Document NWPSAF-KN-DS-001
Version 2.2
May 2018

PenWP Top Level Design

Anton Verhoef, Jur Vogelzang, Jeroen Verspeek and Ad Stoffelen

KNMI, De Bilt, the Netherlands

PenWP Top Level Design

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 29 June, 2011, 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**
1.9	May 2015	Anton Verhoef	First version for PenWP beta release
2.0	Oct 2015	Anton Verhoef	Version for first public PenWP release, split original UM into UM, PS and TLD docs
2.0.01	Nov 2015	Anton Verhoef	Modified according to DRI comments
2.1	Feb 2017	Jur Vogelzang	Added NBECs
2.2	May 2018	Jur Vogelzang	Extended 2DVAR description; adapted to new 2DVAR

Contents

1 Introduction

The Pencil Beam Wind Processor (PenWP) is a software package written mainly in Fortran 90 with some parts in C for handling data from the SeaWinds (on QuikSCAT or ADEOS-II), OSCAT (on Oceansat-2 or ScatSat-1), HSCAT (on HY-2A) and RapidScat (on the International Space Station) scatterometer instruments. This document is the Top Level Design (TLD) of the PenWP software package and it also contains the Module Design. Section 2 provides information on the general design of the PenWP software. Section 3 and further provide information on the individual modules that are part of PenWP.

More information about PenWP can be found in several other documents. The User Manual and Reference Guide (UM) [1] contains more details about the installation and use of the PenWP package. The Product Specification (PS) [2] provides information on the purpose, outputs, inputs, system requirements and functionality of the PenWP software. Reading the UM and the PS should provide sufficient information to the user who wants to apply the PenWP program as a black box. This TLD document is of interest to developers and users who need more specific information on how the processing is done.

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

1.1 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. `penwp/src`), files (e.g. `penwp.F90`), and commands (e.g. `penwp -f input`) are printed in Courier. Software systems in general are addressed using the normal font (e.g. PenWP, genscat).

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

2 Program Design

In this chapter, the design of the PenWP software package is described in detail. Readers to whom only a summary will suffice are referred to the Top Level Design (TLD) in section 2.1. Readers who really want to know the very detail should not only read the complete chapter, but also the documentation within the code.

2.1 Top Level Design

2.1.1 Main program

The main program, PenWP, (file `penwp` in the `penwp/src` directory) is a Unix (Linux) executable which processes pencil beam Ku-band BUFR input files. The main output consists of BUFR files. The output BUFR messages are in the NOAA BUFR format or in the KNMI BUFR format with generic wind section, for a list of descriptors see appendix A in the Product Specification [2]. 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 [1].

When executed, the PenWP 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 2.1, but note that not all of these steps are always invoked. Some of them will be skipped, depending on the command line options. A more detailed representation of the PenWP 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 PenWP program reads the input file specified in the arguments. The BUFR messages are read and mapped onto the PenWP data structure, see subsection 2.1.3. As part of the pre-processing some checks on the input data are done, the atmospheric attenuations are computed and σ^0 calibration is performed when applicable. 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. 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 2.3.

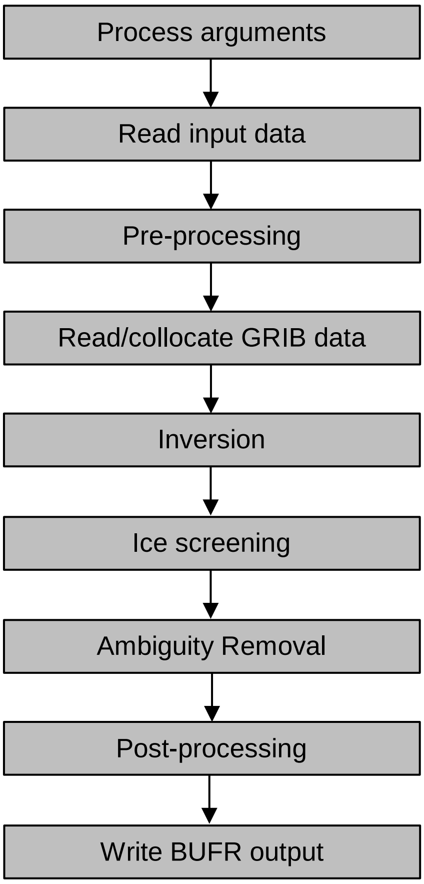

Figure 2.1 Baseline of the Pencil Beam Wind Processor

2.1.2 Layered model structure

PenWP is a Fortran 90 software package consisting of several Fortran 90 modules which are linked after their individual compilation. The PenWP software 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 Ku-band pencil beam specific software. Details on the individual modules can be found in sections 2.2 and 2.3.

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

Module name	Tasks	Comments
penwp_data	Definition of data structures	
penwp_bufr	BUFR file handling	Interface to `genscat/support/bufr`
penwp_prepost	Quality control	Usability of input data is determined
	Atmospheric attenuation	
	Post processing	Setting of flags
	Monitoring	
	Clean up	De-allocation of used memory
penwp_calibrate	Backscatter calibration	
penwp_grib	GRIB file handling	Interface to `genscat/support/grib`
	Collocation of GRIB data	NWP data are interpolated w.r.t. time and location
penwp_inversion	Inversion	Interface to `genscat/inversion`

Module name	Tasks	Comments
penwp_ambrem	Ambiguity Removal	Interface to `genscat/ambrem`
penwp_icemodel	Ice screening	Interface to `genscat/icemodel`

Table 2.1 PenWP process modules.

Each module contains code for performing one or more of the specific tasks. These tasks are briefly described in table 2.1. A more elaborate description is given in section 2.3. The first module listed, *penwp_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 PenWP package are listed in table 2.2. The genscat package is a set of generic modules which can be used to assemble processors as well as pre-processing 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 2.2. The most important classes of modules are related to the inversion processing step (section 3), the Ambiguity Removal step (section 4), the BUFR file handling (section 6), and the GRIB file handling (section 7). 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 section 4
Inversion	Wind retrieval	Inversion in one cell, see section 3
IceModel	Ice screening	Uses ice line and wind cone for ice discrimination
Support	BUFR support	*BufrMod*, based on ECMWF library
	HDF5 support	Reading of HDF5 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 2.2 genscat module classes.

2.1.3 Data Structure

Along track, the scatterometer swath is divided into rows. Within a row (across track), the 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 PenWP 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) data files corresponds to the mapping of a BUFR message to one or more instances of the *row_type* and vice versa.

The main constituent of the *row_type* is the cell data structure, *cell_type*, see figure 2.2. Since most of the processing is done on a cell-by-cell basis the *cell_type* is the pivot data structure of the processor.

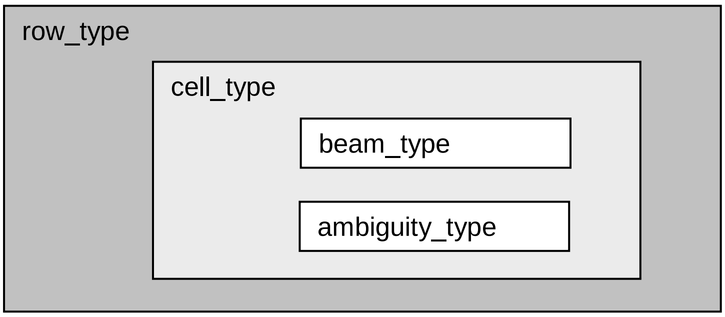

Figure 2.2 Schematic representation of the nested data definitions in the *row_type* data structure.

The σ^0 related level 1b data of a cell are stored in a data structure called *beam_type*. Every cell contains four instances of the *beam_type,* corresponding to the inner fore, outer fore, inner aft, and outer 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.

2.1.4 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 PenWP software two flags are set for every WVC, see table 2.3. 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 2.3 Flags for every WVC (attributes of *cell_type*).

Apart from the flags on WVC level, also the beams contain quality indicators. See section 2.3.1 for more information on this.

2.1.5 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 PenWP 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 2.4 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*
penwp_bufr	*BufrVerbosity*
penwp_hdf5	*hdf5_verbosity*
penwp_grib	*GribVerbosity*
penwp_icemodel	*dbgLevel*

Table 2.4 Verbosity parameters.

2.2 Module design for genscat layer

2.2.1 Module *inversion*

The module *inversion* contains the *genscat* inversion code. Module *post_inversion* contains some routines for probability computations. The modules are located in subdirectory `genscat/inversion`. Details of this module are described in section 3. In the PenWP software package, the inversion module is only used in the *penwp_inversion* module, see section 2.3.6.

2.2.2 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 4. In the PenWP software package, the *ambrem* module is only used in the *penwp_ambrem* module, see section 2.3.7.

2.2.3 Module *icemodel*

The module *icemodel* contains the *genscat* ice screening code. It is located in subdirectory `genscat/icemodel`. In the PenWP software package, the *icemodel* module is only used in the *penwp_icemodel* module, see section 2.3.8.

2.2.4 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 setion 6. In the PenWP software package, the *BufrMod* module is only used in the *penwp_bufr* module, see subsection 2.3.2.

2.2.5 Module *gribio_module*

The *gribio_module* module is the Fortran 90 wrapper around the GRIB API 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 section 7. In the PenWP software package, the *gribio_module* module is used in the *penwp_grib* module, see subsection 2.3.5.

2.2.6 Module *HDF5Mod*

The *HDF5Mod* module is the Fortran 90 wrapper around the HDF5 library from the HDF Group, used for HDF5 input. It is located in subdirectory `genscat/support/hdf5`. In the PenWP software package, the *HDF5Mod* module is only used in the conversion programs `seawinds_hdf2bufr`, `oscat_hdf2bufr`, `oscat_l1b_l2a` and `hscat_hdf2bufr`, see subsection 2.3.10.

2.2.7 Support modules

Subdirectory `genscat/support` contains more support modules besides *Bufrmod*, *gribio_module* and *HDF5Mod*. 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 4.4.

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. It also contains routines for spherical trigonometric calculations used to generate the 2DVAR grid, like angular distance along a great circle and determination of the initial course from one point on a great circle to another.

Subdirectory `datetime` contains module *DateTimeMod* for date and time conversions. PenWP only uses routines *GetElapsedSystemTime* (for calculating the running time of the various processing steps), and *DayJulian* 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. PenWP uses only routines *get_lun* and *free_lun* for opening and closing of a logical unit, respectively.

Subdirectory `num` contains module *numerics* for defining data types and handling missing values, for instance in the BUFR library. This module is needed by many other modules.

Subdirectory `sort`, finally, contains module *SortMod* for sorting the wind vector solutions according to their probability. This module is needed by modules *inversion* and *post_inversion*.

2.3 Module design for process layer

The process layer consists of the modules *penwp_data*, *penwp_bufr*, *penwp_prepost*, *penwp_calibrate*, *penwp_grib*, *penwp_inversion*, *penwp_icemodel* and *penwp_ambrem*. The routines present in these modules are described in the next sections.

2.3.1 Module *penwp_data*

The module *penwp_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 2.5. 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
error_speed	real	Uncertainty in wind speed, not used in PenWP
error_dir	real	Uncertainty in wind direction, not used in PenWP
prob	real	Probability of wind vector solution
conedistance	real	Distance of solution to the GMF

Table 2.5 Ambiguity data structure.

Beam data: Every WVC contains four beams. The information of every beam is stored in the data type *beam_type*. The attributes are listed in table 2.6. 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
sum_weights	real	Sum of weights, used in averaging of level 2a slices
num	integer	Presence of backscatter data, 0 or 1
k_polar	integer	Beam polarisation, 0 = HH pol, 1 = VV pol
lat	real	Beam latitude
lon	real	Beam longitude
atten_value	real	Two-way nadir atmospheric attenuation
azimuth	real	Radar look angle (degrees, counted clockwise from the North)
incidence	real	Incidence angle (degrees, 0 is vertical, 90 is horizontal)
sigma0	real	Radar backscatter (σ^0) in dB
snr	real	Signal to noise ratio
kp_a	real	Noise value Kp α as fraction of 1
kp_b	real	Noise value Kp β as fraction of 1
kp_c	real	Noise value Kp γ in dB
s0_variance_qc	real	σ^0 variance quality control, not used in PenWP
s0_quality	*s0_quality_type*	Flag related to the quality of the backscatter information
s0_mode	*s0_mode_type*	Information about beam type

Attribute	Type	Description
s0_surface	*s0_surface_type*	Information about land or ice presence

Table 2.6 Beam data structure.

Brightness temperature data: The *btemp_type* data type contains information on brightness temperatures. Every WVC contains two brightness temperatures, for the vertically and horizontally polarized beams. The attributes are listed in table 2.7. The routine *init_btemp()* sets all brightness temperature data to missing.

Attribute	Type	Description
k_polar	integer	Beam polarisation, 0 = HH pol, 1 = VV pol
tot_num	integer	Number of slices used in averaging
bright_temp	real	Brightness temperature in K
bright_temp_sd	real	Standard deviation of brightness temperature

Table 2.7 Brightness temperature data structure.

Cell Data: The *cell_type* data type is a key data type in the PenWP software, because many processing steps are done on a cell by cell basis. The attributes are listed in table 2.8. 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_instr_short	integer	Instrument short name, code table 02048
gmf_id	integer	Identifier of GMF used, code table 21119
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
time_to_edge	integer	Time to beginning or end of data file (s)
time_diff_qual	integer	Time difference qualifier, code table 08025
pixel_size_hor	real	Distance between WVCs (meters)
orbit_nr	integer	Orbit number
row_nr	integer	Along track row number
node_nr	integer	Across track cell number
s0_in_cell	integer	Number of beams containing data in cell
rain_prob	real	Probability of rain, not used in PenWP
rain_nof	real	Rain normalised objective function, not used in PenWP
rain_rate	real	Rain rate, not used in PenWP
rain_attenuation	real	Attenuation due to rain, not used in PenWP
btemp (2)	*btemp_type*	Brightness temperature data
beam (4)	*beam_type*	Beam data
software_id_wind	integer	Software identification of level 2 wind processor
generating_app	integer	Generating application of model information

Attribute	Type	Description
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
num_ambigs_n	integer	Number of non-MSS ambiguities
selection	integer	Index of selected wind vector
ambig (0..144)	*ambiguity_type*	Array of wind ambiguities
ice	*ice_type*	Ice information
stress_param	*nwp_stress_param_type*	Wind stress information
process_flag	*process_flag_type*	Processing flag

Table 2.8 Cell data structure.

Ice model data: The *ice_type* contains information related to the ice screening. The attributes are listed in table 2.9. 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
a	real	Ice coordinate
b	real	Ice coordinate
c	real	Ice coordinate
d	real	Ice coordinate
dIce	real	Distance to the ice line
sst	real	Sea surface temperature
wind_sol	real	Wind solution used in ice screening algorithm

Table 2.9 Ice model data structure.

NWP stress parameter data: The *nwp_stress_param_type* data type contains information relevant for wind stress calculations (stress calculation is not implemented in PenWP). The attributes are listed in table 2.10. 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 2.10 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

2.11. A complete row corresponds to a single BUFR message in the PenWP output.

Attribute	Type	Description
num_cells	integer	Actual number of WVC's in this row
cell(76)	*cell_type*	Array of Wind Vector Cells

Table 2.11 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 2.12. 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 2.12 Time data structure.

Wind Data: The *wind_type* data type contains the wind speed and wind direction, see table 2.13. 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 2.13 Wind data structure.

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

Sigma0 quality flag: The *s0_quality_type* data type contains the flag indicating the quality of the σ^0. Each of the four beams in a WVC contains an instance of this flag. The attributes are listed in table 2.14. The function *get_s0_quality()* converts an integer value to the logical flag structure. The function *set_s0_quality()* converts a logical flag structure to an integer value. Note that only a few bits of this flag are used in PenWP.

Attribute	Bit	2^{Bit}	Description
missing			Flag not set (all bits on)
usability	15	32768	σ^0 measurement not usable
noise_ratio	14	16384	Low signal to noise ratio
negative	13	8192	σ^0 is negative

Attribute	Bit	2^{Bit}	Description
range	12	4096	σ^0 is outside acceptable range
pulse	11	2048	Pulse quality not acceptable
convergence	10	1024	Location algorithm does not converge
freq_shift	9	512	Frequency shift beyond range
temperature	8	256	Spacecraft temperature beyond range
attitude	7	128	No applicable attitude records
ephemeris	6	64	Interpolated ephemeris data

Table 2.14 Sigma0 quality flag bits (Fortran).

Sigma0 mode flag: The *s0_mode_type* data type contains the flag indicating the properties of the σ^0 measurement. Each of the four beams in a WVC contains an instance of this flag. The attributes are listed in table 2.15. The function *get_s0_mode()* converts an integer value (BUFR input) to the logical flag structure. The function s*et_s0_mode()* converts a logical flag to an integer value.

Attribute	Bit	2^{Bit}	Description
missing			Flag not set (all bits on)
outer	13	8192	σ^0 is of outer beam
aft	12	4096	σ^0 is aft of satellite
low_res	6	64	Egg data used rather than slice data

Table 2.15 Sigma0 mode flag bits (Fortran).

Sigma0 surface flag: The *s0_surface_type* data type contains the flag indicating land or ice presence in the σ^0 measurement. Each of the four beams in a WVC contains an instance of this flag. The attributes are listed in table 2.16. The function *get_s0_surface()* converts an integer value (BUFR input) to the logical flag structure. The function s*et_s0_surface()* converts a logical flag to an integer value.

Attribute	Bit	2^{Bit}	Description
missing			Flag not set (all bits on)
land	15	32768	Land is present
ice	14	16384	Ice is present
ice_map	5	32	Ice map data not available
atten_map	4	16	Attenuation map data not available

Table 2.16 Sigma0 surface 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 2.17 gives an overview of its attributes. The implementation of this flag is different in the NOAA BUFR format and the KNMI BUFR format with generic wind section. The functions *get_wvc_quality_noaa()* and *get_wvc_quality_gen()* interpret an integer flag (BUFR input) to an instance of *wvc_quality_type*. The functions *get_wvc_quality_noaa()* and *get_wvc_quality_gen()* transform 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 NOAA	2^{Bit} NOAA	Bit KNMI	2^{Bit} KNMI	Description
missing					Flag not set (all bits on)
qual_sigma0	15	32768	22	4194304	Not enough good σ^0 available for wind retrieval
azimuth	14	16384	21	2097152	Poor azimuth diversity among σ^0
kp			20	1048576	Any beam noise content above threshold
monflag	12	4096	19	524288	Product monitoring not used
monvalue	11	2048	18	262144	Product monitoring flag
knmi_qc	10	1024	17	131072	KNMI quality control data rejection
var_qc	9	512	16	65536	Variational quality control data rejection
land	8	256	15	32768	Some portion of wind vector cell is over land
ice	7	128	14	16384	Some portion of wind vector cell is over ice
inversion	6	64	13	8192	Wind inversion not successful
large	5	32	12	4096	Reported wind speed is greater than 30 m/s
small	4	16	11	2048	Reported wind speed is less than or equal to 3 m/s
rain_fail	3	8	10	1024	Rain flag not calculated
rain_detect	2	4	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
four beam	1	2	5	32	One of the four beams is missing
morethan_2_vv	13	8192	4	16	VV polarised beam data in more than two beams

Table 2.17 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 2.18 lists the attributes of the *process_flag_type*. The process flag is only available internally in PenWP. 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 (4)	Invalid data in one of the beams
model_wind	Invalid background wind
ambiguity	Invalid ambiguities
selection	Invalid wind selection

Table 2.18 Cell process flag bits (Fortran).

Table 2.19 provides an overview of all routines and their calls in module *penwp_data*.

Routine	Call	Description
compute_cell_latlon	*merge_rows*	Average beam lat/lon positions to WVC lat/lon position
compute_flight_dir	*preprocess*	Compute satellite flight direction
copy_cell		Copy all information from one cell into another
get_s0_mode	*init_beam*	Convert integer σ^0 mode flag to logical structure
get_s0_quality	*init_beam*	Convert integer σ^0 quality flag to logical structure
get_s0_surface	*init_beam*	Convert integer σ^0 surface flag to logical structure
get_wvc_quality_gen	*init_cell*	Convert integer WVC quality (generic) to logical structure
get_wvc_quality_noaa		Convert integer WVC quality (KNMI) to logical structure
init_ambiguity		Initialise ambiguity structure
init_beam	*init_cell*	Initialise beam structure
init_btemp	*init_cell*	Initialise brightness temperature structure
init_cell		Initialise cell structure
init_ice	*init_cell*	Initialise ice information 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_ice		Print ice information structure
print_nwp_stress_param		Print NWP stress parameters structure
print_process_flag		Print process flag structure
print_s0_mode		Print σ^0 mode flag structure
print_s0_quality		Print σ^0 quality flag structure
print_s0_surface		Print σ^0 surface flag structure
print_time		Print time structure
print_wind		Print wind structure
print_wvc_quality		Print quality flag structure
read_lut_from_file	*init_inversion* *remove_ambiguities*	Read ASCII look-up table from file
set_knmi_flag		Sets/unsets KNMI QC flag depending on other flag settings
set_s0_mode		Convert logical σ^0 mode flag to integer
set_s0_quality		Convert logical σ^0 quality flag to integer
set_s0_surface		Convert logical σ^0 surface flag to integer
set_wvc_quality_gen		Convert logical WVC quality to integer (generic)
set_wvc_quality_noaa		Convert logical WVC quality to integer (NOAA)
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 2.19 Routines in module *penwp_data*

2.3.2 Module *penwp_bufr*

The module *penwp_bufr* maps the PenWP data structure on BUFR messages and vice versa. The *penwp_bufr* module uses the genscat module *BufrMod*, see subsection 2.2.4 for the interface with the BUFR routine library.

Table 2.20 provides an overview of the different routines and their calls in this module.

Routine	Call	Description
bufr_to_row_data_gen	*read_bufr_file*	KNMI format BUFR message into one *row_type*
bufr_to_row_data_noaa	*read_bufr_file*	NOAA format BUFR message into one *row_type*
init_bufr_processing	*read_bufr_file, write_bufr_file*	Initialise module
read_bufr_file	PenWP	Read a complete BUFR file into *row_types*
row_to_bufr_data_gen	*write_bufr_file*	PenWP *row_type* into KNMI format BUFR message
row_to_bufr_data_noaa	*write_bufr_file*	PenWP *row_type* into NOAA format BUFR message
write_bufr_file	PenWP	Write all *row_types* into a complete BUFR file
write_data_row_to_bufr	*write_bufr_file*	Write one *row_type* into a BUFR file

Table 2.20 Routines in module *penwp_bufr*

Note that the BUFR messages always contain exactly one data row.

2.3.3 Module *penwp_prepost*

Module *penwp_prepost* contains the routines to do all the pre-processing 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 completion of missing information, 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
atm_attenuation	*preprocess*	Compute climatological atmospheric attenuations
get_orbit_numbers	*preprocess*	Compute orbit number for OSCAT data
merge_rows	*sort_and_merge*	Merge cells of duplicate data rows
monitoring	*postprocess*	Monitoring
postprocess	PenWP	Main routine of the post processing
preprocess	PenWP	Main routine of the pre processing
process_cleanup	PenWP	Memory management
sort_and_merge	*preprocess*	Sort data rows and merge row information of duplicate rows
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 2.21 Routines of module *penwp_prepost*.

Table 2.21 lists the tasks of the individual routines. PenWP calls *preprocess()* to compute information not present in the level 2a data, like satellite motion direction, time to edge, and atmospheric attenuation. When the input data contain overlapping (duplicate) data rows, the information of these rows in merged in an optimal way, i.e., beam data available in a WVC in one row is used to complete missing beam data in the corresponding WVC of the other row. The *wvc_quality* flag is initialised and the *land* and *ice* flags in *wvc_quality* are set according to the settings of the corresponding flags in the beam *s0_surface* flags.

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 with the same name as the BUFR output file and extension `.mon`. The monitoring parameters are listed in table 2.22. They are calculated separately for five different regions (WVC ranges) of the swath. 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 σ^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*
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 2.22 Parameters in monitoring output.

2.3.4 Module *penwp_calibrate*

The module *penwp_calibrate* performs the calibration of the σ^0's in routine *calibrate_s0*. Based on the results of instrument Ocean Calibration, a bias is added to the backscatter values. The coefficients are obtained specifically for each instrument. Note that the calibration is done again in the reverse order after the post processing in order to write the σ^0's to output as plain copies of the input σ^0's. More information about the calibration can be found in [3].

Routine	Call	Description
calibrate_s0	PenWP	Perform forward or backward backscatter calibration

Table 2.23 Routines in module *penwp_calibrate*

2.3.5 Module *penwp_grib*

The module *penwp_grib* reads in ECMWF GRIB files and collocates the model data with the scatterometer measurements. The *penwp_grib* module uses the genscat module *gribio_module*, see subsection 2.2.5 for the interface with the GRIB routine library.

Table 2.24 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	PenWP	Get land mask, ice mask and background winds using GRIB data
init_grib_processing	*get_grib_data*	Initialise module

Table 2.24 Routines in module *penwp_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, as well as the *ice* flags in the *s0_surface* for each beam. Note that the sea surface temperature screening step is omitted if the ice screening is used; see section 2.3.8. In this case, sea surface temperature information from GRIB will still be used if it is present to support the ice screening. When the sea surface temperature is above 278.15 K (+5.0 °C), the WVC will be assumed to contain no ice.

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, as well as the *land* flags in the *s0_surface* for each beam. 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 cubically interpolated with respect to time linearly interpolated with respect to location and put into the *model_wind* part of each WVC.

2.3.6 Module *penwp_inversion*

Module *penwp_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 2.2.1. Table 2.25 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	PenWP	Loop over all WVCs and perform inversion

Table 2.25 Routines of module *awpd_inversion*.

2.3.7 Module *penwp_ambrem*

Module *penwp_ambrem* controls the ambiguity removal step of the PenWP software. The actual ambiguity removal schemes are implemented in the genscat module *ambrem*, see section 2.2.2. The default method is the KNMI 2DVAR scheme. Table 2.26 lists the tasks of the individual routines.

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

Table 2.26 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 PenWP a batch is just a set of rows. The size of the batch is determined by the resolution of the structure functions and the optimal dimensions for FFT. The routine *remove_ambiguities()* performs the actual ambiguity removal. Finally *select_wind()* passes the selection to the output WVCs.

2.3.8 Module *penwp_icemodel*

Module *penwp_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 σ^0 quadruplets from the four beams 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 [4]. The ice screening is done before the ambiguity removal step. Table 2.27 provides an overview of the routines and their calls in this module.

Routine	Call	Description
calc_aAve	*ice_model*	Calculate space-time averaged values of ice parameter *a*
calc_aSd	*ice_model*	Calculate the standard deviation of ice parameter *a*
calc_ice_coord	*scat_2_ice_map*	Calculate ice coordinates and distance to ice line
calc_pIceGivenX	*ice_model*	Calculate the ice a posteriori probability
calc_SubClass	*ice_model*	Calculate the subclass of a pixel on the ice map
get_class	*update_ice_pixel*	Calculate the ice type of a pixel on the ice map
get_px	*update_ice_pixel*	Get the probability of ice
ice_map_2_scat	*ice_model*	Update cell data structure with information in ice map
ice_model	PenWP	Main routine of ice screening
scat_2_ice_map	*ice_model*	Update the ice map with the information in cell data
smooth	*ice_model*	Smooth the ice map
update_ice_pixel	*scat_2_ice_map*	Update various elements of a pixel on the ice map

Table 2.27 Routines of module *penwp_icemodel*.

2.3.9 Module *penwp*

Module *penwp* is the main program of PenWP. 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.

2.3.10 HDF to BUFR conversion tools

The SeaWinds/RapidScat, OSCAT and HSCAT HDF files all have a different structure. Therefore, three programs for the conversion of HDF5 to BUFR are delivered with PenWP: `seawinds_hdf2bufr`, `oscat_hdf2bufr` and `hscat_hdf2bufr`. All these programs consist of an independent Fortran 90 module with calls to routines in modules *penwp_data*, *penwp_bufr* and *penwp_prepost*. Moreover, several modules in genscat are called. The conversion programs map the datasets in a HDF5 file on the PenWP data structure, which is subsequently written to a BUFR output file.

3 Inversion module

3.1 Background

In the inversion step of the wind retrieval, the radar backscatter observations in terms of the normalized radar cross-sections (σ^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 terms of wind speed and wind direction) to the σ^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 σ^0's. The wind vector solutions correspond to local minima of the MLE function

$$\mathrm{MLE} = \frac{1}{N}\sum_{i=1}^{N}\frac{\left(\sigma^0_{obs}(i) - \sigma^0_{GMF}(i)\right)^2}{K_p(i)} \tag{3.1}$$

With N the number of independent σ^0 measurements available within the wind vector cell, and K_p the covariance of the measurement error. This selection depends on the number of independent σ^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

$$P \propto e^{-\mathrm{MLE}} \quad . \tag{3.2}$$

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 [5] and [6]. PenWP includes the Multiple Solution Scheme (MSS), see [7].

3.2 Routines

The inversion module class contains two modules named *inversion* and *post_inversion*. They are located in subdirectory `genscat/inversion`. Tables 3.1 and 3.2 list all routines in the modules. Appendix B.1 shows the calling tree for the inversion routines.

Routine	Call	Routine	Call
invert_one_wvc	PenWP	*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*

Routine	Call	Routine	Call
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	PenWP	*my_min*	see B.1
init_inv_output	*invert_one_wvc*	*my_max*	see B.1
init_inv_settings_to_default	PenWP	*my_average*	see B.1
write_inv_settings_to_file	not used	*get_indices_lowest_local_minimum*	*invert_one_wvc*
get_inv_settings	PenWP	*my_index_max*	see B.1
set_inv_settings	PenWP	*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 3.1 Routines in module *inversion*.

Routine	Call
normalise_conedist_ers_ascat	not used
calc_kp_ers_ascat	*normalise_conedist_ers_ascat*
calc_geoph_noise_ers_ascat	*calc_kp_ers_ascat*
normalise_conedist_prescat_mode	not used
get_ers_noise_estimate	*normalise_conedist_prescat_mode*
check_ers_ascat_inversion_data	not used
check_wind_solutions_ers_ascat	not used
remove_one_solution	*check_wind_solutions_ers_ascat*
calc_probabilities	PenWP

Table 3.2 Routines of module *post_inversion*.

To establish the MLE function (1), the radar cross section according to the GMF, σ^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 GMF at Ku-band for HH and VV polarization is not known in analytical form. It is only available in the form of lookup tables (in directory `PenWP/data`). The value for σ^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.

3.3 Antenna direction

The output wind direction of inversion routines are generally given in the meteorological convention, see table 3.3. 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 check the convention of the antenna (azimuth) angle and add a correction value if needed.

For Ku-band scatterometers, the radar look angle (antenna angle or simply azimuth) equals 0 if the antenna is orientated towards the North (oceanographic convention). The radar look angle increases clockwise. Therefore, the antenna angle needs does not need a correction.

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 3.3 Conventions for the wind direction.

4 Ambiguity Removal module

4.1 Ambiguity Removal

Ambiguity Removal (AR) schemes select a surface wind vector among the different surface wind vector solutions per WVC 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 WVC, simply called ambiguities, result from the wind retrieval processing 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 minimiser 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. An example is the KNMI Two-Dimensional Variational (2DVAR) scheme.

The implementation of the 2DVAR scheme in PenWP is described in section 4.4.

4.2 Module *ambrem*

Module *Ambrem* is the interface module between the various ambiguity removal methods and the different scatterometer data processors. Table 4.1 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 4.2. Figure 4.1 shows schematically the interdependence of the various modules for Ambiguity Removal.

Routine	Call	Description
InitAmbremModule	PenWP	Initialization of module *Ambrem*
InitAmbremMethod	PenWP	Initialization of specified AR scheme
DoAmbrem	PenWP	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	PenWP	Deallocation of memory

Table 4.1 Routines of module *Ambrem*.

Routine	Description	Documentation
Ambrem2DVAR	Interface to KNMI 2DVAR method	Section 4.4
AmbremBGClosest	Interface to Background Closest method	Section 4.1

Table 4.2 Interface modules for different Ambiguity Removal schemes.

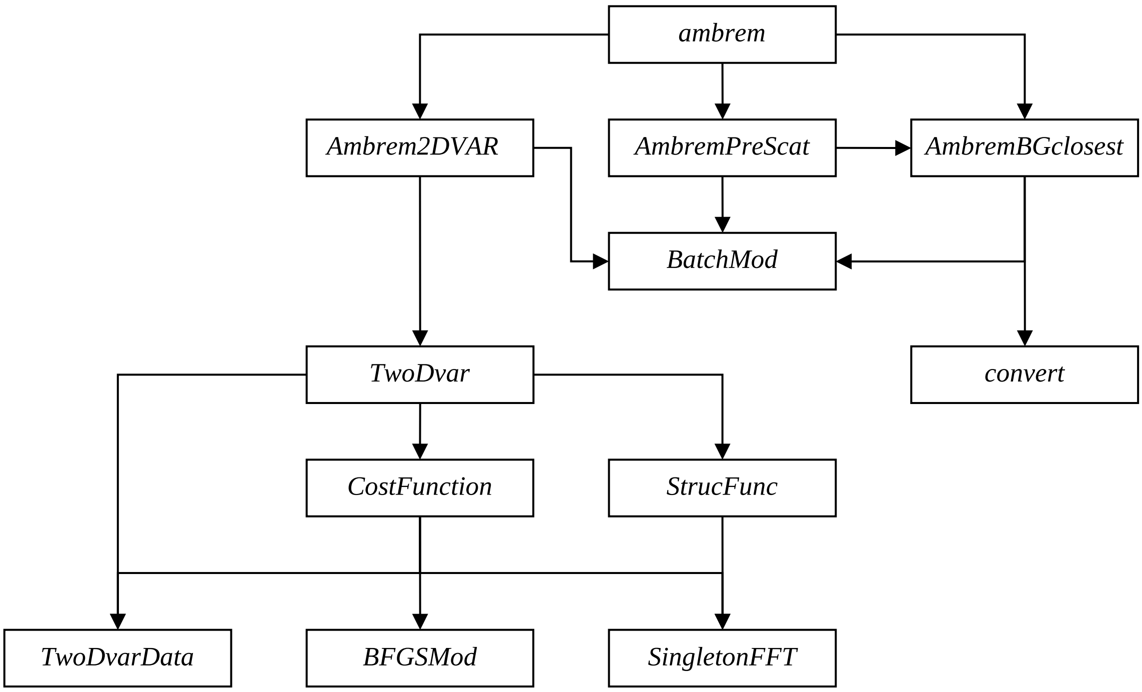

Figure 4.1 Interdependence of the modules for Ambiguity Removal. The connections from module *ambrem* to module *BatchMod* and from module *Ambrem2DVAR* to *convert* are not drawn.

4.3 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 4.2 gives a schematic overview of the batch data structure. Descriptions of the attributes of the individual batch data components are given in table 4.3.

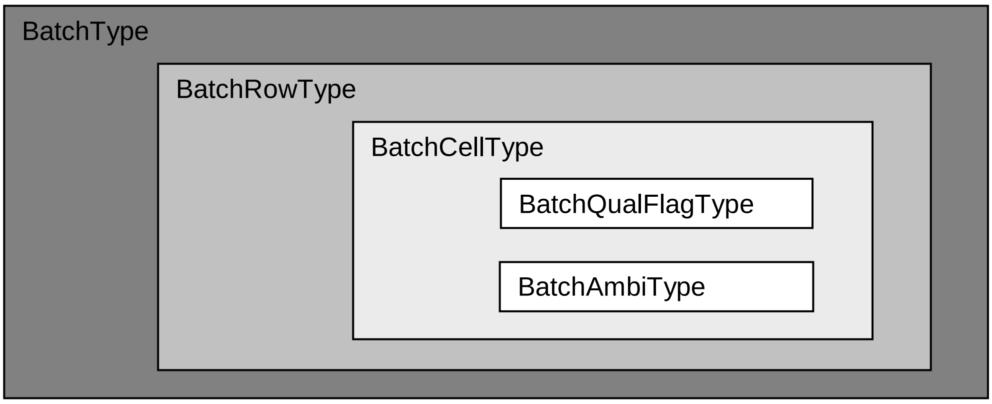

Figure 4.2 Schematic representation of the batch data structure.

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 4.3 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 4.4.

Module *BatchMod* contains a number of routines to control the batch structure. The calls and tasks of the various routines are listed in table 4.5. 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 4.4 Batch quality flag attributes.

Routine	Call	Description
AllocRowsAndCellsAndInitBatch	Processor	Allocation of batch
AllocAndInitBatchRow	*AllocRowsAndCellsAndInitBatch*	Allocation of batch rows
AllocAndInitBatchCell	*AllocAndInitBatchRow*	Allocation of batch cells
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 4.5 Routines of module *BatchMod*.

4.4 The KNMI 2DVAR scheme

4.4.1 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 [8]. Information on 2DVAR can also be found in [9], [10] and [11].

From PenWP version 2.1 onwards, the 2DVAR scheme has been extended with empirical background error correlations, invoked by the `-nbec` command line option. More information on this feature can be found in [16] and references therein.

From PenWP version 2.2 onwards, 2DVAR operates on a grid that is constructed using spherical trigonometric methods. The grid is as regular as possible and correctly handles irregular WVC grids that can be expected from future coastal products.

The calculation of the cost function and its gradient is a 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 4.4.2. Readers interested in the details of the cost function calculations and the minimization should also read the subsequent subsections. Subsection 4.4.3 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 4.4.7 on the actual minimization and subsection 4.4.8 on Fast Fourier Transforms are in fact independent of the cost function itself. The reader might skip these subsections.

4.4.2 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 4.6 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
find_obs_indices_in_2dvar_grid	*BatchInput2DVAR*	Find 2DVAR grid indices and interpolation coefficientsof an observation
get_differenc_vector	*find_obs_indices_in_2dvar_grid*	Calculate difference of two vectors
BatchOutput2DVAR	*Do2DVARonBatch*	Fills the batch data structure with output
Set_WVC_Orientations	*BatchInput2DVAR*	Sets the observation orientation
generate_2dvar_grid	*PenWP*	Generate 2DVAR grid from batch data using spherical trigonometry
dump_2dvar_grid	*PenWP*	For debugging purposes

Table 4.6 Routines of module *Ambrem2DVAR*.

Routine *generate_2dvar_grid* calculates the 2DVAR grid from the batch data. It determines the centres of the first and last row of the batch, defined a great circle through them (the “backbone”), and defines the grid cells at regular distances perpendicular to the backbone (the “ribs”). This routine is called from routine *remove_ambiguities* in module *penwp_ambrem.F90*.

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 4.7. 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 4.7 Routines of module *TwoDvar*.

The *Obs2dvarType* data type is the main data structure for calculating the observation part of the cost function from the observed winds. Its attributes are listed in table 4.8. The *TDV_Type* data type contains all parameters that have to do with the 2DVAR batch grid on which the analysis is actually calculated: dimensions, sizes, and derived parameters. These data structures are defined in module *TwoDvarData* and the routines in this module are listed in table 4.10.

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	Interpolation weight lower left
Wlr	Real	Interpolation weight lower right
Wul	Real	Interpolation weight upper left
Wur	Real	Interpolation 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	df/du
gv	Real	df/dv

Table 4.8 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
VarQC_type	Integer	Type of Variational Quality Control used
NWVC	Integer	Number of WVCs per roe
GEP	Real array	Gross Error Probabilities

Table 4.9 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 4.10 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 4.11.

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 4.11 Attributes of 2DVAR observation quality flag.

The size of the 2DVAR grid grid, *delta*, may be chosen larger than that of the WVCs by an arbitrary factor which is typically 2.This is because the 2DVAR analysis is not needed at its best resolution: it is only used to determine the scatterometer wind direction. Moreover, the true spatial scatterometer resolution is about twice the WVC size. Defining the 2DVAR analysis on a coarser grid also reduces computation time. As a consequence, the analysis must be interpolated to the positions of the observations in order to correctly calculate the observation part of the cost function. That is why the interpolation weights *W* are needed in the *Obs2dvarType* data structure. The weights are calculated in three-dimensional space using simple vector algebra. In doing so, the observation grid may be irregular.

The 2DVAR cost function minimization is done in terms of the wind potential and stream function in the Fourier domain. This is done to keep 2DVAR in line with the ECMWF 4DVAR scheme. Transformation to the Fourier domain simplifies the calculation of the background part of the cost function if the background error correlationa are a function of distance only. As a consequence, the 2DVAR batch grid needs to have a zone free of observations around the observations. The width of this zone (in units of 2DVAR batch grid size) is given by the attribute *GridExtent*. It should be large enough to ensure that the effect of the observations as determined by the background error correlations, has decreased to zero at the edges of the 2DVAR batch grid. Otherwise, numerical errors will be introduced in the Fourier transformations.

4.4.3 Reformulation and transformation

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

$$\min_{v} J(v) \quad , \quad J(v) = J_{obs}(v) + J_{bg}(v) , \qquad (4.1)$$

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

$$J_{bg}(v) = [v - v_{bg}]^T B^{-1} [v - v_{bg}] , \qquad (4.2)$$

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 [8].

4.4.4 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 4.12 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 4.12 Routines of module *CostFunction*.

4.4.5 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., [12] or [13]. For detailed information on the adjoint model in 2DVAR see [8].

4.4.6 Structure Functions

Module *StrucFunc* contains the routines to calculate the covariance matrices (background error correlations, BECs) for the stream function ψ, and the velocity potential χ. Its routines are listed in table 4.13.

Routine	Call	Description
SetCovMat	*Do2DVAR*	Calculate the covariance matrices
StrucFuncPsi	*SetCovMat*	Calculate ψ
StrucFuncChi	*SetCovMat*	Calculate χ

Table 4.13 Routines of module *StrucFunc*.

Routine *SetCovMat* calculates the background error correlation matrix, routines *StrucFuncPsi* and *StrucFuncChi* calculate the BEC for ψ and χ, respectively. By default a Gaussian form is employed for ψ and χ, but this can be changed to empirical BECs with the `-nbec` command line option of PenWP. The empirical BEC is read from file `nbec_ascat-a-coa_cos-auto-4000_tccal_obserrcorr.dat` in directory `genscat/ambrem`.

4.4.7 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 [14]. 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 4.14 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 4.14 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.*

4.4.8 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
SFT_FindNearestDim	PenWP	Find FFT dimension
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 4.15 Fourier transform routines.

Table 4.15 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 PenWP satisfies the requirements in terms of computational speed, this has low priority.

5 Module *iceModelMod*

Module *iceModelMod* is part of the genscat support modules. It contains all the routines for initialising, reading, writing and printing of the SSM/I grids for the North Pole and South Pole region.

5.1 Background

The distribution of backscatter points (combination of $\sigma^0_{\text{HH-fore}}$, $\sigma^0_{\text{VV-fore}}$, $\sigma^0_{\text{HH-aft}}$, and $\sigma^0_{\text{VV-aft}}$) from ocean and sea ice surfaces is notably different. The ice screening method used in PenWP is based on probabilistic distances to ocean wind and sea ice Geophysical Model Functions. Backscatter points closer to the wind GMF have a higher probability of being open water, whereas backscatter points closer to the ice GMF have a higher probability of being ice. A more detailed description of this Bayesian statistics method and ice model is given in [4].

The `-icemodel` option in PenWP basically fills the fields Ice Probability and Ice Age (both present in the KNMI BUFR format with generic wind section). 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 satellite passes over the pole region the corresponding ice map is updated with the new scatterometer 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 data are needed to entirely fill the ice map with data and give meaningful ice model output. Because PenWP handles only one BUFR file at a time, a script is needed that calls PenWP several times. After each PenWP-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 PenWP, 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). All regions having an SST value of more than 5 °C will be assigned as 'surely water'. This helps to suppress wrong ice classifications in rainy areas over open water.

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

```
penwp –f <bufr file> -nwpfl <gribfilelist> -icemodel
```

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

```
penwp –f <bufr file> -icemodel –noinv –noamb
```

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.

5.2 Routines

Table 5.1 provides an overview of the routines in module *iceModelMod*.

Routine	Call	Description
calcPoly3	not used	Calculate a 3rd order polynomial
ExpandDateTime	PenWP	Convert a date/time to a real
ij2latlon	PenWP	Calculate lat lon values from SSM/I grid coordinates
initIceMap	PenWP	Initialise ice map
inv_logit	not used	Calculate the inverse of the logit of p: 1/(1+exp(-p))
latlon2ij	PenWP	Calculate SSM/I grid coordinates from lat lon values
logit	not used	Calculate the logit of p: ln(p/(1-p))
MAPLL	*latlon2ij*	Convert from lat/lon to polar stereographic coordinates
MAPXY	*ij2latlon* (not used)	Convert from polar stereographic to lat/lon coordinates
printClass	not used	Print the class of an ice pixel
print_ice_age_ascat	not used	Print ice age map to graphical .ppm file
printIceAscat	*printIceMap*	Print ice map for ASCAT to graphical .ppm file
printIceMap	PenWP	Print one or more ice map variables to graphical .ppm files
printIcePixel	not used	Print ice pixel information
printIceQscat	*printIceMap*	Print ice map for QuikSCAT/OSCAT to graphical .ppm file
printppm_qc	not used	Print WVC quality flag contents to graphical .ppm file
printppmvar	*printIceMap*	Print variable on ice map to .ppm file, mapped on gray scale
printppmvars	not used	Print three variables to .ppm file, mapped to an RGB scale
printSubclass	*printIceMap*	Print the ice subclass to a .ppm file
RW_IceMap	PenWP	Read or write an ice map from/to a binary restart file
wT	PenWP	Compute moving time average function

Table 5.1 Routines of module *iceModelMod*.

5.3 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**
aIce	real	*a*-ice parameter
aIceAves	real	Average of the *a*-ice parameter
aSd	real	*a*-ice parameter standard deviation
class	integer	Ice class
subClass	integer	Ice subclass
sst	real	Sea surface temperature (K)
pXgivenIce	real	
pXgivenOce	real	
pYgivenIce	real	
pYgivenOce	real	
Pice	real	a-priori ice probability
pIceGivenX	real	a-posteriori ice probability
pIceGivenXave	real	Average a-posteriori ice probability
sumWeightST	real	Sum of weight factors
landmask	logical	land/sea indicator
timePixelNow	DateTime	Date/time of latest ice pixel update
timePixelPrev	DateTime	Date/time of previous ice pixel update

Table 5.2 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
pole	integer	Indicator for North Pole or South Pole
use_sst	logical	Control whether sea surface temp is to be used
timeMapNow	DateTime	Date/time of latest ice map update
timeMapPrev	DateTime	Date/time of previous ice map update
xy	IcePixel(nPixels, nLines)	Pointer to the ice map contents

Table 5.3 Attributes for the *IceMapType* data type.

6 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.

6.1 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 software package. Section 6.2 describes the routines in module *BufrMod*. The publicly available data structures are described in section 6.3. *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 6.4.

6.2 Routines

Table 6.1 provides an overview of the routines in module *BufrMod*. The most important ones are described below.

Routine	**Call**	**Description**
InitAndSetNrOfSubsets	PenWP	Initialization routine
set_BUFR_fileattributes	PenWP	Initialization routine
open_BUFR_file	PenWP	Opens a BUFR file
get_BUFR_nr_of_messages	PenWP	Inquiry of BUFR file
get_BUFR_message	PenWP	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*
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*

Routine	Call	Description
close_BUFR_file	PenWP	Closes a BUFR file
BufrReal2Int	PenWP	Type conversion
BufrInt2Real	PenWP	Type conversion
save_BUFR_message	PenWP	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 6.1 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 6.5. 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 PenWP.

6.3 Data structures

The data type closest to the actual BUFR messages in the BUFR files is the *BufrMessageType*, see table 6.2. 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 6.3, 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 6.2 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 6.3 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)
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

Attribute	Type	Description
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 6.4 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 6.4. 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 6.5.

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 6.5 Attributes of the *bufr_file_attr_data* data type for BUFR files.

6.4 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 [15].

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 6.6. 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 6.6 Routines in library *bufrio*.

6.5 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 the tables are absent, the software tries to create the needed BUFR tables from the text versions, available in genscat.

6.6 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 6.7. Note that these modules are just cosmetic and not essential for the encoding or decoding of the BUFR data. They are not used in PenWP.

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

Table 6.7 Fortran 90 BUFR modules.

7 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 API 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.

7.1 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 software package. Section 7.2 describes the routines in module *gribio_module*. The available data structures are described in section 7.3. The *gribio_module* uses two libraries: from the GRIB software library of ECMWF. This is discussed in some more detail in section 7.4.

7.2 Routines

Table 7.1 provides an overview of the routines in module *gribio_module*. The most important ones are described below.

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

Routine	Call	Description
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
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
read_GRIB_header_info	*open_GRIB_file*	Read 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
get_GRIB_data_values	*get_field_from_GRIB_file, get_from_GRIB_file, get_from_GRIB_filelist*	Read all data from GRIB message
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 7.1 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 PenWP. 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 with constant time intervals (e.g. +3 hrs, +6 hrs, +9 hrs, et cetera or +4 hrs, +5 hrs, +6 hrs, +7 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 optimal 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 PenWP.

7.3 Data structures

Some meta information on the GRIB file is contained in the self-explaining grib*_file_attr_data* data type, see table 7.2.

The decoded GRIB messages in the GRIB files, with their meta information, are contained in the *grib_message_data*, see table 7.3.

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_message_ids	integer array	Message ids assigned by GRIB API
list_of_GRIB_level	integer array	Key to information in messages
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_vals_sizes	integer array	Size of data values arrays

Table 7.2 Attributes for the *grib_file_attr_data* data type.

Attribute	**Type**	**Description**
message_pos_in_file	integer	Position of message in GRIB file
message_id	integer	Message id assigned by GRIB API
date	real	Date when data are valid
time	real	Time when data are valid
derived_date	real	date + time/24

Attribute	Type	Description
derived_time	real	mod(time/24)
total_message_size	integer	Size of message
vals_size	integer	Size of data values array
is_decoded	logical	Status flag
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
real_values	real array, pointer	Decoded real data values

Table 7.3 Attributes for the *grib_message_data* data type.

Attribute	Type	Description
grib_file_attributes	grib_file_attr_data	GRIB file attributes
list_of_GRIB_msgs	grib_message_data array	List of messages in file

Table 7.4 Attributes of the *list_of_grib_files_type* data type for GRIB files.

7.4 Libraries

Module *gribio_module* uses two libraries: from the GRIB API software library of ECMWF: `libgrib_api.a` and `libgrib_api_f90.a`. The GRIB API software library of ECMWF is used as a basis to decode GRIB data. This software library is explained on http://www.ecmwf.int/.

References

[1] Verhoef, A., Vogelzang, J., Verspeek, J. and Stoffelen, A., 2018,
PenWP User Manual and Reference Guide, Report NWPSAF-KN-UD-009, EUMETSAT.

[2] Verhoef, A., Vogelzang, J., Verspeek, J. and Stoffelen, A., 2018,
PenWP Product Specification, Report NWPSAF-KN-DS-002, EUMETSAT.

[3] Risheng Y, A. Stoffelen, A. Verhoef and J. Verspeek, 2012,
NWP Ocean Calibration of Ku-band scatterometers, Proceedings of IGARSS 2012, Munich, Germany, IEEE.

[4] Belmonte Rivas, M. and Stoffelen, A, 2011
New Bayesian algorithm for sea ice detection with QuikSCAT, IEEE Transactions on Geoscience and Remote Sensing, I, **49**, 6, 1894-1901, doi:10.1109/TGRS.2010.2101608.

[5] 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.

[6] Portabella, M., 2002,
Wind field retrieval from satellite radar systems, PhD thesis, University of Barcelona. (Available on http://www.knmi.nl/scatterometer/publications/).

[7] Portabella, M. and Stoffelen, A., 2004,
A probabilistic approach for SeaWinds Data Assimilation, Quart. J. Royal Meteor. Soc., **130**, pp. 127-152.

[8] Vogelzang, J., 2013,
Two dimensional variational ambiguity removal (2DVAR). Report NWPSAF-KN-TR-004, EUMETSAT. (Available on http://www.knmi.nl/scatterometer/publications/ or on https://nwpsaf.eu/deliverables/scatterometer/index.html).

[9] 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/).

[10] 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/).

[11] de Vries, J. and Stoffelen, A., 2000,
2D Variational Ambiguity Removal. KNMI, Feb 2000. (Available on http://www.knmi.nl/scatterometer/publications/).

[12] 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.

[13] Giering, R., 1997,
Tangent linear and Adjoint Model Compiler, Users manual. Max-Planck- Institut fuer Meteorologie.

[14] Liu, D.C., and Nocedal, J., 1989
On the limited memory BFGS method for large scale optimization methods. Mathematical Programming, 45, pp. 503-528.

[15] Dragosavac, M., 2008,
BUFR User's Guide, ECMWF. (Available on http://www.ecmwf.int/)

[16] Vogelzang, J., and Stoffelen, A., 2016,
Developments in ASCAT wind ambiguity removal, v1.0. Report NWPSAF-KN-TR-026, UKMO, UK. (Available on https://nwpsaf.eu/deliverables/scatterometer/index.html).

Appendix A: Calling tree for PenWP

The figures in this appendix show the calling tree for the PenWP software package. Routines in normal print are part of the PenWP process layer. Routines in italic print 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 indicates that this branch will be continued in a following figure.

```
penwp
   ├─ iargc_genscat
   ├─ getarg_genscat
   ├─ write_usage
   ├─ read_bufr_file (→)
   ├─ preprocess (→)
   ├─ calibrate_s0
   ├─ get_grib_data (→)
   ├─ invert_wvcs (→)
   ├─ ice_model (→)
   ├─ remove_ambiguities (→)
   ├─ calibrate_s0
   ├─ postprocess (→)
   ├─ write_bufr_file (→)
   └─ process_cleanup
         └─ GetElapsedSystemTime
```

Figure A.1 Calling tree for *penwp* (top level). Lines ending with an arrow (→) are cut here and will be continued in one of the first level or second level calling trees in the next figures. Lines with italic text indicate genscat routines.

```
(→) read_bufr_file
        ├ GetElapsedSystemTime
        ├ init_bufr_processing
        ├ set_BUFR_fileattributes
        ├ open_BUFR_file (→)
        ├ get_BUFR_nr_of_messages
        ├ get_BUFR_message (→)
        ├ init_cell (→)
        ├ bufr_to_row_data_noaa
        │      ├ BufrReal2Int
        │      ├ get_wvc_quality_noaa
        │      ├ get_s0_quality
        │      ├ get_s0_mode
        │      ├ get_s0_surface
        │      └ test_cell (→)
        ├ bufr_to_row_data_gen
        │      ├ BufrReal2Int
        │      ├ get_s0_quality
        │      ├ get_s0_mode
        │      ├ get_s0_surface
        │      ├ get_wvc_quality_gen
        │      └ test_cell (→)
        └ close_BUFR_file (→)
```

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

```
(→) preprocess
        ├ GetElapsedSystemTime
        ├ sort_and_merge (→)
        ├ ymd2julian
        ├ compute_flight_dir
        │      └ WVC_Orientation
        ├ dB2real
        ├ real2dB
        ├ test_cell (→)
        ├ atm_attenuation
        └ get_orbit_numbers
```

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

```
(→) get_grib_data
        ├ GetElapsedSystemTime
        ├ init_grib_processing
        │       ├ init_GRIB_module
        │       └ set_GRIB_filelist (→)
        ├ inquire_GRIB_filelist (→)
        ├ get_from_GRIB_filelist (→)
        ├ get_colloc_from_GRIB_filelist (→)
        ├ test_cell (→)
        └ dealloc_all_GRIB_messages (→)
```

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

```
(→) invert_wvcs
        ├ GetElapsedSystemTime
        ├ init_inversion
        │       ├ init_inv_settings_to_default
        │       ├ get_inv_settings
        │       ├ set_inv_settings
        │       ├ read_lut_from_file
        │       │       ├ get_lun
        │       │       └ free_lun
        ├ invert_node
        │       ├ init_inv_input
        │       ├ invert_one_wvc (→)
        │       ├ get_closest_solution
        │       │       ├ speeddir_to_u
        │       │       └ speeddir_to_v
        │       └ calc_probabilities
        │               └ GetSortIndex
        ├ set_knmi_flag
        └ test_cell (→)
```

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

```
(→) ice_model
      ├ GetElapsedSystemTime
      ├ initIceMap
      ├ RW_IceMap
      │     ├ get_lun
      │     └ free_lun
      ├ scat_2_ice_map
      │     ├ latlon2ij (→)
      │     ├ calc_ice_coord
      │     ├ speeddir_to_u
      │     ├ speeddir_to_v
      │     ├ SetIntegerDate
      │     ├ SetIntegerTime
      │     └ update_ice_pixel (→)
      ├ calc_pIceGivenX
      │     ├ ExpandDateTime
      │     └ wT (→)
      ├ smooth
      ├ calc_aAve
      │     ├ ExpandDateTime
      │     └ wT (→)
      ├ calc_aSd
      │     ├ ExpandDateTime
      │     └ wT (→)
      ├ calc_SubClass
      ├ ice_map_2_scat
      │     └ set_knmi_flag
      └ printIceMap (→)
```

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

```
(→) remove_ambiguities
        ├ GetElapsedSystemTime
        ├ InitAmbremModule
        │      └ InitBatchModule
        ├ SFT_FindNearestDim
        ├ InitAmbremMethod
        │      ├ InitAmbremBGclosest
        │      ├ InitTwodvarModule (→)
        │      └ InitDummyMethod
        ├ read_lut_from_file
        │      ├ get_lun
        │      └ free_lun
        ├ fill_batch
        │      ├ AllocRowsAndCellsAndInitBatch
        │      │      ├ InitBatch
        │      │      └ AllocAndInitBatchRow
        │      │             ├ InitBatchRow
        │      │             └ InitBatchCell
        │      ├ AllocAndInitBatchCell
        │      │      ├ InitBatchCell
        │      │      └ InitBatchAmbi
        │      ├ speeddir_to_u
        │      ├ speeddir_to_v
        │      └ TestBatch
        │             └ TestBatchRow
        │                    └ TestBatchCell
        ├ generate_2dvar_grid
        │      ├ get_haversine_angular_distance
        │      ├ get_course
        │      └ get_point_from_course
        ├ DoAmbrem (→)
        ├ select_wind
        │      ├ TestBatchCell
        │      ├ set_knmi_flag
        │      └ test_cell (→)
        ├ DeallocBatch
        │      └ DeallocBatchRows
        │             └ DeallocBatchCells
        │                    └ DeallocBatchAmbis
        └ ExitAmbremMethod
               └ ExitTwodvarModule
                      └ TDV_Exit
```

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

```
(→) postprocess
        ├ GetElapsedSystemTime
        ├ monitoring
        │       ├ speeddir_to_u
        │       ├ speeddir_to_v
        │       ├ get_lun
        │       └ free_lun
        ├ write_properties
        │       ├ get_lun
        │       └ free_lun
        └ write_binary_output
                ├ get_lun
                └ free_lun
```

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

```
(→) write_bufr_file
        ├ GetElapsedSystemTime
        ├ init_bufr_processing
        ├ set_BUFR_file_attributes
        ├ open_BUFR_file (→)
        ├ write_data_row_to_bufr
        │       ├ InitAndSetNrOfSubsets
        │       ├ row_to_bufr_data_gen
        │       │       ├ BufrInt2Real
        │       │       ├ set_s0_quality
        │       │       ├ set_s0_mode
        │       │       ├ set_s0_surface
        │       │       └ set_wvc_quality_gen
        │       ├ row_to_bufr_data_noaa
        │       │       ├ BufrInt2Real
        │       │       ├ set_wvc_quality_noaa
        │       │       ├ set_s0_quality
        │       │       ├ set_s0_mode
        │       │       └ set_s0_surface
        │       └ save_BUFR_message (→)
        └ close_BUFR_file (→)
```

Figure A.9 Calling tree for routine write_bufr_file (first level).

```
(→) init_cell
       ├ init_time
       ├ init_btemp
       ├ init_beam
       │      ├ get_s0_quality
       │      ├ get_s0_mode
       │      └ get_s0_surface
       ├ init_wind
       ├ get_wvc_quality_gen
       ├ init_icemodel
       ├ init_nwp_stress_param
       └ init_process_flag
```

Figure A.10 Calling tree for routine *init_cell* (second level).

```
(→) test_cell
       ├ test_time
       ├ test_beam
       └ test_wind
```

Figure A.11 Calling tree for routine *test_cell* (second level).

```
(→) sort_and_merge
       ├ GetSortIndex
       ├ merge_rows
       │      ├ set_knmi_flag
       │      ├ compute_cell_latlon
       │      └ test_cell (→)
       ├ init_cell (→)
       └ copy_cell
```

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

```
(→) print_cell
         ├  print_time
         ├  print_beam
         ├  print_wind
         ├  print_wvc_quality
         ├  print_ambiguity
         └  print_process_flag
```

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

```
(→) update_ice_pixel
         ├  ExpandDateTime
         ├  get_class
         └  get_px
```

Figure A.14 Calling tree for routine *update_ice_pixel* (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 italic printing. 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.

```
(→) invert_one_wvc
        ├  init_inv_settings_to_default
        ├  init_inv_output
        ├  print_message
        ├  check_input_data
        │      ├  print_input_data_of_inversion
        │      ├  my_exit
        │      └  print_message
        ├  convert_sigma_to_zspace
        ├  calc_normalisation
        │      └  calc_var_s0
        ├  find_minimum_cone_dist (→)
        ├  my_min
        ├  my_average
        ├  my_max
        ├  get_indices_lowest_local_minimum
        │      ├  my_index_max
        │      └  print_message
        ├  do_parabolic_winddir_search
        │      └  get_parabolic_minimum
        │             └  my_exit
        ├  GetSortIndex
        ├  SortWithIndex
        ├  calc_sign_MLE
        │      └  calc_sigma0 (→)
        └  fill_wind_quality_code (→)
```

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

```
(→) find_minimum_cone_dist
      ├ calc_cone_distance
      │     └ calc_sigma0 (→)
      └ get_parabolic_minimum
            └ my_exit
```

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

```
(→) calc_sigma0
      ├ read_LUT
      │     ├ get_lun
      │     └ free_lun
      ├ create_LUT_C_VV
      │     ├ get_lun
      │     ├ calc_sigma0_cmod4
      │     │     ├ Get_Br_from_Look_Up_Table
      │     │     └ f1
      │     ├ calc_sigma0_cmod5 (_5, _n)
      │     └ free_lun
      ├ test_for_identical_LUTs
      ├ my_exit
      └ INTERPOLATE
```

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*.

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 italic printing. 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.

```
(→) InitTwodvarModule
      └  TDV_Init
            ├  Set_CFW
            └  Set_HelmholzCoefficients
```

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

```
(→) DoAmbrem
        ├  TestBatch
        │      └  TestBatchRow
        │             └  TestBatchCell
        ├  AmbRem1stRank
        ├  DoAmbremBGclosestOnBatch
        │      └  uv_to_dir
        ├  DoAmbremPreScatOnBatch
        │      └  DoAmbremBGclosestOnBatch
        │             └  uv_to_dir
        ├  Do2DVARonBatch
        │      ├  BatchInput2DVAR
        │      │      ├  TestBatchCell
        │      │      ├  InitObs2DVAR (→)
        │      │      ├  Set_WVC_Orientations
        │      │      │      └  WVC_Orientation
        │      │      ├  rotuv
        │      │      ├  latlon2xyx
        │      │      ├  find_obs_indices_in_2dvar_grid
        │      │      │      └  get_difference_vector
        │      │      └  PrintObs2DVAR
        │      ├  Do2DVAR (→)
        │      └  BatchOutput2DVAR
        │             ├  rotuv
        │             ├  InitObs2DVAR (→)
        │             └  DeallocObs2DVAR
        └  DoDummyMeth
```

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

```
(→) InitObs2dvar
        └  InitOneObs2dvar
               └  TestObs2dvar
                      └  set2DVARQualFlag
```

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

```
(→) Do2DVAR
        ├  TestObs2dvar
        │      └  set2DVARQualFlag
        ├  Prn2DVARQualFlag
        ├  SetCovMat
        │      ├  StrucFuncPsi
        │      ├  StrucFuncChi
        │      └  SingletonFFT2d (→)
        ├  Jt (→)
        ├  Minimise
        │      ├  Jt (→)
        │      └  LBFGS
        │             ├  daxpy
        │             ├  ddot
        │             ├  LB1
        │             └  MCSRCH
        │                    └  MCSTEP
        ├  TestObs2dvar
        │      └  set2DVARQualFlag
        └  DumpAnalysisField
```

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

```
(→) Jt
        ├  Jb
        └  Jo
               ├  Unpack_ControlVector
               ├  Uncondition
               │      └  SingletonFFT2d (→)
               ├  JoScat
               ├  Uncondition_adj
               │      └  SingletonFFT2d (→)
               └  Pack_ControlVector
```

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

```
(→) SingletonFFT2d
       └  fft
              ├  SFT_PrimeFactors
              ├  SFT_Permute
              │      ├  SFT_PermuteSinglevariate
              │      └  SFT_PermuteMultivariate
              ├  SFT_Base2
              ├  SFT_Base3
              ├  SFT_Base4
              ├  SFT_Base5
              ├  SFT_BaseOdd
              └  SFT_Rotate
```

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 italic are part of genscat. Underlined routines followed by (E) belong to the ECMWF BUFR library. Other underlined routines 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.

```
(→) open_BUFR_file
        ├ bufr_open
        ├ bufr_error
        └ bufr_split
```

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

```
(→) close_BUFR_file
        ├ bufr_close
        └ bufr_error
```

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

```
(→) get_BUFR_message
        ├ get_expected_BUFR_msg_size
        ├ bufr_read_allsections
        ├ bufr_error
        ├ bufr_get_section_sizes
        ├ bufr_swap_allsections
        └ ExpandBufrMessage
                ├ BUS012 (E)
                ├ PrintBufrErrorCode
                ├ CheckBufrTables
                │       ├ get_file_size
                │       ├ encode_table_b
                │       └ encode_table_d
                ├ BUFREX (E)
                ├ FillBufrSecData
                └ BUSEL (E)
```

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

```
(→) save_BUFR_message
        ├ EncodeBufrData
        │       ├ CheckBufrData
        │       ├ FillBufrData
        │       ├ BUFREN (E)
        │       └ PrintBufrErrorCode
        ├ bufr_swap_allsections
        ├ bufr_write_allsections
        └ bufr_error
```

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 italic are part of genscat. Underlined routines followed by (E) belong to the ECMWF GRIB API library. 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.

```
(→) set_GRIB_filelist
      └ open_GRIB_file
            ├ grib_open_file (E)
            ├ grib_multi_support_on (E)
            ├ grib_new_from_file (E)
            └ read_GRIB_header_info
                  └ grib_get (E)
```

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

```
(→) inquire_GRIB_filelist
      └ get_GRIB_msgnr
            ├ display_req_GRIB_msg_properties
            └ display_GRIB_message_properties
```

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

```
(→) get_from_GRIB_filelist
        ├ get_GRIB_msgnr
        │      ├ display_req_GRIB_msg_properties
        │      └ display_GRIB_message_properties
        ├ display_req_GRIB_msg_properties
        ├ display_GRIB_message_properties
        ├ get_GRIB_data_values
        │      ├ grib_get (E)
        │      ├ grib_is_missing (E)
        │      └ grib_set (E)
        ├ get_angle_distance
        └ extract_data_from_GRIB_message
```

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

```
(→) get_colloc_from_GRIB_filelist
        ├ convert_to_derived_datetime
        │      └ conv_date_to_daycount
        ├ get_analyse_date_and_times
        │      └ inquire_GRIB_filelist (→)
        ├ check_proximity_to_analyse
        │      └ conv_date_to_daycount
        ├ inquire_GRIB_filelist (→)
        └ get_from_GRIB_filelist (→)
```

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

```
(→) dealloc_all_GRIB_messages
        ├ dealloc_GRIB_message
        ├ grib_release (E)
        └ grib_close_file (E)
```

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

Appendix B5: Calling tree for HDF5 routines

The figures in this appendix show the calling tree for the HDF5 file handling routines in genscat. All routines are part of genscat, as indicated by the italic printing. Underlined routines followed by (H) belong to the HDFGROUP HDF5 library. Other underlined routines belong to the *hdf5io* 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. Note that these routines are called only from the HDF to BUFR conversion tools, see section 2.3.10.

```
(→) h5f_open
        └ h5f_open_c
                ├ H5Fopen (H)
                └ H5Eset_auto (H)
```

Figure B5.1 Calling tree for HDF5 file handling routine *h5f_open*.

```
(→) h5g_open
        └ h5g_open_c
                └ H5Gopen (H)
```

Figure B5.2 Calling tree for HDF5 file handling routine *h5g_open*.

```
(→) h5d_open
        └ h5d_open_c
                └ H5Dopen (H)
```

Figure B5.3 Calling tree for HDF5 file handling routine *h5d_open*.

```
(→) h5a_get_string
        ├ h5a_open_name_c
        │      └ H5Aopen_name (H)
        ├ h5a_get_type_c
        │      └ H5Aget_type (H)
        ├ h5a_read_char_c
        │      └ H5Aread (H)
        ├ h5t_close_c
        │      └ H5Tclose (H)
        └ h5a_close_c
               └ H5Aclose (H)
```

Figure B5.4 Calling tree for HDF5 file handling routine *h5a_get_string*.

```
(→) h5d_get_npoints
        ├ h5d_get_space_c
        │      └ H5Dget_space (H)
        ├ h5s_get_select_npoints_c
        │      └ H5Sget_select_npoints (H)
        └ h5s_close_c
               └ H5Sclose (H)
```

Figure B5.5 Calling tree for HDF5 file handling routine *h5d_get_npoints*.

```
(→) h5d_read_int
        └ h5d_read_int_c
               └ H5Dread (H)
```

Figure B5.6 Calling tree for HDF5 file handling routine *h5d_read_int*.

```
(→) h5d_read_string
        ├ h5a_get_type_c
        │     └ H5Aget_type (H)
        ├ h5t_get_size_c
        │     └ H5Tget_size
        ├ h5a_read_char_c
        │     └ H5Aread (H)
        └ h5t_close_c
              └ H5Tclose (H)
```

Figure B5.7 Calling tree for HDF5 file handling routine *h5d_read_string*.

```
(→) h5d_read_float
        └ h5d_read_float_c
              └ H5Dread (H)
```

Figure B5.8 Calling tree for HDF5 file handling routine *h5d_read_float*.

```
(→) h5d_close
        └ h5d_close_c
              └ H5Dclose (H)
```

Figure B5.9 Calling tree for HDF5 file handling routine *h5d_close*.

```
(→) h5g_close
        └ h5g_close_c
              └ H5Gclose (H)
```

Figure B5.10 Calling tree for HDF5 file handling routine *h5g_close*.

```
(→) h5f_close
        └ h5f_close_c
              └ H5Fclose (H)
```

Figure B5.11 Calling tree for HDF5 file handling routine *h5f_close*.

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 italic printing. 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.

```
(→) latlon2ij
       └  mapll
```

Figure B6.1 Calling tree for routine *latlon2ij*.

```
(→) wT
       ├  ExpandIntegerDate
       ├  ExpandIntegerTime
       └  ymd2julian
```

Figure B6.2 Calling tree for routine *wT*.

```
(→) printIceMap
       ├  printIceAscat
       │     ├  get_lun
       │     └  free_lun
       ├  printIceQscat
       │     ├  get_lun
       │     └  free_lun
       ├  printSubclass
       │     ├  get_lun
       │     └  free_lun
       └  printppmvar
             ├  get_lun
             └  free_lun
```

Figure B6.3 Calling tree for routine *printIceMap*.

Appendix C: Acronyms

Name	**Description**
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
HDF5	Hierarchical Data Format version 5
KNMI	Koninklijk Nederlands Meteorologisch Instituut (Royal Netherlands Meteorological Institute)
Ku-band	Radar wavelength at about 2 cm
L1b	Level 1b product
LUT	Look up table
Metop	Meteorological operational Satellite
MLE	Maximum Likelihood Estimator
MSS	Multiple Solution Scheme
NOAA	United States National Oceanic and Atmospheric Administration
NWP	Numerical Weather Prediction
OSI	Ocean and Sea Ice
PenWP	Pencil Beam Wind Processor
QC	Quality Control
RMS	Root Mean Square
SAF	Satellite Application Facility
SSM/I	Special Sensor Microwave / Imager
SST	Sea Surface Temperature
WVC	Wind Vector Cell, also called node or cell

Table C.1 List of acronyms.