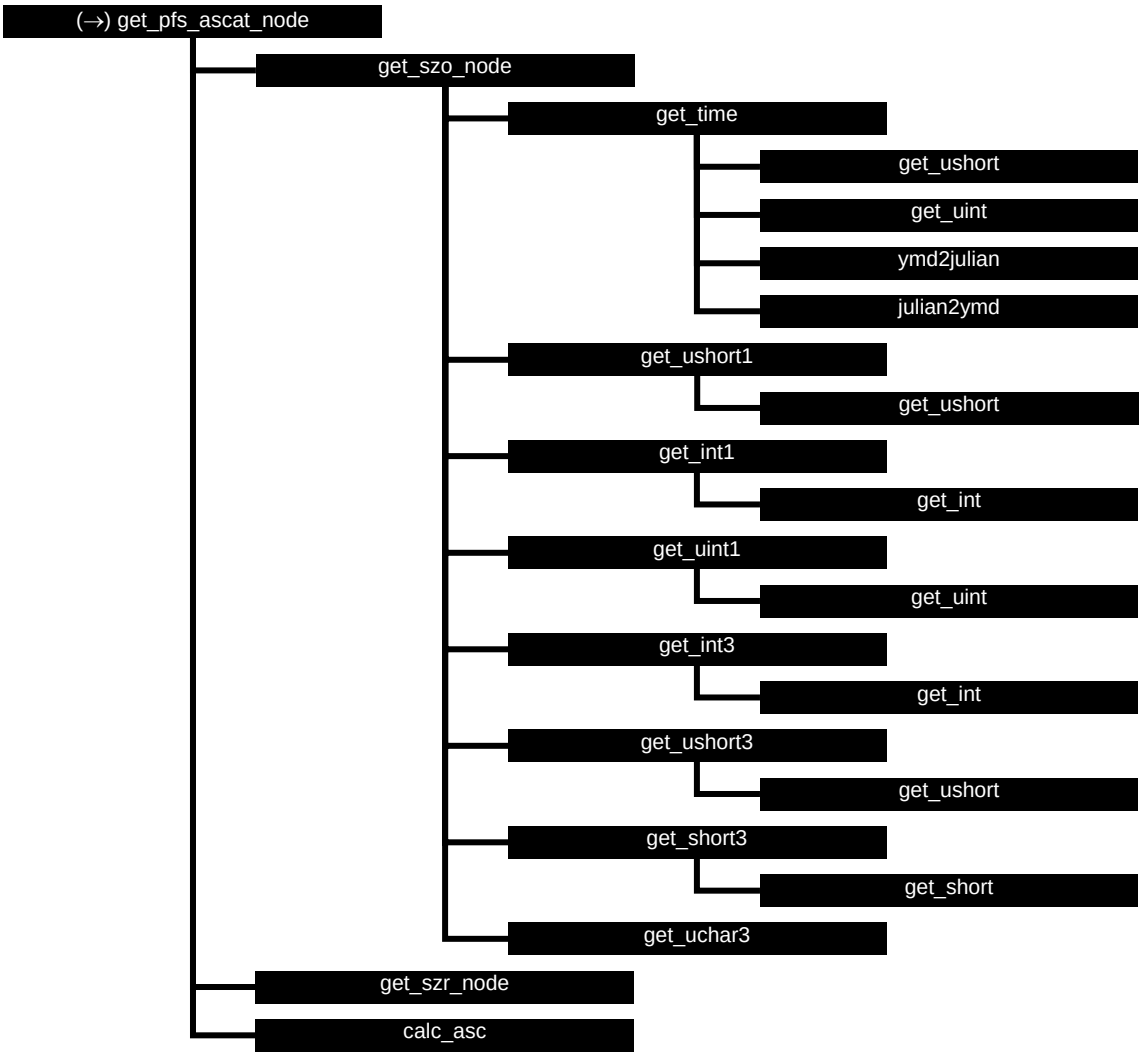


**Figure B5.3** Calling tree for PFS file handling routine *get_pfs_ascat_node*. The calling tree for *get_szr_node* is identical to to the one of *get_szo_node*.

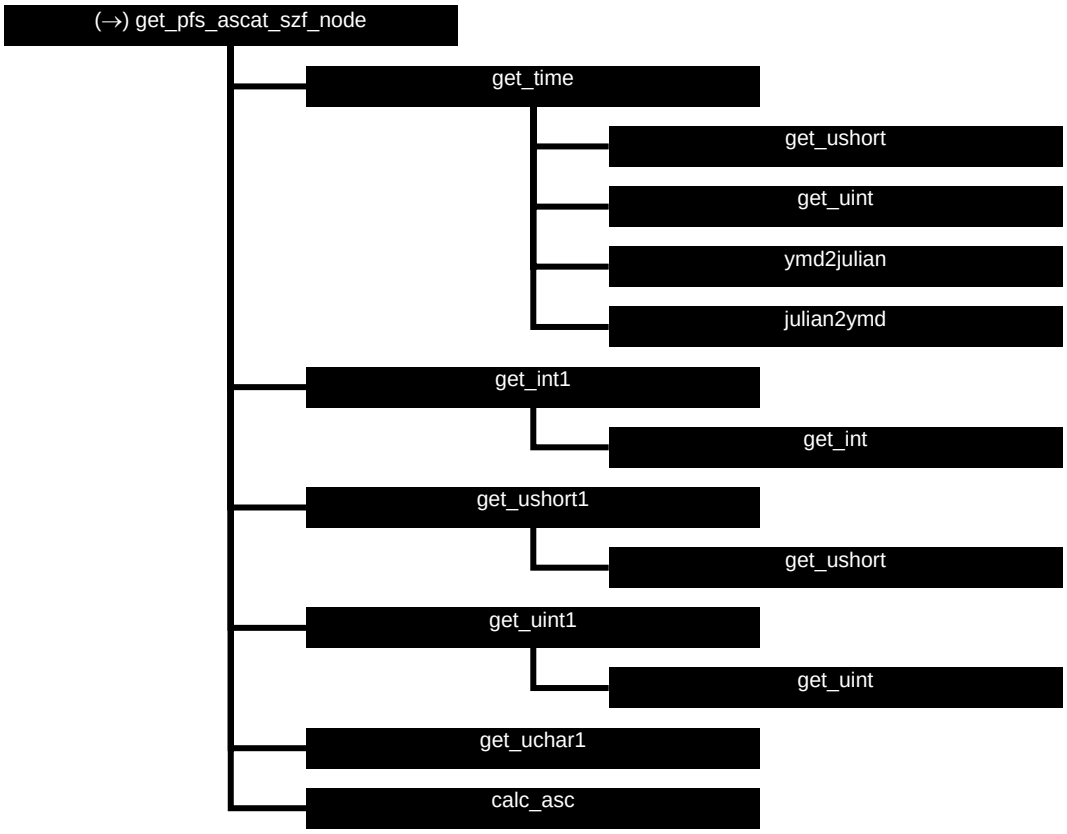


**Figure B5.4** Calling tree for PFS file handling routine *get_pfs_ascat_szf_node*.

# Appendix B6

## Calling tree for ice model routines

The figures in this appendix show the calling tree for the ice model routines in genscat. All routines are part of genscat, as indicated by the black boxes. An arrow (→) before a routine name indicates that this part of the calling tree is a continuation of a branch in a previous figure. The same arrow after a routine name indicates that this branch will be continued in a following figure.

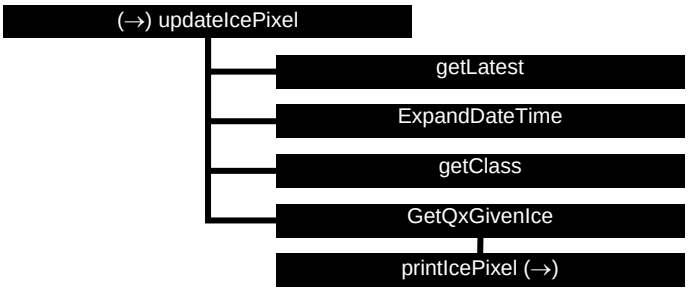


**Figure B6.1** Calling tree for routine *updateIcePixel* (second level).

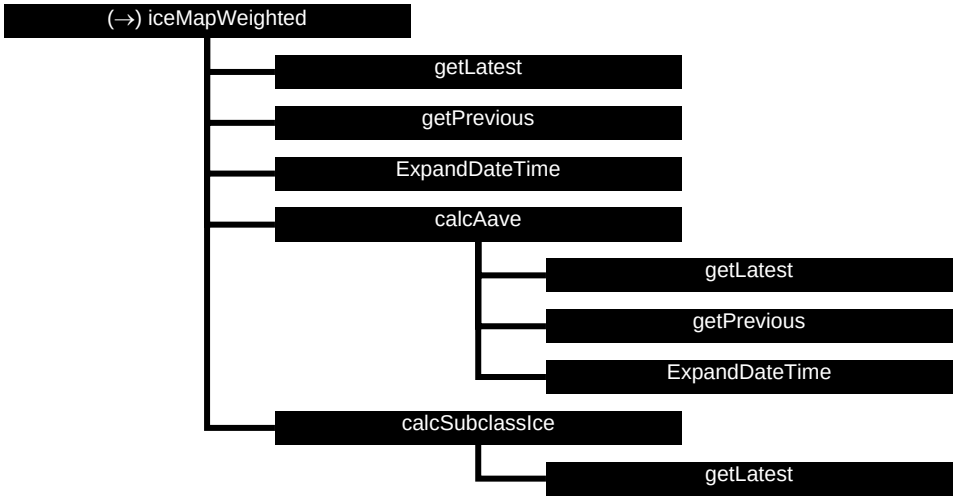


**Figure B6.2** Calling tree for routine *iceMapWeighted* (second level).

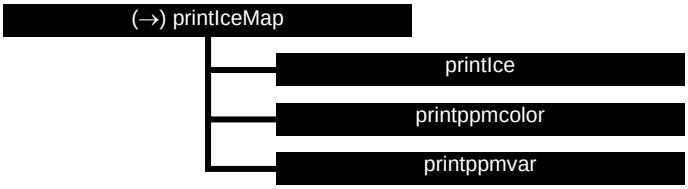


**Figure B6.3** Calling tree for routine *printIceMap* (second level).


**Figure B6.4** Calling tree for routine *printIcePixel* (second level).

# Appendix C

# ASCAT BUFR data descriptors

| Number | Descriptor | Parameter | Unit |
|---|---|---|---|
| 1 | 001033 | Identification Of Originating/Generating Centre | Code Table |
| 2 | 001034 | Identification Of Originating/Generating Sub-Centre | Code Table |
| 3 | 025060 | Software Identification | Numeric |
| 4 | 001007 | Satellite Identifier | Code Table |
| 5 | 002019 | Satellite Instruments | Code Table |
| 6 | 001012 | Direction Of Motion Of Moving Observing Platform | Degree True |
| 7 | 004001 | Year | Year |
| 8 | 004002 | Month | Month |
| 9 | 004003 | Day | Day |
| 10 | 004004 | Hour | Hour |
| 11 | 004005 | Minute | Minute |
| 12 | 004006 | Second | Second |
| 13 | 005001 | Latitude (High Accuracy) | Degree |
| 14 | 006001 | Longitude (High Accuracy) | Degree |
| 15 | 005033 | Pixel Size On Horizontal-1 | M |
| 16 | 005040 | Orbit Number | Numeric |
| 17 | 006034 | Cross Track Cell Number | Numeric |
| 18 | 010095 | Height Of Atmosphere Used | m |
| 19 | 021157 | Loss Per Unit Length Of Atmosphere Used | dB/m |
| 20 | 021150 | Beam Collocation | Flag Table |
| 21 | 008085 | Beam Identifier | Code Table |
| 22 | 002111 | Radar Incidence Angle | Degree |
| 23 | 002134 | Antenna Beam Azimuth | Degree |
| 24 | 021062 | Backscatter | dB |
| 25 | 021063 | Radiometric Resolution (Noise Value) | % |
| 26 | 021158 | ASCAT Kp Estimate Quality | Code Table |
| 27 | 021159 | ASCAT Sigma-0 Usability | Code Table |
| 28 | 021160 | ASCAT Use Of Synthetic Data | Numeric |
| 29 | 021161 | ASCAT Synthetic Data Quality | Numeric |
| 30 | 021162 | ASCAT Satellite Orbit And Attitude Quality | Numeric |
| 31 | 021163 | ASCAT Solar Array Reflection Contamination | Numeric |
| 32 | 021164 | ASCAT Telemetry Presence And Quality | Numeric |
| 33 | 021165 | ASCAT Extrapolated Reference Function | Numeric |
| 34 | 021166 | ASCAT Land Fraction | Numeric |
| 35 | 008085 | Beam Identifier | Code Table |
| 36 | 002111 | Radar Incidence Angle | Degree |
| 37 | 002134 | Antenna Beam Azimuth | Degree |
| 38 | 021062 | Backscatter | dB |
| 39 | 021063 | Radiometric Resolution (Noise Value) | % |
| 40 | 021158 | ASCAT Kp Estimate Quality | Code Table |

| Number | Descriptor | Parameter | Unit |
|---|---|---|---|
| 41 | 021159 | ASCAT Sigma-0 Usability | Code Table |
| 42 | 021160 | ASCAT Use Of Synthetic Data | Numeric |
| 43 | 021161 | ASCAT Synthetic Data Quality | Numeric |
| 44 | 021162 | ASCAT Satellite Orbit And Attitude Quality | Numeric |
| 45 | 021163 | ASCAT Solar Array Reflection Contamination | Numeric |
| 46 | 021164 | ASCAT Telemetry Presence And Quality | Numeric |
| 47 | 021165 | ASCAT Extrapolated Reference Function | Numeric |
| 48 | 021166 | ASCAT Land Fraction | Numeric |
| 49 | 008085 | Beam Identifier | Code Table |
| 50 | 002111 | Radar Incidence Angle | Degree |
| 51 | 002134 | Antenna Beam Azimuth | Degree |
| 52 | 021062 | Backscatter | dB |
| 53 | 021063 | Radiometric Resolution (Noise Value) | % |
| 54 | 021158 | ASCAT Kp Estimate Quality | Code Table |
| 55 | 021159 | ASCAT Sigma-0 Usability | Code Table |
| 56 | 021160 | ASCAT Use Of Synthetic Data | Numeric |
| 57 | 021161 | ASCAT Synthetic Data Quality | Numeric |
| 58 | 021162 | ASCAT Satellite Orbit And Attitude Quality | Numeric |
| 59 | 021163 | ASCAT Solar Array Reflection Contamination | Numeric |
| 60 | 021164 | ASCAT Telemetry Presence And Quality | Numeric |
| 61 | 021165 | ASCAT Extrapolated Reference Function | Numeric |
| 62 | 021166 | ASCAT Land Fraction | Numeric |
| 63 | 025060 | Software Identification | Numeric |
| 64 | 025062 | Database Identification | Numeric |
| 65 | 040001 | Surface Soil Moisture (Ms) | % |
| 66 | 040002 | Estimated Error In Surface Soil Moisture | % |
| 67 | 021062 | Backscatter | dB |
| 68 | 021151 | Estimated Error In Sigma0 At 40 Deg Incidence Angle | dB |
| 69 | 021152 | Slope At 40 Deg Incidence Angle | dB/Degree |
| 70 | 021153 | Estimated Error In Slope At 40 Deg Incidence Angle | dB/Degree |
| 71 | 021154 | Soil Moisture Sensitivity | dB |
| 72 | 021062 | Backscatter | dB |
| 73 | 021088 | Wet Backscatter | dB |
| 74 | 040003 | Mean Surface Soil Moisture | Numeric |
| 75 | 040004 | Rain Fall Detection | Numeric |
| 76 | 040005 | Soil Moisture Correction Flag | Flag Table |
| 77 | 040006 | Soil Moisture Processing Flag | Flag Table |
| 78 | 040007 | Soil Moisture Quality | % |
| 79 | 020065 | Snow Cover | % |
| 80 | 040008 | Frozen Land Surface Fraction | % |
| 81 | 040009 | Inundation And Wetland Fraction | % |
| 82 | 040010 | Topographic Complexity | % |
| 83 | 025060 | Software Identification | Numeric |
| 84 | 001032 | Generating Application | Code Table |
| 85 | 011082 | Model Wind Speed At 10 m | m/s |
| 86 | 011081 | Model Wind Direction At 10 m | Degree True |
| 87 | 020095 | Ice Probability | Numeric |
| 88 | 020096 | Ice Age (A-Parameter) | dB |
| 89 | 021155 | Wind Vector Cell Quality | Flag Table |
| 90 | 021101 | Number Of Vector Ambiguities | Numeric |
| 91 | 021102 | Index Of Selected Wind Vector | Numeric |
| 92 | 031001 | Delayed Descriptor Replication Factor | Numeric |
| 93 | 011012 | Wind Speed At 10 m | m/s |
| 94 | 011011 | Wind Direction At 10 m | Degree True |
| 95 | 021156 | Backscatter Distance | Numeric |

| Number | Descriptor | Parameter | Unit |
|---|---|---|---|
| 96 | 021104 | Likelihood Computed For Solution | Numeric |
| 97 | 011012 | Wind Speed At 10 m | m/s |
| 98 | 011011 | Wind Direction At 10 m | Degree True |
| 99 | 021156 | Backscatter Distance | Numeric |
| 100 | 021104 | Likelihood Computed For Solution | Numeric |

**Table C.1** List of data descriptors. Note that descriptor numbers 93-96 can be repeated 1 to 144 times, depending on the value of the Delayed Descriptor Replication Factor (descriptor number 92)

# Appendix D

# Acronyms

| Name | Description |
|---|---|
| AMI | Active Microwave Instrument, scatterometer on ERS-1 and ERS-2 satellites |
| AR | Ambiguity Removal |
| ASCAT | Advanced SCATterometer on MetOp |
| BUFR | Binary Universal Form for the Representation of data |
| C-band | Radar wavelength at about 5 cm |
| ERS | European Remote Sensing satellites |
| ECMWF | European Centre for Medium-range Weather Forecasts |
| EUMETSAT | European Organization for the Exploitation of Meteorological Satellites |
| genscat | generic scatterometer software routines |
| GMF | Geophysical model function |
| HIRLAM | High resolution Local Area Model |
| KNMI | Koninklijk Nederlands Meteorologisch Instituut (Royal Netherlands Meteorological Institute) |
| Ku-band | Radar wavelength at about 2 cm |
| L1b | Level 1b product |
| LSM | Land Sea Mask |
| LUT | Look up table |
| MetOp | Meteorological Operational Satellite |
| MLE | Maximum Likelihood Estimator |
| MSS | Multiple Solution Scheme |
| NRCS | Normalized Radar Cross-Section ($\sigma_0$) |
| NWP | Numerical Weather Prediction |
| OSI | Ocean and Sea Ice |
| PFS | Product Format Specification (native MetOp file format) |
| QC | Quality Control |
| RFSCAT | Rotating Fan beam Scatterometer |
| RMS | Root Mean Square |
| SAF | Satellite Application Facility |
| SSM | Surface Soil Moisture |
| SST | Sea Surface Temperature |
| WVC | Wind Vector Cell, also called node or cell |

**Table D.1** List of acronyms.