NWP SAF

Satellite Application Facility
for Numerical Weather Prediction

Document NWPSAF-KN-UD-002

Version 1.3

04-09-2006

SDP User Manual and Reference Guide

Scat group

Jos de Kloe, Marcos Portabella, Ad Stoffelen, Anton Verhoef, Jeroen Verspeek, and Jur Vogelzang

KNMI, De Bilt, The Netherlands

SDP User Manual and Reference Guide

Scat group

Jos de Kloe, Marcos Portabella, Ad Stoffelen, Anton Verhoef, Jeroen Verspeek,

and Jur Vogelzang

KNMI, De Bilt, The Netherlands

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

Change record			
Version	Date	Author / changed by	Remarks
0.0	Oct 2004	Hans Bonekamp	First draft
1.0	May 2005	Hans Bonekamp	Beta release
1.1	11-01-2006	Jur Vogelzang	Beta release
1.2	27-03-2006	Jur Vogelzang	First public release
1.3	04-09-2006	Jur Vogelzang	Routines moved from SwsSupport to genscat; index types removed; some typo's corrected.

Contents

Preface **5**

1 Introduction **7**
1.1 Aims and scope 7
1.2 Development of SDP 7
1.3 Testing SDP 8
1.4 User Manual and Reference Guide 8
1.5 Conventions 9

2 SDP User Manual **11**
2.1 Why using the SDP program ? 11
2.2 Modes of using SDP 16
2.3 Installing SDP 16
2.3.1 Directories and files 17
2.3.2 Environment variables 18
2.3.3 Installing BUFR library 19
2.3.4 Compilation and linking 20
2.4 Command line options 22
2.5 Scripts 25
2.6 Testruns 25
2.7 Documentation 27

3 SDP Product Specification **29**
3.1 Purpose of program SDP 29
3.2 Output specification 29
3.3 Input specification 30
3.4 System requirements 30

3.5 Details of functionality 31
3.5.1 BUFR IO and coding 31
3.5.2 Output resolution 31
3.5.3 Quality Control 31
3.5.4 Inversion 32
3.5.5 Ambiguity Removal 32
3.5.6 Monitoring 33
3.6 Details of performance 33

4 Program Design 35
4.1 Top Level Design 35
4.1.1 Main program 35
4.1.2 Layered model structure 36
4.1.3 Data structure 39
4.1.4 Quality flagging and error handling 40
4.1.5 Verbosity 40
4.2 Module Design for genscat layer 41
4.2.1 Module *inversion* 41
4.2.2 Module *ambrem* 41
4.2.3 Module *Bufrmod* 41
4.2.4 Support modules 41
4.3 Module Design for SeaWinds layer 42
4.3.1 Module *SwsData* 42
4.3.2 Module *SwsBufr* 49
4.3.3 Module *SwsSupport* 50
4.4 Module design for process layer 51
4.4.1 Module *SdpSupport* 51
4.4.2 Module *SdpIO* 51
4.4.3 Module *SdpPrePost* 52
4.4.4 Module *SdpInversion* 54
4.4.5 Module *SdpAmbrem* 54

5 Inversion module class 57
5.1 Background 57
5.2 Routines 58
5.3 Antenna direction 59

6 Ambiguity Removal module class 61
6.1 Ambiguity Removal 61
6.2 Module *Ambrem* 62
6.3 Module *BatchMod* 62

6.4 The KNMI 2DVar scheme 65
6.4.1 Introduction 65
6.4.2 Data structure, interface and initialization 66
6.4.3 Reformulation and transformation 68
6.4.4 Module *CostFunc* 70
6.4.5 Adjoint method 71
6.4.6 Structure Functions 72
6.4.7 Complex linear algebra 73
6.4.8 Minimisation 73
6.4.9 MultiFFT 74
6.5 The PreScat scheme 75

7 Module *BufrMod* 77
7.1 Background 77
7.2 Routines 77
7.3 Data structures 79
7.4 Libraries 81
7.4 BUFR table routines 82
7.5 Center specific modules 82

References 83

Appendix A Calling tree for SDP 85

Appendix B1 Calling tree for inversion routines 95

Appendix B2 Calling tree for AR routines 99

Appendix B3 Calling tree for BUFR routines 105

Appendix C SeaWinds BUFR data description 109

Appendix D ECMWF BUFR data routines 113

Appendix E Acronyms 117

Preface

Preface to version 1.0

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

For several years, the KNMI observation research group has been developing processing code to supply a Near Real Time (NRT) level 2 surface wind product based on the SeaWinds Scatterometer level 1 Normalized Radar Cross Section data (σ_0). This work is coordinated and supervised by Ad Stoffelen. In the beginning only an adaptation of his ERS code existed. Later Marcos Portobella and Julia Figa added modifications and extensions to improve, e.g., the wind retrieval and quality control algorithms. In 2003, John de Vries finished the first official release of a processor within the NWP SAF. This processor is called the QuikSCAT Data Processor (QDP). QDP is available for the meteorological community since spring 2004. Several users run QDP operationally. At KNMI, Anton Verhoef is running QDP and providing support for QDP as part of Initial Operational Phase (IOP) of the Ocean Sea-Ice (OSI) SAF wind product.

Meanwhile, Jos de Kloe has been updating the code for ERS and ASCAT scatterometer wind processing. For many parts of the process steps (e.g., the BUFR handling and part of the wind retrieval) a large overlap with SeaWinds Data processing coding exists. The KNMI SCAT group is working towards generic NRT scatterometer processing. As a result, a new modular processing code for SeaWinds data has been developed within the NWP SAF IOP. The working name of this code is currently the SeaWinds Data Processor (SDP). This document is the corresponding reference manual. I hope this manual will strongly contribute to the comprehension of future developers and of users interested in the details of the processing.

Hans Bonekamp, October 2004

Preface to version 1.1

This is the first version of the SDP User Manual and Reference Guide that will be distributed to a larger audience, as deliverable in the NWP SAF project. After Hans Bonekamp left to EUMETSAT, Jos de Kloe, Marcos Portabella, and Anton Verhoef (as beta tester) continued working on the SDP code. They removed a number of bugs and made a lot of improvements: memory management was revised and the Generic Wind Section BUFR format was introduced. My role was to adapt the first draft of this document. With the help of Jos and Anton I found my way into the code. I made a number of adaptations and extensions to the original text, but left the underlying structure of the document unchanged.

The reader is kindly invited to give his comments in order to improve future versions of this document.

Jur Vogelzang, September 2005

Preface to version 1.2

Version 1.2 will be the first public version of SDP. The recommendations made by EUMETSAT during the Delivery Readiness Inspection in November 2005 were all implemented. Moreover, almost all known problems have been solved. The reader is kindly invited to give his comments in order to improve future versions of this document.

Jur Vogelzang, March 2006

Preface to version 1.3

Version 1.3 is an update of the first public version of SDP. Some routines in modules SwsSupport and Ambrem2DVAR were moved to genscat, which led to some differences in the program structure. The index_type datatype is no longer needed and has been removed. The importance of setting the environment variables to their proper values during compilation and linking has been stressed. The inversion module has been improved at very low wind speed and flag management has been revised. Some typo's were corrected.

Jur Vogelzang, September 2006

Chapter 1

Introduction

1.1 Aims and scope

The SeaWinds Data Processor (SDP) is a software package written in Fortran90 for handling data from the SeaWinds scatterometer instruments. Details of these instruments can be found on several sites and in several other documents. Important references are listed at the end of this section.

SDP generates surface winds based on SeaWinds data. In particular, it allows performing the ambiguity removal with the 2DVar method and it supports the MSS scheme, as an alternative to the DIRTH scheme employed by NOAA. The output of SDP consists of wind vectors which represent surface winds within the ground swath of the scatterometer. Input of SDP are Normalized Radar Cross Section (NRCS, σ_0) data. These data may be real-time. The input and output files of SDP are in BUFR format.

For SeaWinds on QuikSCAT the data are available for several years. Unfortunately, due to its failure after 9 months, a ready to use real-time (BUFR, see subsection 3.5.1) product for Seawinds on Adeos II is not available.

More information can be found in [*Kerkmann*, 1998; *Leidner et al.*, 2000; *Portabella*, 2002; *Stoffelen*, 1998].

1.2 Development of SDP

SDP is developed within the NWP SAF IOP program as code which can be run in an operational setting. The coding is in Fortran 90 and has followed the procedures specified for the NWP SAF.

Table 1.1 provides an overview of the persons involved in the development. Special attention has been paid on robustness and readability. SDP may be run on every modern UNIX or LINUX machine. In principle, SDP can also be run on a Windows machine if a UNIX emulator like Cygwin or MinGW is installed.

Task	Person
Coordinator	Ad Stoffelen
Lead Investigator	Hans Bonekamp, Jur Vogelzang
Development Team	Hans Bonekamp, Jos de Kloe, Anton Verhoef, Jur Vogelzang
Integrator	Hans Bonekamp, Jur Vogelzang
Project Team Leader	Ad Stoffelen
Beta testers	Marcos Portabella, Anton Verhoef
Reviewers	Ad Stoffelen, Jos de Kloe, Marcos Portabella

Table 1.1 Overview of development tasks.

1.3 Testing SDP

Modules are tested by test programs and test routines. Many test routines or test support routines are part of the modules themselves. Test programs can be compiled separately. For the SDP program, the description of the test programs and the results of the testing are reported in [*SCAT group, 2005*].

1.4 User Manual and Reference Guide

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

Chapter 2 is the user manual (UM) for the SDP program. This chapter provides the basic information for installing, compiling, and running SDP.

Chapter 3 contains the Product Specification (PS) of the SDP program. Reading the UM and the PS should provide sufficient information to the user who wants to apply the SDP program as a black box.

The subsequent chapters are of interest to developers and users who need more specific information on how the processing is done. The Top Level Design (TLD) of the code and the Module Design (MD) of the SDP code can be found in chapter 4.

Several modules are very generic for NRT scatterometer data processing. Examples are the modules for the BUFR handling, ambiguity removal, and parts of the wind retrieval. These generic modules are part of the genscat layer and are described in chapters 5, 6 and 7.

The appendices of this document contain a complete calling tree of the SDP program up to and including the genscat layer. The appendices also contain a list of SeaWinds BUFR data descriptors, a list of the ECMWF BUFR routines, and a list of acronyms.

Finally, many sections end with a remarks alinea. Mostly, the remarks contain some

recommendations for future development or explain the correspondence with the QDP code. These remarks will be reconsidered in future versions of this reference book.

1.5 Conventions

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

Names of directories and subdirectories (e.g. `/SDP/sdp`), files (e.g. `sdp.F90`), and commands (e.g. `sdp -f input`) are printed in Courier. When addressing software systems in general, the normal font is used (e.g. SDP, genscat).

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

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

Chapter 2

SDP User Manual

This chapter is the user manual of the SDP program. The SDP program is the follow-up of the QDP program. Therefore, the QDP user manual [*de Vries et al.*, 2004] is in some cases appropriate to understand the operations of SDP. However, SDP has extended capabilities, such as higher resolution and the Multi Solution Scheme (MSS).

Section 2.2 provides information on how to install, compile, and link the SDP software. The command line arguments of SDP are discussed in section 2.3. Section 2.4 gives information on some scripts for running SDP that are part of this release.

2.1 Why using the SDP program ?

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

The general scheme of SDP (and any other wind scatterometer data processor is given in figure 2.1. The input of the SDP program is the NOAA SeaWinds level 2b BUFR wind product. However, only the level 1 data contained in the NOAA (the σ_0 values) are used in SDP.

The SDP processing chain contains five steps (see figure 2.1):

1. Pre-processing. The input BUFR file is decoded and the σ_0 values are written in the data structures of SDP.

2. Inversion. The σ_0 values are compared to the Geophysical Model Function (GMF) by means of a Maximum Likelihood Estimator (MLE). The wind vectors that give the best description of the σ_0 values (the solutions) are retained. The MLE is also used to assign a probability to each wind vector. The normal scheme allows 4 solutions at most, but in the Multi Solution Scheme (MSS) the maximum number of solutions is 144.
3. Quality Control. Solutions that lie far away from the GMF are likely to be contaminated by rain, sea ice, and/or confused sea state. During Quality Control these solutions are identified and flagged.
4. Ambiguity Removal. This procedure identifies the most probable solution using some form of external information. SDP uses a two-dimensional variational scheme (2DVar) as default. A cost function is minimized that consists of a background wind field and all solutions with their probability, using mass conservation and continuity as constraints. The background wind field is obtained from the NCEP model winds in the NOAA SeaWinds level 2b product (the input file of SDP).
5. Quality Monitoring. The last step is to write the results in BUFR format and to output quality indicators.

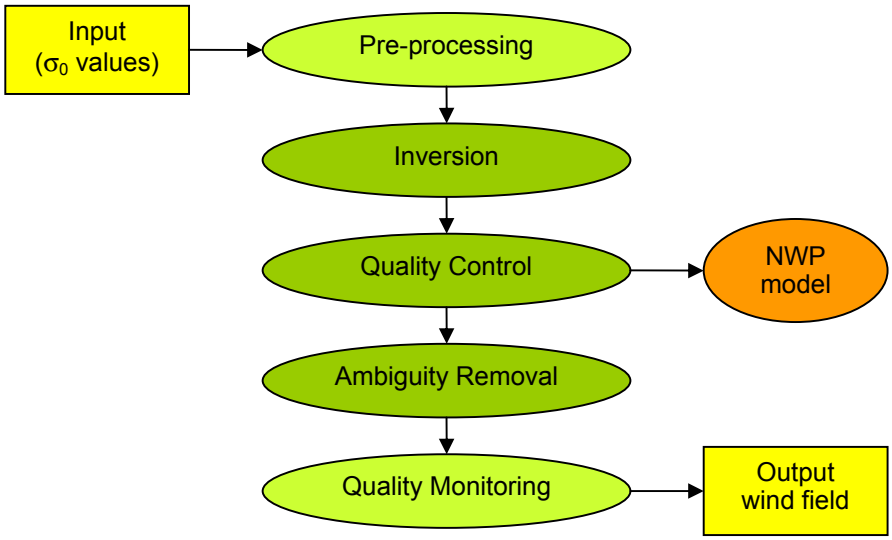


Figure 2.1 SDP processing scheme. When using MSS the wind vectors and their probabilities after Quality Control may be fed directly in the Data Assimilation step of a Numerical Weather Prediction model.

Step 1 and 5 of the processing chain are rather trivial; the real work is done in steps 2, 3, and 4. Note that an inconsistency may arise if the output wind field is assimilated into a numerical weather prediction (NWP) model: in the data assimilation step the scatterometer wind field will be checked for mass conservation and continuity, but this has already been done in the 2DVar

Ambiguity Removal step! Therefore it is recommended to feed the wind solutions and their probabilities directly into the NWP data assimilation step after Quality Control, as indicated in figure 2.1.

As further detailed in chapter 3, SDP profits from developments in

- inversion and output of the full probability density function of the vector wind (Multi Solution Scheme, MSS);
- rain detection and Quality Control (QC);
- meteorologically balanced Ambiguity Removal (2DVar);
- quality monitoring;
- variable resolution.

Figure 2.2 shows some example wind fields that demonstrate the improvements achievable with SDP using MSS and 2DVar.

Another important - but not yet validated - aspect of the SDP program is the possibility to create an output wind product with a different resolution. Figure 2.3 shows an example of a SDP result at 25 km resolution. There is, of course, a trade-off between the output resolution and the output accuracy. The SDP program may help to process the data in the most appropriate manner for the application under consideration.

SDP yields wind fields with high accuracy. Table 2.1 shows the results of a study on its accuracy. The table gives the mean root mean square difference with the ECMWF First Guess at Appropriate Time (FGAT) for the SDP processed winds without and with MSS. As a reference, the results for NCEP model winds are given in the last column. The MSS result is much better, especially at nadir, and further improves on the NCEP winds.

Swath region	SDP standard	SDP with MSS	NCEP
Sweet	2.48	2.23	2.85
nadir	2.98	2.45	2.96

Table 2.1 Mean vector root mean square difference with ECMWF FGAT winds for SDP processed winds and 1000 mb level NCEP model winds.

A complete specification of the SDP program can be found in the Product Specification in Chapter 4. The program is based on generic genscat routines for inversion, ambiguity removal, and BUFR file handling. These routines are discussed in more detail in chapters 5 – 7.

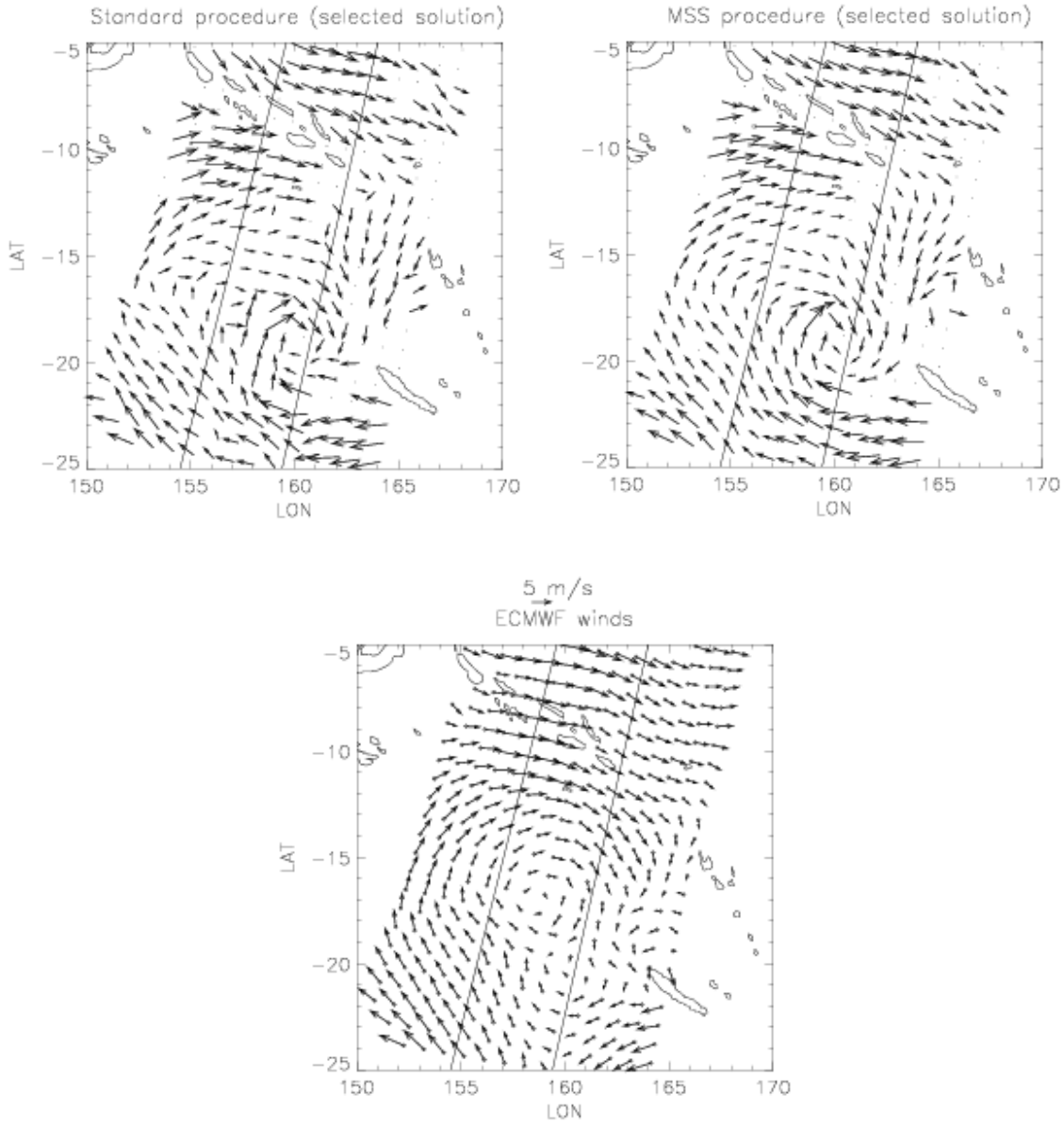


Figure 2.2 An example of the advantages of SDP. The upper left image shows the wind field obtained from SeaWinds using the standard NOAA processing. The field contains some errors at low wind speeds and doesn't look smooth. The upper right image shows the wind field obtained by running SDP in MSS mode, retaining the most probable solution. As a reference, the lower image shows the ECMWF first guess winds. The scatterometer fields contain more detail and, even more important for prediction, put the structure at a different location.

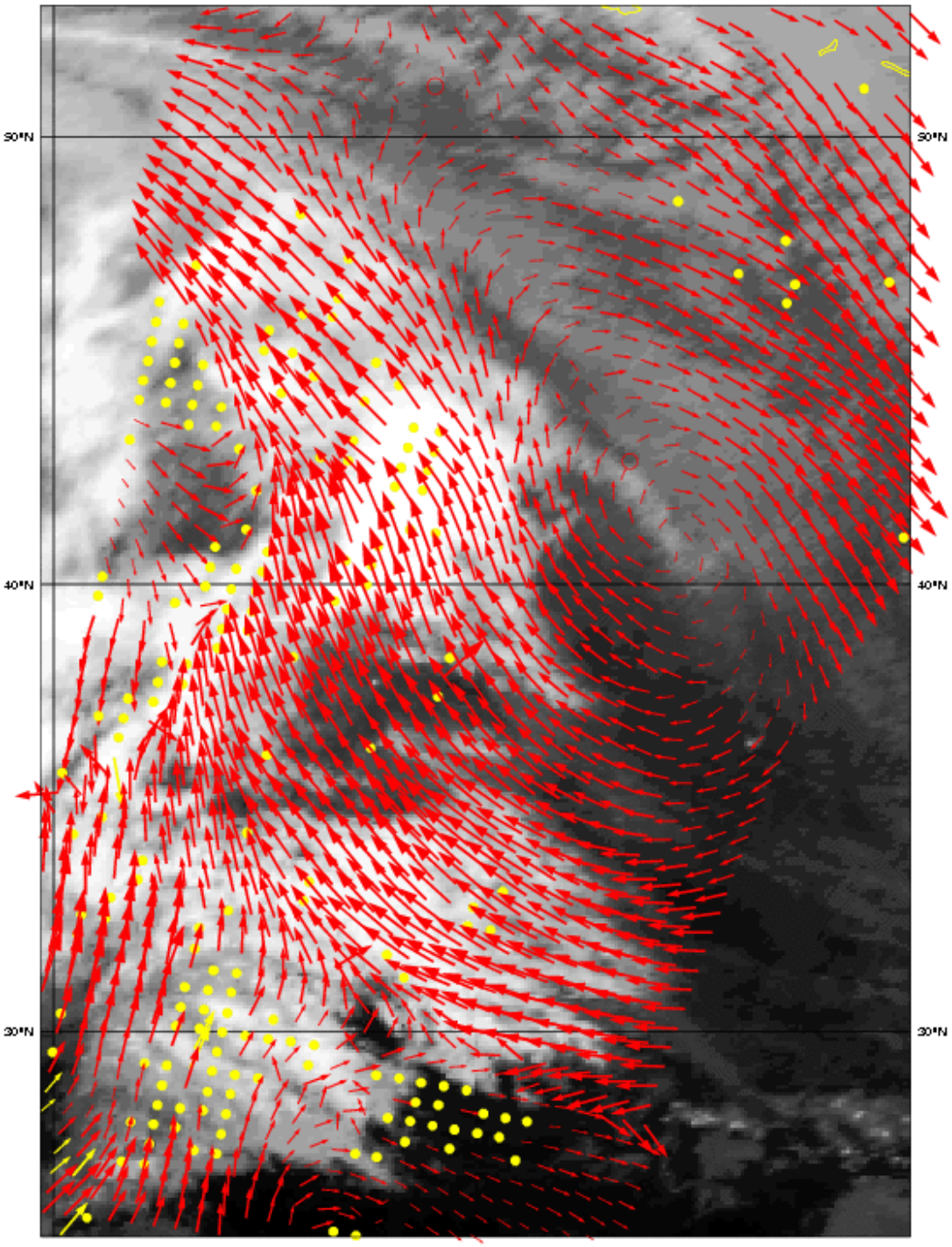


Figure 2.3 SDP wind field retrieved in MSS mode for January 31, 2005, at 25 km resolution, overlaid on an IR satellite image. Only wind arrows 50 km apart are shown. The cold front on the left of the image is clear and sharp. The yellow dots are rejected WVC's, mostly because of rain.

2.2 Modes of using SDP

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

The operational SDP SeaWinds product, available as a deliverable from the NWP SAF project, could be the starting point for NWP assimilation:

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

1) and 2) can be based on SDP in standard or MSS mode, and at various resolution. MSS is somewhat more dependent on the first guess in 2D-VAR than the SDP standard, but much less noisy (see above) A more noticeable advantage is thus obtained by using the local first guess and potentially the full hi-res benefit of the SeaWinds data is achieved. At the moment, the 25-km mode is experimental, since at KNMI we are now objectively evaluating the added value of MSS and 2DVar at 25 km. Please contact the NWP SAF helpdesk if this mode will be implemented (address: http://www.metoffice.com/research/interproj/nwpsaf/) The mode of using SDP thus depends on the opportunities, experience, and time the user has to experiment with SeaWinds in the NWP system under consideration.

The SDP program can, of course, also be used to create a stand-alone wind product. Such a stand-alone SeaWinds wind product is a deliverable of the OSI SAF project. More information on this project can be found at the project web site, http://www.osi-saf.org/index.php.

2.3 Installing SDP

SDP is written in Fortran 90 (with a few low level modules in C) and is designed to run on a modern computer system under LINUX or UNIX. SDP needs a Fortran 90 compiler and a C compiler for installation. SDP comes along with a complete make system for compilation. The makefile contains installation scripts which are written in Bourne shell to enhance portability. When compiled, SDP requires about 60 Mb disk space.

In principle, SDP may also run under Windows. However, SDP needs the BUFR library from ECMWF, and this poses some restrictions on the systems supported. Under Windows one must use a (free) UNIX emulator like Cygwin (see http://www.cygwin.com/ for more information and download) or MinGW/MSYS (http://www.mingw.org/).

To install SDP, the following steps must be taken:

1. Copy the SDP package (file `SDP_1.3.tar.gz`) to the directory from which SDP will be applied, and unzip and untar it. This will create subdirectories `SDP` and `genscat` that contain all code needed (see 2.3.1).
2. Download the ECMWF BUFR library file `bufr_000310.tar.gz` (or another version not earlier than 000240) and copy it to directory `/genscat/support/bufr`. See also 2.3.3.
3. Go to the work directory (the one above directories `SDP` and `genscat` and enter InstallSDP. The script will ask for the compiler used and invoke the make system fot compilation and linking of the software (see also 2.3.4).

SDP is now ready for use, provided that the environment variables discussed in section 2.3.2 have the proper settings. See also 2.4 and 2.5.

2.3.1 Directories and files

All code for SDP is stored in a file named `SDP_1.3.tar.gz` that is made available in the framework of the NWP SAF project. This file should be placed in the directory from which SDP is to be run. After unzipping (with `gzip -df SDP_1.3.tar.gz`) and untarring (with `tar -xf SDP_1.3.tar`), the SDP package is extracted in subdirectories `SDP` and `genscat`, which are located in the directory where the original file `SDP_1.3.tar.gz` was located. Subdirectories `SDP` and `genscat` each contain a number of files and subdirectories. A copy of the release notes and the script `InstallSDP` can also be found in the directory containing `SDP` and `genscat`.

Tables 2.1 and 2.2 lists the contents of directories `SDP` and `genscat`, respectively, together with the main contents of the various parts.

Name	Type	Contents
`data`	subdirectory	Look Up Table for the SeaWinds Geophysical Model Function (GMF)
`docs`	subdirectory	Documentation, including this document
`exec`	subdirectory	Shell scripts for running SDP with various input options
`makefile`	file	Makefile for compiling SDP under LINUX or UNIX
`python`	subdirectory	Python scripts for running SDP under various operating systems
`readme.txt`	file	Readme file with some information on SDP.
`sdp`	subdirectory	Source code for main SDP program and supporting routines
`sws`	subdirectory	Source code for SeaWinds dependent routines
`test`	subdirectory	Example BUFR input and output files for testing purposes.

Table 2.1 Contents of directory `SDP`.

Name	Type	Contents
ambrem	subdirectory	Source code for ambiguity removal routines
ambrem/twodvar	subdirectory	Source code for KNMI 2DVar ambiguity removal routines
inversion	subdirectory	Source code for inversion routines
main	subdirectory	Dummy subdirectory to facilitate the make system
Makefile	file	Makefile for compiling Gencat
Objects.txt	file	Part of the makefile
Readme.txt	file	Readme file with some information on genscat
Set_Makeoptions	script file	Script needed by the make system.
support	subdirectory	Collection of general purpose routines sorted in subdirectories
support/BFGS	subdirectory	Source code for minimization routines needed in 2DVar
support/bufr	subdirectory	BUFR tables (in subdirectories) and source code for BUFR file handling routines
support/datetime	subdirectory	Source code for date and time conversion routines
support/file	subdirectory	Source code for file handling routines
support/multifft	subdirectory	Source code for FFT routines needed in minimization
use_g95	script file	Script for choosing the GNU g95 Fortran compiler
use_gfortran	script file	Script for choosing the GNU-GCC compiler collection
use_ifort	script file	Script for using the Intel Fortran compiler
use_pgf90	script file	Script for using the Portland Fortran compiler

Table 2.2 Contents of directory genscat.

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

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

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

2.3.2 Environment variables

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

The PLATFORM variable depends on the operating system used. It should be set to big_endian under IRIX and SUN OS, and to little_endian under LINUX, OSF, and Windows. The PLATFORM variable is needed to guide SDP to the correct version of the lookup table containing the K_u-band Geophysical Model Function (GMF) needed for the inversion. These tables are in binary form, and the various operating systems have different representations of binary data.

Name	Value(s)
PLATFORM	big_endian little_endian
BUFR_TABLES	genscat/support/bufr/bufr_tables/
LUT_FILENAME_KU_HH	SDP/data/${PLATFORM}/nscat2_250_73_51_hh.dat
LUT_FILENAME_KU_VV	SDP/data/${PLATFORM}/nscat2_250_73_51_hh.dat
EXP_MLE_FILENAME	Depends on resolution, see table 2.4

Table 2.3 Environment variables for SDP.

The BUFR_TABLES variable guides SDP to the BUFR tables needed to read the input and write the output.

The variables LUT_FILENAME_KU_HH and LUT_FILENAME_KU_VV point SDP to the correct K_u-band GMF lookup tables at HH and VV polarization, respectively. Note that these variables contain the PLATFORM variable already discussed.

Resolution (m)	Value of **EXP_MLE_FILENAME**
25	SDP/data/${PLATFORM}/mean_bufr_1r_mle_knmi9_25_r5_mm.dat
50	SDP/data/${PLATFORM}/mean_bufr_1r_mle_knmi9_50_r5_mm.dat
100	SDP/data/${PLATFORM}/mean_bufr_1r_mle_knmi9_100_r5_mm.dat

Table 2.4 Values of variable EXP_MLE_FILENAME for various resolutions.

The EXP_MLE_FILENAME variable points SDP to a lookup table containing the mean MLE's as a function of node number and wind speed [*Portabella*, 2002]. The mean MLE's are needed for quality control, see section 4.4.3. These LUT's are resolution dependent, so the value of EXP_MLE_FILENAME must agree with the resolution specified in the command line options of SDP (see section 2.3). The possible values of EXP_MLE_FILENAME are shown in table 2.4.

2.3.3 Installing BUFR library

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

Directory genscat/support/bufr contains the shell script make.bufr.lib, which unzips, untars, and compiles the BUFR library file downloaded from ECMWF. This script is part of the genscat make system and is automatically invoked when compiling genscat. The current version assumes BUFR version 000310, but later versions (or earlier, but not earlier than 000240) can be used if the reference to file bufr_000310 is set to the appropriate file name in scripts make.bufr.lib and make.clean.bufr.lib, that are both located in directory genscat/support/bufr.

BUFR file handling at the lowest level is difficult to achieve. Therefore some routines were coded in C. These routines are collected in library BUFRIO (see also section 7.4). Its source code is located in file `bufrio.c` in subdirectory `genscat/support/bufr`. Compilation is done within the genscat make system and requires no further action from the user (see 2.3.4).

2.3.4 Compilation and linking

Compilation and linking of SDP under LINUX or UNIX is done in three steps by the script `InstallSDP`:

1. Set the compiler environment variables according to the choice entered on request. This is equivalent of running the appropriate `use_*` scripts in directory `genscat;`
2. Go to directory `genscat` and invoke the make system;
3. Go to directory `SDP` and invoke the make system to produce the executable `sdp` in directory `SDP/sdp`.

Before activating the make system, some environment variables identifying the compiler should be set. These variables are listed in table 2.5. The environment variables in table 2.5 are set by the script `InstallSDP`, but can also be set by using one of the `use_*` scripts located in directory `genscat.` Table 2.6 shows the properties of these scripts. The scripts are in Bourne shell (extension `.bsh`) and in C shell (extension `.csh`). Note that all scripts select the GNU gcc C compiler.

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

Table 2.5 Environment variables for compilation and linking.

Script	Fortran compiler	C compiler	Remarks
use_g95	g95	gcc	GNU compilers by A. Vaught
use_gfortran	gfortran	gcc	GNU-GCC 4.0 compiler collection
use_ifort	ifort	gcc	Intel Fortran compiler
use_pgf90	g90	gcc	Portland Fortran compiler

Table 2.6 Properties of the four `use_*` scripts.

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

If the user wants to use a Fortran or C compiler not included in table 2.6, he can make his own version of the `InstallSDP` or `use_*` script, or include the environment variables for compilation and linking in his startup file. The user must extend

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

The `Makefile` refers to each subdirectory of genscat, invoking execution of the local `Makefile` and, in cases where a subdirectory contains code as well as a subdirectory containing code, `Makefile_thisdir`. The makefiles need supplementary information from the files `Objects.txt` which are present in each directory containing code. The settings for the compilers are located in file `Makeoptions` in directory `genscat`. This file is generated by the Bourne shell script `Set_Makeoptions` which is called automatically by the genscat make system. The local `Makefile` in subdirectory `genscat/support/bufr` calls the script `make.bufr.lib` for compilation of the BUFR library (see 2.3.3). It also contains the Fortran program `test_modules` that generates the binary BUFR tables B and D from the ASCII tables already present, and is executed automatically by the make system. Program `test_modules` can also be used to test the genscat BUFR module, see 2.7. The `Makefile` in subdirectory `genscat/support/bufr/bufr_tables` calls the shell scripts `run_make_symlinks_for_first_table` and `run_make_all_needed_symlinks_for_SEAWINDS`. These scripts make copies of the generic binary BUFR tables B and D under different names. There are four different naming conventions in BUFR version 000240 to 000280, and binary files are generated for each of them. The copies could be replaced by symbolic links to save disk space, but this is not guaranteed to work on each system (symbolic links are not understood by Cygwin under Windows XP). Further information on the make system is given in the inline comments in the scripts and makefiles.

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

The GMF tables in `SDP/data` are set to read-only. Some systems (e.g. Cygwin) require write permission for properly reading those tables. This should be done separately using the command `chmod u+w` in the appropriate subdirectory.

When recompiling (part of) SDP or genscat with the make system, for instance when installing a new version of the BUFR library, one should be sure that the proper environment variables for compilation and linking are set. To recompile all of the software enter `InstallSDP again`. To recompile part of the software invoke the make system where needed. Don't forget to rerun the `use_*` commands to select the right compiler.

2.4 Command Line Options

The SDP program is started from directory `SDP/sdp` with the command

`sdp [options] < -f BUFRfile | -fl FileList >`

with `<>` indicating obligatory input, `[]` indicating obligatory input, and | indicating alternatives. The following command line options are available:

`-f <BUFRfile>` Process a single BUFR input file with name `BUFRfile`.
The BUFR input file should have the NOAA format.
Example: `sdp -f QS_D02001_S0006_E0120_B1320303` will process this file. The results will be written on a file with the name `QS_D02001_S0006_E0120_B1320303~`. In general, each output BUFR file has the same name as the corresponding input file, but with a tilde attached. Either this option or the next one is obligatory.

`-fl <FileList>` Process a list of BUFR input files in the file named `FileList`. Either this option or the previous one is obligatory

`-mss` Use the Multiple Solution Scheme for Ambiguity Removal.
If the Multiple Solution Scheme (MSS) is switched on, SDP internally works with 144 different solutions for the wind vector. If MSS is switched off, SDP calculates four solutions at most. MSS is switched off as default.

`-genericws <n>` Produce a second BUFR file in generic wind section format.
This option generates a second BUFR output file in the KNMI generic wind section format not yet approved by the WMO. The number `n` specifies the number of wind vector solutions written in the output file. The number `n` should not exceed 144. The name of the output file is the same as that of the input file, but with an extension `~.genws`.
Example: the command `sdp -genericws 144 -f QSExample` (without MSS switched on) will produce a second output file with name `QSExample~.genws`. However, this file contains only 4 wind solutions at most because MSS is switched off by default. The other 140 solutions are set to missing.
Without MSS switched on, it is more appropriate to set `n` equal to 4.

`-resol <i>` Select output resolution.
The output resolution is controlled by a single resolution index `i`, with `i` an integer from 0 to 15. The least significant two bits define the across track in multiples of 25 km, and the next two bits similarly define the along track resolution (see table 2.7). The default value is 15, i.e. 100 km resolution.

Index	Resolution in km		Implemented?
	Across track	Along track	
0	25	25	Yes (not yet validated)
1	50	25	No
3	100	25	No
4	25	50	No
5	50	50	Yes (not yet validated)
7	100	50	No
12	25	100	No
13	50	100	No
15	100	100	Yes (default)

Table 2.7 Resolution specification.

Example: the command `sdp -f QSExample -resol 0` will process the SeaWinds file `QSExample` with a resolution of 25 km along track and across track. The results are written on file `QSExample~`.
Warning: the environment variable `EXP_MLE_FILENAME` should have the correct value corresponding to the resolution set with the `-resol` command.

`-qdp` Processing in QDP mode.
This mode of operation is equal to the old QDP scheme. MSS is switched off and the inversion uses no parabolic fitting to find the minimum in the cost function when determining the wind direction. The resolution is set to 100 km (resolution index 15).

`-nwp` Sort the output in assimilation windows.
This option is inactive as sorting in assimilation windows is not yet implemented.

`-anawin <3h|6h>` Specify the temporal size of the assimilation window (analysis period).
The length of the analysis period can be either 3h or 6h. This option is inactive as sorting in assimilation windows is not yet implemented.

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

-noamb Switch off ambiguity removal (default switched on).
This option is useful when is run in MSS mode, and selection of the scatterometer wind is left to the data assimilation procedure of the Numerical Weather Prediction model. In other words: the NWP model is fed with a large number of solutions and their probability, and finds the best value when comparing with other data sources. This avoids too large influence of the NWP model. Such a procedure will be implemented for KNMI's HIRLAM.

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

-mon Switch on the monitoring function.
The results are written on a file with the same name as the input file, but with an extension `.mon` added. As default no monitoring file is produced.

-mononly Write the monitoring file without any processing.
The command `sdp -mononly` has the same effect as the command `sdp -mon -noinv -noamb –nowrite`.
Warning: the `–qdp` option is switched off by the `–mononly` option.

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

Running the command `sdp` without any command line options will yield the following output on the console:

```
Usage: sdp [options] < -f BUFR file | -fl file list >
with [.] : free options
     <.> : mandatory options
      |  : choice between alternatives

Options:

-f <BUFRfile>  - process file named BUFRfile
-fl <Filelist> - process list of BUFR files in Filelist
-mss           - use Multiple Solution Scheme MSS
-genericws <N> - write second BUFR file with generic wind section
                 containing N wind solutions
-resol <I>     - set resolution index to value I
-qdp           - process in QDP mode
-nwp           - sort output in assimilation windows (inactive)
```

```
-anawin <3|6>  - set size of assimilation window to 3 or 6 hours (inactive)
-noinv         - switch off inversion
-noamb         - switch off ambiguity removal
-nowrite       - do not produce BUFR output
-mon           - switch on monitoring
-mononly       - write monitoring info without processing
-verbosity <L> - set verbosity level to L
```

Running the command `sdp` with an illegal option *`Illegal`* will produce the same output, but preceded by the error message:

```
Invalid option Illegal
```

2.5 Scripts

Directory `SDP/execs` contains four Bourne shell scripts for running SDP with specific input options and the correct environment variables. The reader is referred to the scripts themselves to find out their use and operation.

Directory `SDP/python` contains Python scripts for execution of the SDP program on different platforms (e.g., Linux, SGI, and Sun). The main goal of these scripts is to test the operation of the program.

A dedicated Python package called `seawindspy` contains support data and procedures, see the folder `SDP/python/seawindspy`. For example, it contains the Python Classes `SdpClass` and `QdpClass` to operate SDP or QDP in Python scripts. Python is a freeware object-oriented programming language. It can be obtained from www.python.org.

It is recommended to use shell scripts for running SDP to avoid errors caused by conflicting values of environment variables and command line options.

2.6 Testruns

Directory `SDP/tests` contains four BUFR files for testing the SDP executable.. File `QS_D02001_S0006_E0120_B1320303` is an input file for SDP. Files `SDP_Testrun_1`, `SDP_Testrun_2` ,and `SDP_Testrun_3` are SDP output files for the runs specified in table 2.8.

Copying file `QS_D02001_S0006_E0120_B1320303` to directory `SDP/execs` and running one of the commands of table 2.8 will yield a BUFR output file with the default name `QS_D02001_S0006_E0120_B1320303~` which should contain the same results as one of the three `SDP_Testrun` files, depending on which command is applied.

Command	Result identical with
`sdp_025 -f ../tests/QS_D02001_S0006_E0120_B1320303`	`SDP_Testrun_1`
`sdp_025 -f ../tests/QS_D02001_S0006_E0120_B1320303 -mss`	`SDP_Testrun_2`
`sdp_qdp -f ../tests/QS_D02001_S0006_E0120_B1320303`	`SDP_Testrun_3`

Table 2.8 SDP testruns.

Figure 2.4 shows the global coverage of the testrun. SeaWinds covered part of the Indian Ocean southeast of India, part of the Barentz Sea north of Scandinavia, small parts of the Hudson Bay, the Great Lakes, and the Gulf of Mexico, and a large strip in the Pacific west of South America. The colors indicate the magnitude of the wind speed as indicated by the legendum. Figure 2.4 shows the results of testrun number 2, but the two other testruns will yield very similar results for the magnitude of the wind speed. More information on these tests (and other tests) is given in the SDP Test Report [*SCAT group,* 2005].

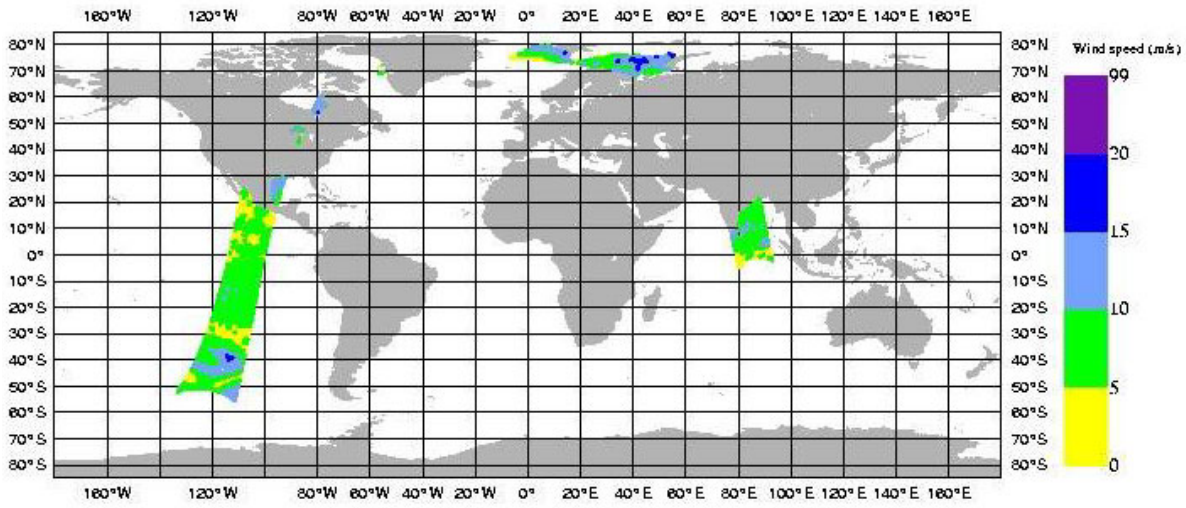


Figure 2.4 Global coverage of the testruns. Wind speed results for testrun 2 are shown.

Due to round-off differences, a simple file comparison may not be appropriate to test the SDP output. It is then necessary to decode the BUFR files and compare the retrieved wind field with the one in the `SDP_Testrun` file. BUFR decoding software is not part of the SDP package, but may be obtained from KNMI if requested. See also below.

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

```
test_modules [BUFRinput]
```

where `BUFRinput` is the BUFR input file.

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

Subdirectories `convert`, `num`, `file` and `datetime` of `genscat/support` contain test programs for the module in that subdirectory. The test programs write their result to the standard output. For comparison, a copy of the output is contained in the `.output` files. Table 2.9 gives an overview of the genscat test programs.

Directory	Program name	Output file	Remarks
genscat/support/bufr	test_modules	testwriting.bufr	Part of make system
genscat/support/convert	test_convert	test_convert.output	Wind speed conversion
genscat/support/datetime	TestDateTimeMod	TestDateTimeMod.output	Date and time conversion
genscat/support/file	TestLunManager	TestLunManager.output	File management
genscat/support/numerics	test_numerics	test_numerics.output	Numerical issues

Table 2.9 Test programs in genscat.

2.7 Documentation

Directory `SDP/docs` contains some documentation on SDP, including this document and the Test Report. Further information can be found in the `readme.txt` files, and in the comments in scripts, makefiles and source code.

Chapter 2

SDP product specification

3.1 Purpose of program SDP

The SeaWinds Data Processor (SDP) program has been developed to fully exploit σ_0 data from the SeaWinds scatterometer instruments on either the QuikScat or Adeos-II (Midori-II) satellites to generate surface winds. SDP may be used real-time. The main application of SDP is to form the core of an Observation Operator for SeaWinds Scatterometer data within an operation Numerical Weather Prediction System.

Program SDP is also a level 2 data processor. It reads data from the NOAA `SWS\_met` product, see [*Leidner et al.*, 2000]. SDP applies improved algorithms for inversion, Quality Control, and Ambiguity Removal at various spatial resolutions. These methods are mainly developed and published by KNMI. The output of SDP is again a BUFR file.

3.2 Output specification

The wind vectors generated by SDP represent the instantaneous mean surface wind at 10 m anemometer height in a 2D array of Wind Vector Cells (WVC's) with specified size (optionally 100×100 km^2, 50×50 km^2, or 25×25 km^2). These WVC's are part of the ground swath of the instrument and are numbered with revolution numbers, along-track row numbers, and across-track node numbers. Therefore, every WVC is identified by a unique (lat, lon, time) triple or a unique (revolution number, row number, node number) triple.

In conventional mode, the wind output for every WVC consists of up to 4 ambiguities (wind vector alternatives, with varying probabilities). The selected wind vector is indicated by a

selection flag. For every WVC additional parameters are stored. These are e.g.: (latitude, longitude, time) information, (revolution, row, node) numbers, background wind vector, cell quality flag, and information on the scatterometer beams including σ_0 and K_p data. The output file is structured according to the conventions of the `SWS\_met` input product (NOAA format).

A second output file is produced if the `genericws` option is switched on. This file is in the so-called Generic Wind Section format and contains up to 144 wind vector solutions and their probabilities. This format is not yet approved by the WMO.

3.3 Input Specification

Input of SDP is the SeaWinds Scatterometer Near-Real-Time BUFR Geophysical Data Product, or shortly the `SWS\_Met` Data Product. This product is created by NOAA. Though it is in fact already a level 2 product in itself, it should be stressed here that only the basic level 1 data from this NOAA product are used as input for the SDP program.

For SeaWinds on QuikSCAT the data have now been available for several years. It contains WVC-composite σ_0 data based on slices of the scatterometer pulse footprint. Details of this product can be found in [*Leidner et al.*, 2000].

Unfortunately, the Adeos-II satellite collapsed after 9 months of operation. A similar data product is not (yet) available for this satellite.

Remarks:

- At KNMI, the data are gathered in a daily archive file. These `SWS\_met` files are stored in the MOS system.
- At ECMWF, the MARS system contains `SWS\_met` data stored in 6-hourly BUFR files. These files are also suitable as input for the SDP program.

3.4 System requirements

Table 3.1 shows the platform and compiler combinations for which SDP has been tested. However, the SDP program is designed to run on any UNIX (LINUX) based computer platform with a Fortran compiler and a C compiler. The equivalent of a modern personal computer will suffice to provide a timely NRT wind product. SDP requires about 80 MB disk space when installed and compiled.

Platform	Fortran compiler	C compiler
Suse LINUX work station	Portland pgf90 GNU g95	GNU gcc
SunOS UNIX	Sun Fortran	GNU gcc
Windows XP PC with Cygwin	GNU g95	GNU gcc

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

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

3.5 Details of functionality

3.5.1 BUFR IO and coding

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

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

Appendix A contains a listing of the data descriptors of the BUFR data input and the BUFR data output of the SDP program in the SWS_met BUFR product format (NOAA format). For more details on BUFR and the SWS_met BUFR product, the reader is referred to [*Dragosavac*, 1994; *Leidner et al.*, 2000].

ECMWF maintains a library of routines reading (writing) and decoding (encoding) the binary BUFR messages. This library forms the basis of the genscat BUFR module and hence the SDP program BUFR interface, see Chapter 7.

3.5.2 Output resolution

An important feature of the SDP program is that it may produce a level 2 wind product on different resolutions. Of course, there is a trade off between the output resolution and the statistical error of the mean wind vectors. Therefore KNMI has developed a SeaWinds product with 100 km resolution for assimilation in most NWP models. However, a different resolution may be optimal for a specific NWP application. The statistical error of the wind vectors for the higher resolutions is currently a topic of further testing.

3.5.3 Quality Control

The quality of every WVC is controlled. An import aspect is the contamination of the K_u-band scatterometer signals by rain. The rain flag used in the SDP program is based on the value of the normalized maximum likelihood estimator (MLE) [*Portabella and Stoffelen,* 2001, 2002]. Compared to the JPL flag, the KNMI flag accepts more non-rain winds between 10 and 20 m/s that occur in meteorologically dynamic areas. It also yields less tropical rain contaminated winds [*Portabella and Stoffelen,* 2001, 2002].

3.5.4 Inversion

In the inversion step of 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 term of wind speed and wind direction) to a σ_0 value. The GMF depends not only wind speed and wind direction but also on the measurement geometry (relative azimuth and incidence angle) and beam parameters (frequency and polarization).

For SeaWinds, a maximum likelihood estimator (MLE) is used to preselect a set of wind vector solutions and associated probabilities that yields the best match with the observed σ_0's. This preselection depends on the number of independent σ_0 values available within the wind vector cell.

The SDP program also includes the Multiple Solution Scheme (MSS). In MSS mode, a much larger preselection of wind vector solutions is produced. The wind vector solutions are ranked according to their probability based on the MLE and constitute the full wind vector probability density function. Subsequently, the 2DVar Ambiguity Removal method, see e.g., section 3.5.5 is applied with a much larger set of wind vector solutions. The output may be written in the so called Generic Wind Section BUFR format, which allows up to 144 wind vector solutions but is still to be approved by the WMO. Details on the KNMI SeaWinds inversion approach can be found in [*Portabella,* 2002]. MSS compares better to an independent NWP model reference than conventional four-solution schemes at 100 km resolution [*Portabella and Stoffelen,* 2004].

Technical information on the KNMI inversion approach can be found in Chapter 5. Details of the original JPL SeaWinds wind retrieval can be found in [*Draper and Long,* 2002].

3.5.5 Ambiguity Removal

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

et al., 2004]. These documents may be downloaded from the EUMETSAT website, www.eumetsat.int, or the KNMI website, www.knmi.nl/scatterometer.

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

Remarks:

- The Fortran implementation of the 2DVar system strongly differs from that of QDP.
- The recent genscat development on ambiguity removal allows the use of the PRESCAT ambiguity removal scheme.

3.5.6 Monitoring

For the automatic ingestion of observations into their NWP systems meteorological centers require quality checks on the NRT products. For the Seawinds BUFR products a monitor flag is developed. This flag indicates that several measures on the level of corruption of the output BUFR files are over a specified threshold. Onset of the flag indicates that the input should be rejected for ingestion by the NWP system. Details on the monitor developed can be found in the NWP SAF document [*de Vries et al.*, 2004], downloadable from the EUMETSAT or KNMI website, www.eumetsat.int or www.knmi.nl/scatterometer, respectively.

3.6 Details of performance

SDP is delivered with a BUFR input file named `QS_D02001_S0006_E0120_B1320303`, which contains half an orbit of data. Table 3.2 gives the approximate times needed for processing this file under various options on a personal workstation with a 2.66 GHz Pentium 4 processor under LINUX using the GNU g95 Fortran compiler.

Script	Resolution (m)	MSS?	Inversion (seconds)	AR (seconds)	BUFR IO (seconds)	Total (seconds)
`sdp_025`	25	No	56	11	5	75
`sdp_025`	25	Yes	58	70	5	136
`sdp_qdp`	100	No	4	16	1	24

Table 3.2 Approximate times needed by SDP to process BUFR file `QS_D02001_S0006_E0120_B1320303` under various input options.

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

The processing times depend only little on the number of WVC's in the orbit being processed. The choice of platform and compiler will generate more variation. Using the Portland pgf90 compiler rather than the GNU g95 compiler in the examples in table 3.2 will result in processing

times that are 50% to 100% larger.

Chapter 4

Program Design

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

4.1 Top Level Design

4.1.1 Main program

The main program, SDP, (file `sdp` in the `SDP/sdp` directory) is a UNIX (LINUX) executable which processes SeaWinds BUFR input files. The main output consists of BUFR files. The output BUFR messages have the same descriptors as the input messages. The user may provide arguments and parameters according to UNIX command line standards. The purpose of the different options is described in the User Manual (chapter 2).

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

The first step is to process the arguments given at the command line. Next, the SDP program loops over the input files specified in the arguments. For every input file the BUFR messages are read and mapped onto the SeaWinds data structure, see e.g., subsection 4.1.3. As part of the preprocessing a similar SeaWinds data structure is created for the output. Subsequently, the

output data structure is filled with level 1 (σ_0-related) data. The next steps are the inversion and the ambiguity removal. These steps are performed on the output data. The loop over the input files ends with the post-processing step (which includes some conversions and the monitoring) and the mapping of the output data structure onto BUFR messages of the BUFR output file. The different stages in the processing correspond directly to specific modules of the code. These modules form the process layer, see section 4.4.

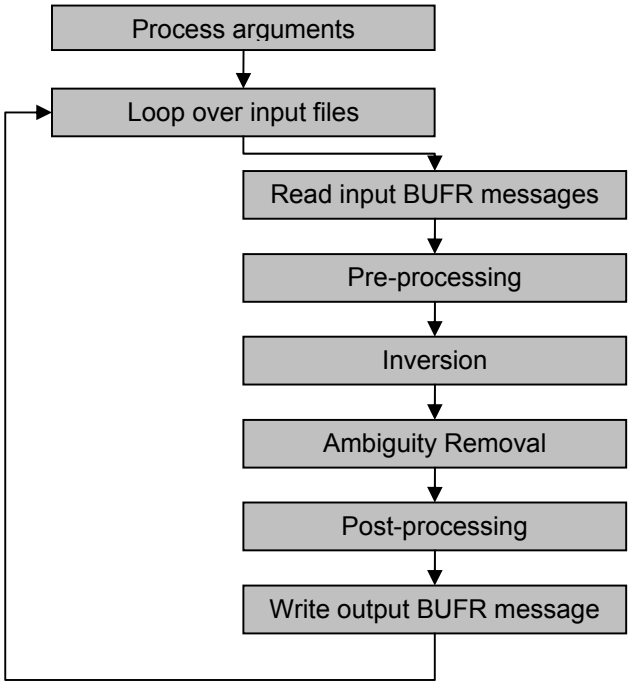


Figure 4.1 Baseline of the Seawinds Data Processor

4.1.2 Layered model structure

SDP is a Fortran90 program consisting of several Fortran90 modules which are linked after their individual compilation. The SPD program is set up from three layers of software modules, see Figure 4.2. The purpose of the layer structure is to divide the code with respect to its genericity. Details on the individual modules can be found in sections 4.2 to 4.4.

The first layer (the process layer) consists of five modules which serve the main steps of the process. These steps are:

1) BUFR input and output;
2) pre- and post-processing;
3) inversion;
4) ambiguity removal;
5) support.

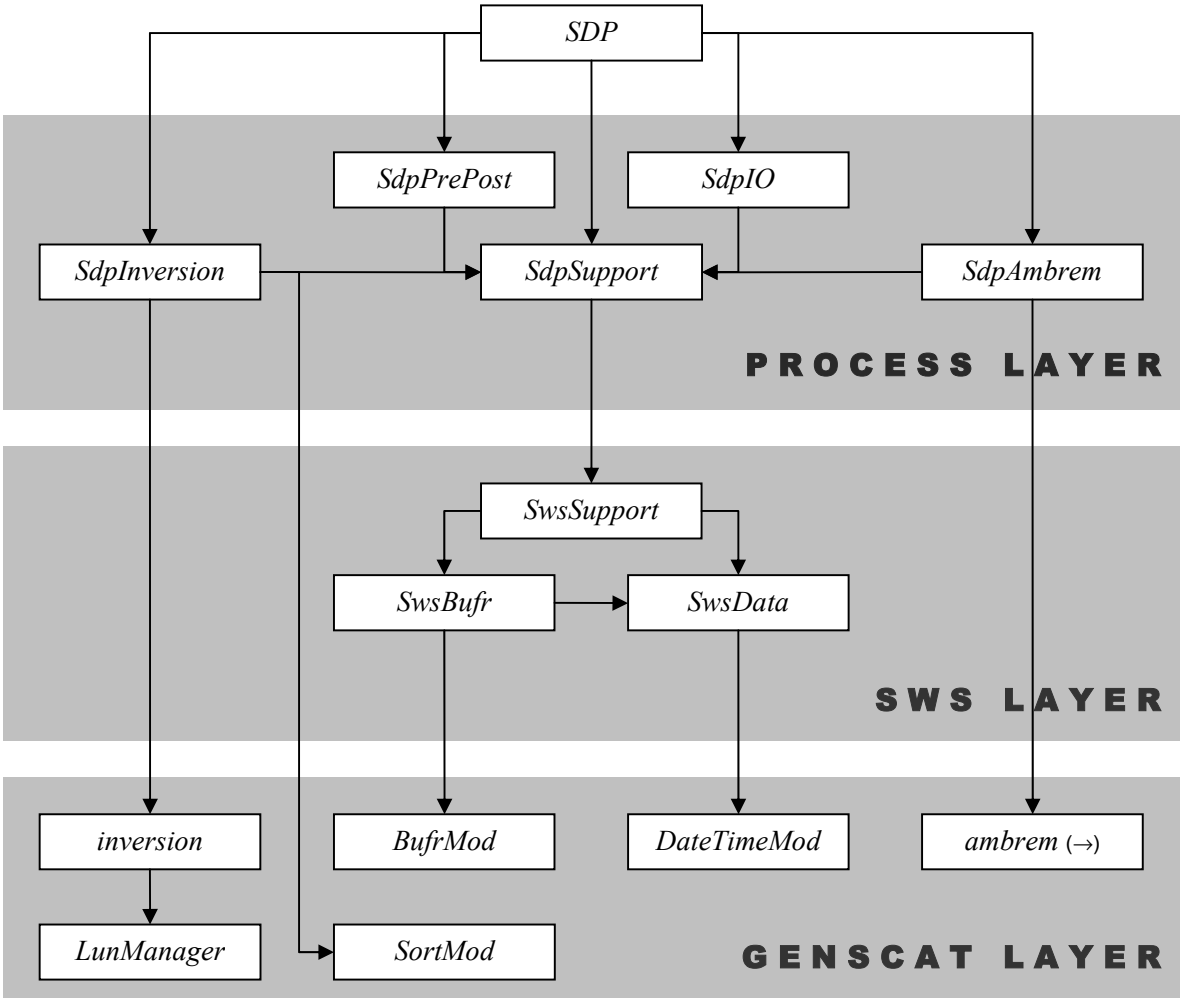


Figure 4.2 Module layer and top level module dependencies. The dependencies for module *ambrem* are continued in figure 6.1

Module name	Tasks	Comments
SdpIO	BUFR file handling	
	Command line processing	
SdpPrePost	Spatial averaging	Averaging to 50 m or 100 m resolution
	Quality control	Usability of input data
	Rain flagging	Rain flag based on normalized MLE
	Scale conversion	Linear versus logarithmic
	Monitoring	Monitoring
SdpInversion	Inversion	Interface to `genscat/inversion`
SdpAmbrem	Ambiguity Removal	Interface to `genscat/ambrem`
SdpSupport	Support for processing	Definition of data structures
		Interface to `genscat/support` via *SwsSupport*

Table 4.1 SDP process modules.

Each module contains code for performing one or more of the specific tasks. These tasks are shortly described in Table 4.1. A more elaborate description is given in section 4.4. The last module listed, *SdpSupport* is a general support module. This module is used by the other four modules of the process layer for the inclusion of definitions of the data structures and the support routines. (Note that the names of the process modules start with the prefix *Sdp* while the source code is stored in the subdirectory with the name `sdp`).

The second layer (the SeaWinds layer) consists of SeaWinds Data Support modules. These modules, see table 4.2, contain the SeaWinds data structure definitions and the interface between these data structures and the (input/output) BUFR data format. The key module is *SwsData*. This module contains all the important data types that are introduced for the processing. An overview of these data structures is given in subsection 4.1.3. Details on the actual types and routines are given in section 4.3. The names of these modules start with the prefix *Sws*. The *Sws*-modules are stored in the subdirectory `SDP/sws`.

Finally, the third module layer is the genscat layer. The genscat module classes (i.e., groups of modules) used in the SDP program are listed in table 4.3. genscat is a set of generic modules which can be used to assemble processors as well as pre-, and post-processing tools for different scatterometer instruments available for the user community. A short description of the main (interface) modules is given in section 4.2. The most important classes of modules are related to the inversion processing step (chapter 5), the Ambiguity Removal step (chapter 6), and the BUFR file handling (chapter 7). The genscat modules are located in subdirectory `genscat`.

Module name	Tasks	Description
SwsBufr	BUFR handling	Mapping of BUFR messages on SeaWinds data structure
SwsData	Data definitions,	Composed type declarations
	Data quality control	Checking and flagging
SwsSupport	Processing support	Interface to `genscat/support` routines

Table 4.2 SeaWinds data support modules.

Module class	Tasks	Description
Ambrem	Ambiguity Removal	2DVar and other schemes, see chapter 6
Inversion	Wind retrieval	Inversion in one cell, see chapter 5
Support	BUFR support	*BufrMod*, based on ECMWF library
	FFT, minimization	Support for 2DVar
	Error handling	Print error messages
	File handling	Finding, opening and closing free file units
	Conversion	Conversion of meteorological quantities
	Sorting	Sorting of ambiguities to their probability
	Date and time	General purpose

Table 4.3 genscat module classes.

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 and ASCAT data processing.

The layer set-up facilitates a fast and comprehensive development of pre- and post-processing functionality without interfering with the code of the processor itself. In fact, the SeaWinds support layer does not contain any real processing functionality, but this layer provides the required functionality to develop applications which only need the input or output of the process.

4.1.3 Data Structure

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

The main constituent of the *SwsRowType* is the cell data structure, *CellType*, see figure 4.3. Since most of the processing is done on a cell-by-cell basis the *CellType* is the pivot data structure of the processor. The level 1 data of a cell are stored in a data structure called *BeamType*.

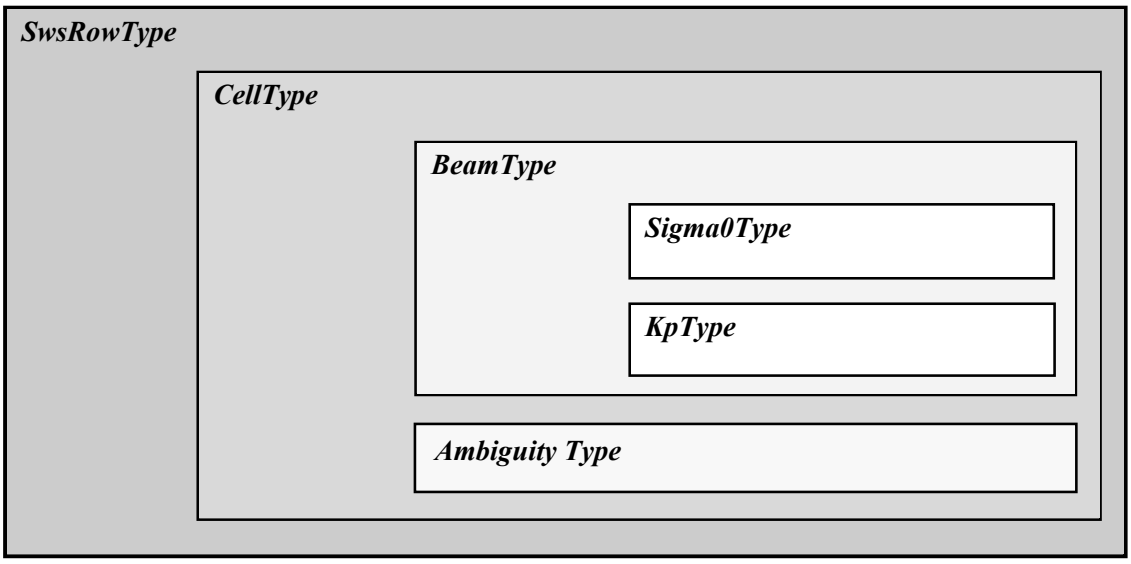


Figure 4.3 Schematic representation of the nested data definitions in the *SwsRowType* data structure.

Every cell contains 4 instances of the *BeamType,* corresponding to the inner fore and aft beams and the outer fore and aft beams. The *BeamType* is further subdivided in the *Sigma0Type* containing σ_0-related data and the *KpType*. The latter contains the σ_0 variance coefficients.

A cell may also contain an array of instances of the *AmbiguityType* 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 chapter 6.

Remarks:

- In QDP the input and output array structure are called *obs* and *obs2*, respectively. In SDP, this naming convention is reused by giving the input instances of *CellType* the name *cll* and the output instances of *CellType* the name *cll2*.

4.1.4 Quality flagging and error handling

Important aspects of the data processing are to check the validity of the data and to check the data quality. In the SDP program two WVC flags are set for every WVC, see table 4.4, and three flags are set for each of the four beams, see table 4.5. Therefore, 14 flags in total report on the quality and other aspects of the data in each WVC. Furthermore, 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 SDP 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
Quality Flag	Quality checking	In BUFR output
Process Flag	Range checking	Not in BUFR output

Table 4.4 Flags for every WVC (attributes of *CellType*).

Flag	Tasks	Description
Surf Flag	Check surface condition	In BUFR output
Mode Flag	Check mode	In BUFR output
Qual Flag	Check quality	In BUFR output

Table 4.5 Flags for every beam (attributes of *Sigma0Type*).

4.1.5 Verbosity

Every routine in a module may produce some data and statements for the log of the processor. To control the size the log, several modules contain parameters for the level of verbosity. The verbosity of the SDP program may be controlled by the verbosity command line option *verbosity.*

In general, there are three levels of verbosity specified:

≤ -1: be quiet as possible;
0: only report top level processing information;
≥ 1: report additional information.

Of course, errors are logged in any case. Table 4.6 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*
SwsBufr	*BufrVerbosity*

Table 4.6 Verbosity parameters.

4.2 Module Design for genscat layer

4.2.1 Module *inversion*

The module *inversion* contains the *genscat* inversion code. It is located in subdirectory `genscat/inversion`. Details of this module are described in chapter 5. In the SDP program, the inversion module is only used in the *SdpInversion* module, see subsection 4.4.4.

4.2.2 Module *ambrem*

The module *ambrem* is the main module of the genscat Ambiguity Removal code. It is located in subdirectory `genscat/ambrem`. Details of this module are described in chapter 6. In the SDP program, the *ambrem* module is only used in the *SdpAmbRem* module, see subsection 4.4.5.

4.2.3 Module *Bufrmod*

Genscat contains several support modules. In particular, the *BufrMod* module is the Fortran90 wrapper around the BUFR library used for BUFR input and output. It is located in subdirectory `SDP/genscat/support/bufr`. Details of this module are described in chapter 7. In the SDP program, the *BufrMod* module is only used in the *SwsBufr* module, see subsection 4.3.2.

4.2.4 Support modules

Subdirectory `genscat/support` contains more support modules besides *Bufrmod*. The KNMI 2DVar Ambiguity Removal method requires minimization of a cost function and numerical Fourier transformation. These routines are located in subdirectories `BFGS` and `multiFFT`,

respectively, and are discussed in more detail in section 6.4.

Subdirectory `convert` contains module *convert* for the conversion of meteorological and geographical quantities. So far, only routine *uv_to_sd* is used by module *AmbremBGclosest*, but this may change in future updates of SDP.

Subdirectory `datetime` contains module *DateTimeMod* for date and time conversions. SDP only uses routines *GetElapsedSystemTime* (for calculating the running time of the various processing steps) and *julian2ymd* (for conversion of Julian day number to 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. SDP uses only routines *get_lun* and *free_lun* (for opening and closing, respectively, of a logical unit) in the genscat routine *calc_sigma0* (see figure B1.4).

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

Subdirectory `sort`, finally, contains module *SortMod* for sorting the wind vector solutions according to their probability.

4.3 Module Design for SeaWinds layer

The SeaWinds layer consists of the modules *SwsData, SwsBufr*, and *SwsSupport*. Table 4.7 lists the routines within these modules. A star indicates that the routine is not (yet) called in the processing chain.

4.3.1 Module *SwsData*

The module *SwsData* 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. *WindType*, *TimeType*, and *RainType*. Using these data types (and of course the standard types as integer, real etc.), more complex (composed) data types are derived. Examples are *BeamType*, *AmbiguityType*, *CellType*, and *SwsRowType*. A complete description of all types is given below. The attributes of all these types have intentionally self-documenting names.

Example: the *KpType* has been introduced for the σ_0 variance K_p. The common three coefficients of K_p, i.e., α, β, and γ, are stored for every beam in the Sws_met BUFR messages. The values of these coefficients are copied into an instance of *KpType* (part of *BeamType*), respectively as the real attributes *Alpha*, *Beta* and *Gamma* (see table 4.12).

In the following the different data types are described in alphabetical order.

SwsBufr	***SwsData***		
OpenSwsBufrFile	*MergeRow*	*InitAmbi (*)*	*PrnProcessFlag*
CloseSwsBufrFile	*CheckCell (*)*	*PrintAmbi*	*getCellQualFlagNOAA*
SwsBufrInit	*TestCell*	*InitAntenna*	*getCellQualFlagGen*
ReadSwsBufrData	*InitCell*	*PrintAntenna*	*setCellQualFlagNOAA*
WriteSwsBufrData	*CopyCell*	*InitRain*	*setCellQualFlagGen*
Values2CellNOAA	*PrintCell*	*PrintRain*	*PrnCellQualFlag*
Values2CellGen	*SetDummyCell (*)*	*SetDummyWind (*)*	*getSigma0QualFlag*
Cell2ValuesNOAA	*InitBeam*	*InitWind (*)*	*setSigma0QualFlag*
Cell2ValuesGen	*PrintBeam*	*PrintWind*	*PrnSigma0QualFlag*
	InitSigma0	*TestWind*	*getSigma0ModeFlag*
	TestSigma0	*InitTime*	*setSigma0ModeFlag*
	PrintSigma0	*PrintTime*	*PrnSigma0ModeFlag*
	InitKp	*TestTime*	*getSigma0SurfFlag*
	TestKp	*InitProcessFlag*	*setSigma0SurfFlag*
	PrintKp	*getProcessFlag (*)*	*PrnSigma0SurfFlag*
		setProcessFlag (*)	

Table 4.7 Routines in the genscat layer modules. Routines marked with *(*)* are not needed for SDP. Note that module *SwsSupport* contains no routines..

Ambiguity data: The *AmbiguityType* data type contains information on an individual ambiguity (wind vector solution). The attributes are listed in table 4.8. The routine *InitAmbi()* sets all ambiguity data to missing. The routine *PrintAmbi()* may be used to print all ambiguity data.

Attribute	Type	Description
Wind	*WindType*	Wind vector solution
Error	*WindType*	Error in wind vector solution
Prob	*Real*	Probability of wind vector solution

Table 4.8 Ambiguity data structure.

Antenna data: The *AntennaType* data type contains additional information on the scatterometer beams, see *CellType*. The attributes are listed in table 4.9. The routine *InitAntenna()* sets all antenna data to missing. The routine *PrintAntenna()* may be used to print all antenna data.

Attribute	Type	Description
Num	*Integer*	Beam number
Polarization	*Real (integer)*	Polarization (H or V)
Tb_Mean	*Real (integer)*	Mean brightness temperature
Tb_StdDev	*Real (integer)*	Standard deviation of brightness temperature

Table 4.9 Antenna data structure.

Beam data: Every WVC contains up to 4 beams. The information of every beam is stored in the data type *BeamType*. The attributes are listed in table 4.10. The routine *InitBeam()* sets all beam data to missing. The routine *PrintBeam()* may be used to print all beam data.

Attribute	Type	Description
Num	*Integer*	Beam number: 1 = inner fore, 2 = outer fore, 3 = inner aft, and 4 = outer aft
Sigma0	*Sigma0Type*	σ_0 data
Kp	*KpType*	K_p data
K_Polar	*Real (integer)*	K_p data

Table 4.10 Beam data structure.

Cell Data: The *CellType* data type is a key data type in the SDP program, because many processing steps are done on a cell by cell basis. The attributes are listed in table 4.11. The routine *InitCell()* sets the cell data to missing values. Also the flags are set to missing. The routine *TestCell()* tests the validity of data. This routine sets the cell process flag. The routine *PrintCell()* may be used to print the cell data.

Attribute	Type	Description
RevNr	Integer	Revolution (orbit) number
RowNr	Integer	Row number (along track)
NodeNr	Integer	Node number (across track)
Lat	Real (integer)	Latitude of cell
Lon	Real (integer)	Longitude of cell
Across_Track_Res	Real (integer)	Across track resolution
Along_Track_Res	Real (integer)	Along track resolution
Time_to_Edge	Real (integer)	Time to edge
TimeDiff	Real (integer)	Time difference
Time	*TimeType*	Date and time
Satellite_ID	Integer	Satellite identification
Sat_Motion	Real (integer)	Satellite motion
Instrument_ID	Integer	Instrument identification
GMF_ID	Integer	GMF identification
Software_ID	Integer	Processor identification
Sigma0_In_cell	Integer	Number of beams for cell
Rain	*RainType*	Rain data
Antenna(2)	*AntennaType*	Brightness temperature
Beam(4)	*BeamType*	Beam data σ_0 K_p
Num_Ambigs	Integer	Number of ambiguities
Selection	Integer	
Ambi	*AmbiguityType*	Array of ambiguities
Model	*WindType*	Model wind
EC	*WindType*	ECMWF wind (KNMI)
JPL	*WindType*	JPL wind (KNMI)
TwoDV	*WindType*	2DVar analysis wind (KNMI)
Quality_Flag	*CellQualFlagType*	Quality flag
ProcessFlag	*ProcessFlagType*	Processing flag

Table 4.11 Cell data structure.

NB. The routine *CheckCell()* may be used to select cells with a specified quality. The selection is controlled by a check flag which is an instance of the *CellProcessFlagType*.

***K_p* data**: The error variance of the σ_0 signals are specified in terms of K_p values. K_p values are generally a quadratic approximation in terms of σ_0. The coefficients of this approximation are stored in instances of *KpType*, see table 4.12. The routine *InitKp()* sets the K_p coefficients to missing values. The routine *TestKp()* tests the validity of coefficients specification (see also the cell process flag). The routine *PrintKp()* may be used to print the coefficients.

Attribute	Type	Description
Alpha	Real (integer)	Variance coefficient of quadratic term
Beta	Real (integer)	Variance coefficient of linear term
Gamma	Real (integer)	Variance offset coefficient

Table 4.12 Variance (K_p) data structure.

Normalized Radar Cross Section (σ_0) data: The *Sigma0Type* data type contains the σ_0 (Normalized Radar Cross-Section) information of a specific beam. The attributes are listed in table 4.13. The data types of the flags are discussed further on in this section. The routine *InitSigma0()* sets the σ_0 data to missing values. Also the flags are set to missing. The routine *TestSigma0()* tests the validity of the σ_0 data (see also the cell process flag). The routine *PrintSigma0()* may be used to print the σ_0 data.

Attribute	Type	Description
Lat	Real (integer)	Latitude
Lon	Real (integer)	Longitude
Atten_Value	Real (integer)	Attenuation value
Azimuth	Real (integer)	Azimuth angle
Incidence	Real (integer)	Incidence angle
Value	Real (integer)	σ_0 value
Qual_Flag	*Sigma0QualFlagType*	σ_0 quality flag
Mode_Flag	*Sigma0ModeFlagType*	σ_0 mode flag
Surf_Flag	*Sigma0SurfFlagType*	σ_0 surface flag
Variance_QC	Real (integer)	Variational quality control value

Table 4.13 Signal σ_0 data structure.

Rain data: For every WVC, information on rain is stored in the data type *RainType*. The attributes are listed in table 4.14. The routine *InitRain()* sets all rain data to missing. The routine *PrintRain()* may be used to print all the rain data.

Attribute	Type	Description
MP	Integer	
NOF	Integer	
Rate	Real (integer)	Rain rate
Attenuation	Real (integer)	Attenuation correction

Table 4.14 Rain data structure.

Row data: The data of a complete row of the swath is stored in the data type *SwsRowType*, see table 4.15. The routine *InitRow()* sets all row data to missing. A complete row corresponds to a single BUFR message in the SDP input and output, see module *SwsBufr* in subsection 4.3.4. In some cases two messages are stored for the same row. The routine *MergeRow()* is used to combine the data.

Attribute	Type	Description
RevNr	Integer	Revolution number
RowNr	Integer	Along track row number
NrCells	Integer	Actual number of WVC's
FirstNode	Integer	Node number of first non-empty WVC cell
Cell(76)	*CellType*	Array of cells

Table 4.15 SeaWinds row data structure.

Time data: The *TimeType* data type contains a tuple of 6 integers representing both the date and the time, see table 4.16. The routine *InitTime()* sets the time tuple to missing values. The routine *TestTime()* tests the validity of the date and time specification (see also the cell process flag). The routine *PrintTime()* can be used to print the time tuple.

Attribute	**Type**	**Description**
Year	Integer	19XX or 20XX
Month	Integer	1 – 12
Day	Integer	1 – 31
Hour	Integer	0 – 23
Min	Integer	0 – 59
Sec	Integer	0 – 59

Table 4.16 Time data structure.

Wind Data: The *WindType* data type contains the wind speed and wind direction, see table 4.17. The routine *SetDummyWind()* fills the wind data type with arbitrary values (remark: should use randomization). The routine *InitWind()* sets the wind vector to missing. The routine *PrintWind()* may be used to print the wind vector. The routine *TestWind()* tests the validity of the wind specification, see also the cell process flag.

Attribute	**Type**	**Description**
Speed	Real (integer)	Wind speed
Dir	Real (integer)	Wind direction

Table 4.17 Wind data structure.

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

Cell quality flag: Every WVC contains a flag for its quality. Therefore the *CellType* contains an instance of the *CellQualFlagType*. Table 4.18 gives an overview of its attributes. The function *getCellQualFlag()* interprets an integer flag (BUFR input) to an instance of *CellQualFlagType*. The function *setCellQualFlag()* transforms an instance of *CellQualFlagType* to an integer flag.

Note that the MLE and AR flags, bits 10 and 9, have a different definition than the original NOAA product. The MLE flag has been modified following the procedure described in section 4.3.3. The AR flag indicates the quality of the solution found by KNMI's 2D variational Ambiguity Removal procedure (chapter 6).

Attribute	Bit	2^{Bit}	Description
Missing			Flag not set (all bits on)
QualSigma0	15	32768	Inferior quality of σ_0 *data*
Azimuth	14	16384	Invalid azimuth angle
Reserved3	13	8192	
MonFlag	12	4096	Monitoring flag not calculated
MonValue	11	2048	Monitor flag
MLE	10	1024	KNMI + JPL Quality Control flag
AR	9	512	KNMI VarQC flag
Land	8	256	Land flag
Ice	7	128	Ice flag
Retrieval	6	64	No retrieval
Large	5	32	σ_0 too large
Small	4	16	σ_0 too small
RainFall	3	8	Rain flag not calculated
RainDetect	2	4	Rain detected
FourBeam	1	2	*Sigma0_in_Cell* does not equal 4

Table 4.18 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 *TestCell*.

Table 4.19 lists the attributes of the *CellProcessFlagType*. The function *getCellProcessFlag()* interprets an integer flag (BUFR input) to an instance of *CellProcessFlagType*. The function *setCellProcessFlag()* transforms an instance of *CellProcessFlagType* to an integer flag. The routines *PrnCellProcessFlag()* and *PrnCellQualityFlag()* may be used to print the bit values of the flags.

Attribute	Bit	2^{Bit}	Description
Missing		2147483647	Flag not set (all bits on)
RevNr	30	1073741824	Invalid revolution number
RowNr	29	536870912	Invalid row number
NodeNr	28	268435456	Invalid node number
Lat	27	134217728	Invalid latitude
Lon	26	67108864	Invalid longitude
MLEQC	25	33554432	MLE quality control set
Along_Track_Res	24	16777216	Invalid along track resolution
Across_Track_Res	23	8388608	Invalid across track resolution
ModelWind	22	4194304	Invalid background wind
Time2Edge	21	2097152	Invalid time to edge
Year	20	1048576	Invalid year specification
Month	19	524288	Invalid moth specification
Day	18	262144	Invalid day specification
Hour	17	131072	Invalid hour specification
Minute	16	65536	Invalid minute specification
Second	15	32768	Invalid second specification
Beam(4)	14	16384	Invalid data of outer aft beam
Beam(3)	13	8192	Invalid data of inner aft beam
Beam(2)	12	4096	Invalid data of outer fore beam
Beam(1)	11	2048	Invalid data of inner fore beam
Sigma0_In_Cell	10	1024	Invalid number of cells
	9	512	
Ambiguity	8	256	Invalid ambiguities
Selection	7	128	Invalid selection
Rain	6	64	Invalid rain data
Tb	5	32	Invalid brightness temperature

Table 4.19 Cell process flag bits (Fortran).

Flags for σ_0 data: Every beam contains an instance of *Sigma0Type*. This instance has three attributes to flag the information on σ_0. These attributes are of type *Sigma0QualFlagType*, *Sigma0ModeFlagType*, and *Sigma0SurfFlagType*. Table 4.20 gives an overview of the (bit) attributes of these flags.

The functions *getSigma0QualFlag()*, *setSigma0QualFlag()*, *getSigma0ModeFlag()*, *setSigma0ModeFlag()*, *getSigma0SurfFlag()*, and *setSigma0SurfFlag()* are introduced for the (backward) conversions to the corresponding integer flags. The routines *PrnSigma0QualFlag()*, *PrnSigma0ModeFlag()*, and *PrnSigma0SurfFlag()* may be used to print the bit values of the flags.

Quality flag			
Attribute	**Bit**	**2^{Bit}**	**Description**
Missing		2147483647	Flag not set (all bits on)
Useability	15	32768	σ_0 value
NoiseRatio	14	16384	Azimuth diversity
Negative	13	8192	Negativeσ_0 value
Range	12	4096	2DVar
Pulse	11	2048	Pulse
Convergence	10	1024	Convergence
FreqShift	9	512	Frequency shift
Temperature	8	256	Temperature
Attitude	7	128	Attitude
Ephemeresis	6	64	Ephemereris
Mode flag			
Attribute	**Bit**	**2^{Bit}**	**Description**
Horizontal	16	65536	
Vertical	15	32768	
Right	14	16384	
Left	13	8192	
HoriVert	12	4096	
RightLeft	11	2048	
Surface flag			
Attribute	**Bit**	**2^{Bit}**	**Description**
Land	15	32768	
Ice	14	16384	
IceMap	5	32	
AttenuationMap	4	16	

Table 4.20 σ_0 flag bits for quality, mode, and surface (Fortran).

4.3.2 Module *SwsBufr*

The module *SwsBufr* maps the SeaWinds data structure on BUFR messages and vice versa. A list of the BUFR data descriptors can be found in appendix A. Satellite and GMF identifiers are listed in tables 4.21 and 4.22. The module uses the genscat module *BufrMod*, see subsection 4.2.3, for the interface with the BUFR routine library. The SeaWinds data structure is defined in module *SwsData,* see subsection 4.3.1.

Satellite	Parameter	Value
ADEOS-1	*Adeos1Id*	280
QuikSCAT	*QscatId*	281
ADEOS-2	*Adeos2Id*	282

Table 4.21 BUFR SeaWinds satellite identifiers.

Instrument	Parameter	Value
Reserved	*GmfReserved*	0
SASS	*GmfSass*	1
SASS2	*GmfSass2*	2
NSCAT0	*GmfNscat0*	3
NSCAT1	*GmfNscat1*	4
NSCAT2	*GmfNscat2*	5
NSCAT2P	*GmfNscat2P*	6
QSCAT1	*GmfQscat1*	7

Table 4.22 BUFR GMF identifiers.

Table 4.23 provides an overview of the different routines and their calls in this module. Ex general, the SDP module *SdpIO* uses the *SwsBufr* module to set up its BUFR interface. The genscat support routines *GetCurrentDate()* and *GetCurrentTime()* are used to tag the BUFR messages with the date and time of creation.

Routine	Call	Description
SwsBufrInit		Initialize module *SwsBufr*
OpenSwsBufrInit		Open BUFR file
CloseSwsBufrInit		Close BUFR file
ReadSwsBufrInit		BUFR message to *SwsRowType*
WriteSwsBufrInit		*SwsRowType* to BUFR message
Values2CellNOAA	*WriteSwsBufrData*	BUFR values array to *CellType* in NOAA format
Values2CellGen	*WriteSwsBufrData*	BUFR values array to *CellType* in generic format
Cell2ValuesNOAA	*ReadSwsBufrData*	*CellType* to BUFR values array in NOAA format
Cell2ValuesGen	*ReadSwsBufrData*	*CellType* to BUFR values array in generic format

Table 4.23 Routines in module *SwsBufr*

Note that the routines *Values2Cell* and *Cell2Values*, which convert between BUFR and SDP internal representation, have two variants: one for the official NOAA BUFR format that supports up to four wind solutions, and one for the experimental generic format that supports up to 144 wind solutions. The latter format has not yet been approved by the WMO.

Remarks:

- BUFR message subset indices are fixed for Sws_Met BUFR. Therefore they are set once during the initialization of *SwsBufr* (for example in ERS processing). These indices must be computed from the BUFR data descriptors.

4.3.3 Module *SwsSupport*

- Module *SwsSupport* is the interface between the SWS layer and the general purpose routines in `genscat/support`. This module contains no routines or declarations, but only some use-statements referring to genscat routines.

4.4 Module Design for process layer

The process (SDP) layer consists of the modules *SdpAmbrem*, *SdpInversion*, *SdpIO*, *SdpPrePost*, and *SdpSupport*. Module *SdpSupport* contains only declarations and initializations, no subroutines. Table 4.24 lists the routines in the other modules. Routines indicated by a star are not called in the SDP processing chain.

SdpAmbRem	***SdpInversion***	***SdpIO***	***SdpPrePost***	
RemoveAmbiguity	*InitInversion*	*ReadBufrInput*	*ProcessInit*	*Monitoring*
GetBatch	*InitMeanMle*	*WriteBufrOutput*	*Preprocess*	*MonitoringCalculateData*
SelectWind	*CalcSortProb*	*ProcessSwsFileName*	*CopyInputOutput*	*MonitoringWriteStats*
InitProbGross	*InversionInCell*	*GetNwpFileNames*	*PrepareInput*	*MonitoringSetMonitorBits*
DummyAmbRem (*)	*InvertWVCs*	*GetOutputFileNames*	*SetInputMleQC*	*OutputConversion*
	DummyInversion (*)	*ProcessArguments*	*PrepareOutput*	*DummyPreProcess (*)*
		usage	*PostProcess*	

Table 4.24 Routines in the process layer modules.

4.4.1 Module *SdpSupport*

Module *SdpSupport* contains many support routines for the processing steps of the SDP program. The module inherits a lot of functionality (data structures and routines) from the *Sws*-modules, see section 4.3. In addition, the module contains the global definitions of the SDP program. Table 4.25 provides an overview.

Name	Type	Description	Remark
AlongRes	Real	Output along track resolution	
AcrossRes	Real	Output across track resolution	
RowStride	Integer	Along track Stride	
NodeStride	Integer	Across track stride	
NrInputRows	Integer	Actual number of input rows	*Nrows* in QDP
NrOutputRows	Integer	Actual number of output rows	*Nrows2* in QDP
NrInputNodes	Integer	Actual number of input WVC's	
NrOutputNodes	Integer	Actual number of output WVC's	
VerbosityLevel	Integer	Verbosity level	Default 0
ResolutionIndex	Integer	Index of resolution (0 – 15)	Default 0
Lnwp	Logical	Switch NWP	Default .false.
Lqdp	Logical	Switch QDP mode	Default .false.
Lmss	Logical	Switch MSS	Default .false.
Lqc	Logical	Switch quality control	Default .true.
Linvert	Logical	Switch inversion	Default .true.
Lambrem	Logical	Switch ambiguity removal	Default .true.
Lmonitor	Logical	Switch monitoring	Default .false.
InpRow()	*SwsRowType*	Input orbit rows	*Obs* in QDP
Outrow()	*SwsRowType*	Output orbit rows	*Obs2* in QDP

Table 4.25 Globals for the processing steps in the SDP program defined in module *SdpSupport.*

4.4.2 Module *SdpIO*

Module *SdpIO* has two tasks. The first task is to process the command line options and parameters; the second task is to read (write) BUFR messages from (to) the input (output) BUFR files. The data have to be converted from the BUFR data structures to the SeaWinds data structures and vice versa. Table 4.26 provides an overview of the different routines and their calls in this module.

Routine	Call	Description
ReadBufrInput	SDP	Read BUFR message from input file
WriteBufrOutput	SDP	Write BUFR message on output file
ProcessSwsFileName	*ProcessArguments*	
GetNwpFileNames	*ProcessArguments*	
GetOutputFileNames	SDP	
ProcessArguments	SDP	Process SDP command line options
usage	*ProcessArguments*	Report on the use of SDP

Table 4.26 Routines of module *SdpIO*.

4.4.3 Module *SdpPrePost*

Module *SdpPrePost* contains the routines to do all the pre- and postprocessing. Preprocessing consists of the procedures between the reading of the BUFR input and the wind retrieval for the output product. This includes assessments of the quality of the input data, rain flagging, land and ice flagging, and interpolation to the specified resolution.

Routine	Call	Description
ProcessInit	SDP	Initialization of the processing
PreProcess	SDP	Main routine of the preprocessing
PrepareInput	*PreProcess*	Preparation of the input cells for averaging
SetInputMleQc	*PreProcess*	Set normalized MLE quality control tag to input cells
PrepareOutput	*PreProcess*	Preparation of output cells (supercells) by averaging
CopyInputOutput	*PreProcess*	Copy σ_0 and K_p data from input to output
PostProcess	SDP	Main routine of the postprocessing
OutputConversion	*PostProcess*	Convert output from internal data types to BUFR format
Monitoring	*PostProcess*	Monitoring
MonitoringCalculateData	*Monitoring*	
MonitoringWriteStats	*Monitoring*	
MonitoringSetMonitorBits	*Monitoring*	
ProcessCleanUp	SDP	Memory management

Table 4.27 Routines of module *SdpPreprocess*.

Table 4.27 lists the tasks of the individual routines. SDP first calls routine *ProcessInit()* to be sure that essential dependencies are set and/or initialized. Next *PrepareInput()* is called to sort the row with respect to the revolution, row and node numbers. It also checks on the appearance of double

rows, that is, rows with the same revolution and row number. If *PrepareInput()* finds a double row it merges it into one row. In that case the number of input rows will be reduced. Once the input rows are initialized, *SetInputMleQC()* will set the quality flag using the normalized MLE, R_n, defined as [*Portabella,* 2000; *Portabella and Stoffelen,* 2001]

$$R_n = \frac{MLE}{\langle MLE \rangle} \quad , \tag{4.1}$$

with MLE the maximum likelihood estimator and $\langle MLE \rangle$ its average value, obtained from actual measurements. The MLE is defined as

$$MLE = \frac{1}{N} \sum_{i=1}^{N} \frac{\left(\sigma_0^{\text{meas},i} - \sigma_0^{\text{simul},i}\right)^2}{K_p(\sigma_0^{\text{simul},i})} \quad . \tag{4.2}$$

In (4.2), $\sigma_0^{\text{meas},i}$ stands for the measured value of the radar cross section in a WVC, and $\sigma_0^{\text{simul},i}$ for the simulated value which depends on wind speed and direction. The denominator $K_p(\sigma_0^{\text{simul},i})$ quantifies the noise in the simulation, i.e., the estimated uncertainty in the GMF. The summation is over all beams of the scatterometer. For SeaWinds, N=4 in the sweet and central swath.

The MLE can be regarded upon as the distance between an actual scatterometer measurement and the GMF in N-dimensional measurement space. The MLE is related to the probability P that the GMF at a certain wind speed and direction represents the measurement by

$$P \propto e^{-MLE} \quad . \tag{4.3}$$

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, probably because the measurements are contaminated by ice, rain, and/or confused sea state, phenomena not included in he GMF. The ratio R_n further refines this notion by taking the uncertainty of the GMF into account. *Portabella* [2000] and *Portabella and Stoffelen* [2001] derive the following threshold values for R_n as a function of wind speed w

$$R_n^{\text{thres}} = \begin{cases} 4 - 0.05(w-5)^2 & , \quad w < 15 \text{ m/s} \\ 2 & , \quad w > 15 \text{ m/s} \end{cases} \quad . \tag{4.4}$$

When $R_n < R_n^{\text{thres}}$, the solution is considered close enough to the measurement and is accepted. If, on the other hand, $R_n > R_n^{\text{thres}}$, the solution lies too far away from the measurement, probably because the measurement is not well described by the GMF. The solution is therefore rejected.

In addition, for the nadir swath (that is for node number between 28 and 49) the rain flag of the NOAA product is adopted. That is, if NOAA sets the rain flag for this region also the MLE rain flag is set. The main task of the *PrepareOutput()* is to average the input data and to produce an output orbit with a lower resolution (less rows, less nodes). The wind vector cells of the output orbit are sometimes called supercells. The averaging concerns all input data needed to define the temporal and spatial location of the output cell and the beam data (σ_0, K_p) of the output cell. In addition, *PrepareOutput()* adjusts the quality flags of the output cells and the output σ_0 data.

Postprocessing consists of the procedure between the ambiguity removal step and the BUFR encoding of the output. Currently, postprocessing is confined to some simple conversions. It also includes the monitoring.

4.4.4 Module *SdpInversion*

Module *SdpInversion* 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 module *Inversion*, see subsection 4.2.1. Table 4.28 provides an overview of the different routines and their calls in this module.

Routine	Call	Description
InitInversion	*InvertWVCs*	Initialization
InitMeanMle	*InitInversion*	Set the mean MLE
CalcSortProb	*InversionInCell*	Calculate the probabilities and sort the ambiguities on probability
InversionInCell	*InvertWVCs*	Call to the genscat inversion module
InvertWVCs	SDP	Loop over all output cells

Table 4.28 Routines of module *SpdInversion*.

4.4.5 Module *SdpAmbrem*

Module *SdpAmbrem* controls the ambiguity removal step of the SDP program. The actual ambiguity removal schemes are implemented in the genscat module *ambrem*, see subsection 4.2.2. The default method is the KNMI 2DVar scheme. Table 4.29 lists the tasks of the individual routines.

Routine	Call	Description
RemoveAmbiguity	SDP	Main routine of ambiguity removal
GetBatch	*RemoveAmbiguity*	Obtain a batch of observations
SelectWind	*RemoveAmbiguity*	Final selection
InitProbGross	*RemoveAmbiguity*	Set the gross probabilities

Table 4.29 Routines of module *SpdAmbrem*.

The ambiguity removal scheme works on a so-called batch. The batch is defined in the *GetBatch()* routine. For the SDP program a batch is just a set of rows. The size of the batch is determined by the resolution of the structure functions and the number of FFT. The genscat routine *DoAmbrem()* performs the actual ambiguity removal scheme.

Finally *SelectWind* passes the selection to the output WVC's.

Chapter 5

Inversion module class

5.1 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 term of wind speed and wind direction) to a σ_0 value. The GMF further depends not only wind speed and wind direction, but also on the measurement geometry (relative azimuth and incidence angle), and beam parameters (frequency, polarisation). For SeaWinds, a maximum likelihood estimator (MLE) is used to select a set wind vector solutions that optimally match the observed σ_0's. The wind vector solutions correspond to local minima of the MLE function

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

With N the number of independent 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.

Details on the SeaWinds inversion problem can be found in [*Portabella,* 2002]. Details on the original JPL inversion approach can be found in [*Draper et al.,* 2002]. The SDP program includes the Multiple Solution Scheme (MSS), see [*Portabella and Stoffelen*, 2001].

5.2 Routines

The inversion module class contains only one module named *inversion*. It is located in subdirectory `genscat/inversion`. Table 5.1 lists all routines in this module. Appendix B.1 shows the calling tree for the inversion routines.

Routine	Call	Routine	Call
invert_one_wvc	SDP	*set_wind_speed_first_guess*	see B.1
fill_wind_quality_code	*invert_one_wvc*	*get_dynamic_range*	not used
remove_one_solution	*fill_wind_quality_code*	*get_GMF_version_used*	not used
save_inv_input	not used	*calc_sigma0*	not used
read_inv_input	not used	*INTERPOLATE*	generic
save_inv_output	not used	*interpolate1d*	*calc_sigma0*
do_parabolic_winddir_search	*invert_one_wvc*	*interpolated2d*	*calc_sigma0*
calc_normalisation	*invert_one_wvc*	*interpolate2dv*	*calc_sigma0*
calc_sign_MLE	*invert_one_wvc*	*interpolate3d*	*calc_sigma0*
print_message	see B.1	*read_LUT*	*calc_sigma0*
init_inv_input	SDP	*create_LUT_C_VV*	*calc_sigma0*
init_inv_output	*invert_one_wvc*	*test_for_identical_LUTs*	*calc_sigma0*
init_inv_settings_to_default	SDP	*my_mod360*	not used
write_inv_settings_to_file	not used	*my_mod*	not used
get_inv_settings	SDP	*my_min*	see B.1
set_inv_settings	SDP	*my_max*	see B.1
check_input_data	*invert_one_wvc*	*my_average*	see B.1
find_minimum_cone_dist	*invert_one_wvc*	*get_indices_lowest_local_minimum*	*invert_one_wvc*
get_parabolic_minimum	*do_parabolic_winddir_search*	*my_index_max*	see B.1
calc_cone_distance	*find_minimum_cone_dist*	*my_exit*	see B.1
calc_dist_to_cone_center	*fill_wind_quality_code*	*print_wind_quality_code*	see B.1
convert_sigma_to_zspace	*invert_one_wvc*	*print_input_data_of_inversion*	*check_input_data*
get_ers_node_formfactor	*calc_var_s0*	*print_output_data_of_inversion*	see B.1
calc_var_s0_ers	*invert_one_wvc*	*print_inout_data_of_inversion*	not used
get_wind_speed_first_guess	*find_minimum_cone_dist*	*calc_sigma0_cmod4*	*create_LUT_C_VV*
get_ers_noise_estimate	*calc_var_s0*	*f1*	*calc_sigma0_cmod4*
calc_var_s0	*calc_normalisation*	*Get_Br_from_Look_Up_Table*	*calc_sigma0_cmod4*
get_wind_speed_first_guess	*find_minimum_cone_dist*	*calc_sigma0_cmod5*	*create_LUT_C_VV*

Table 5.1 Routines in module *inversion*.

To establish the MLE function (1), the radar cross section according to the GMF, σ_o^{GMF}, must be calculated. This is done in routine *calc_sigma0*. The GMF at K_u band for HH and VV polarization needed for SeaWinds, is not known in analytical form. It is only available in the form of Look Up Tables (in directory `SDP/lut`). The value for σ_o^{GMF} is obtained from interpolation of these tables. The interpolation is done via symbolic routine *INTERPOLATE* which is set to *interpolate1d*, *interpolate2d*, *interpolate2dv*, or *interpolate3d*, depending on the type of interpolation needed.

For C-band at VV polarization the GMF is given in analytical form (routines *calc_sigma0_cmod4* and *calc_sigma0_cmod5*, respectively). In order to treat all scatterometer types in the same way, the radar cross section at C-band is also calculated from interpolation of Look Up Tables (LUTs). If a C-band LUT is not present it will be created by routine *create_LUT_C_VV*. This routine calls one of the routines *calc_sigma0_cmod4* or *calc_sigma0_cmod5* that contain the analytical expressions of the CMOD4 or CMOD5 algorithm. Routines *get_lun* and *free_lun* from module *LunManager* in subdirectory `genscat/support/file` are needed when reading and creating the LUTs.

5.3 Antenna direction

The output wind direction of inversion routines are generally given in the meteorological convention, see table 5.2. The inversion routine uses a wind direction that is relative to the antenna direction. The convention is that if the wind blows towards the antenna then this relative wind direction equals to 0. Therefore, it is important to be certain about the convention of your antenna (azimuth) angle.

For the SeaWinds Met product the radar look angle (antenna angle or simply azimuth) equals 0 if the antenna is orientated towards the north. The SeaWinds radar look angle increases clockwise. For ERS, however the antenna direction equals zero if the antenna directs towards the south. Therefore the final output wind direction needs a correction of 180 degrees.

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

Table 5.2 Meteorological conventions for the wind direction.

Chapter 6

Ambiguity Removal module class

6.1 Ambiguity Removal

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

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

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

meteorologically consistent.

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

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

6.2 Module *Ambrem*

Module *Ambrem* is the interface module between the various ambiguity removal methods and the different scatterometer data processors. Table 6.1 provides an overview of the different routines and their calls. A dummy method and the first rank selection method are implemented as part of *ambrem*. More elaborate Ambiguity Removal methods have an interface module, see table 6.2. Figure 6.1 shows schematically the interdependence of the various modules for Ambiguity Removal.

Routine	Call	Description
InitAmbremModule	SDP	Initialization of module *Ambrem*
InitAmbremMethod	SDP	Initialization of specified AR scheme
DoAmbrem	SDP	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	

Table 6.1 Routines of module *Ambrem*.

Routine	Description	Documentation
Ambrem2DVAR	Interface to KNMI 2DVar method	Section 6.4
AmbremBGClosest	Interface to Background Closest method	Section 6.1
AmbremPrescat	Interface to Prescat method	Section 6.5

Table 6.2 Interface modules for different Ambiguity Removal schemes.

6.3 Module *BatchMod*

After the wind retrieval step, the Ambiguity Removal step is performed on selections of the available data. In general, these selections are just a compact part of the swath or a compact part of the world ocean. The batch module *BatchMod* facilitates these selections of data. In fact, a batch data structure is introduced to create an interface between the swath related data and the data structures of the different AR methods. Consequently, the attributes of the batch data

structures are a mixture of swath items and AR scheme items. Figure 6.2 gives a schematic overview of the batch data structure. Descriptions of the attributes of the individual batch data components are given in table 6.3.

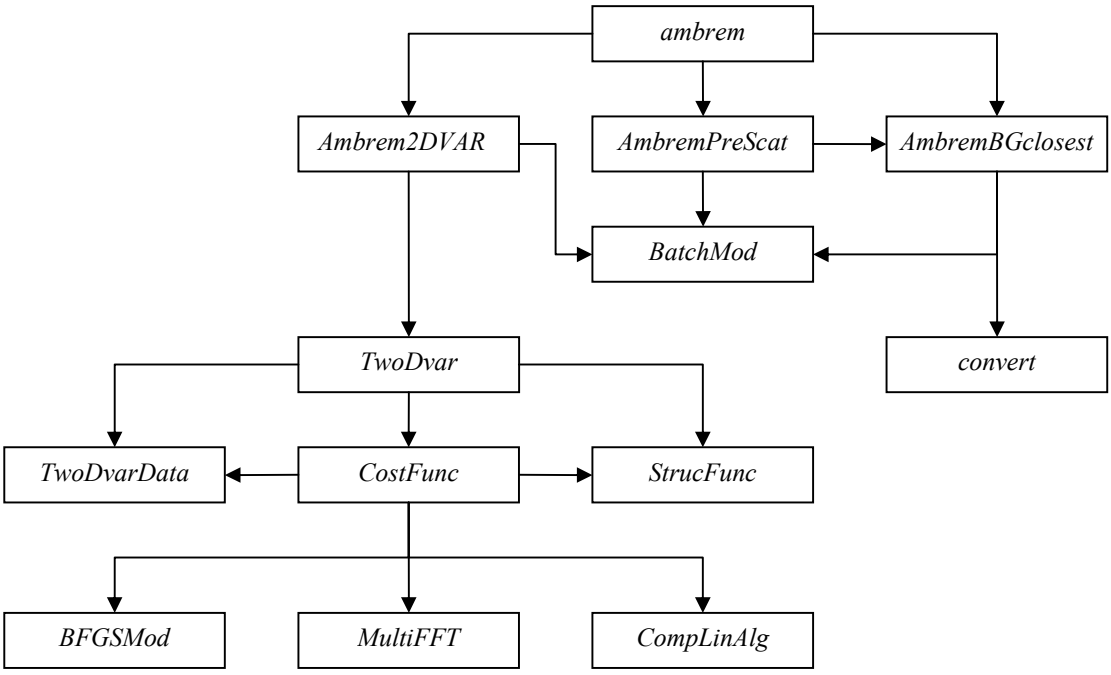


Figure 6.1 Interdependence of the modules for Ambiguity Removal. The connections from module *ambrem* to module *BatchMod* and from module *Ambrem2DVAR* to *convert* are not drawn.

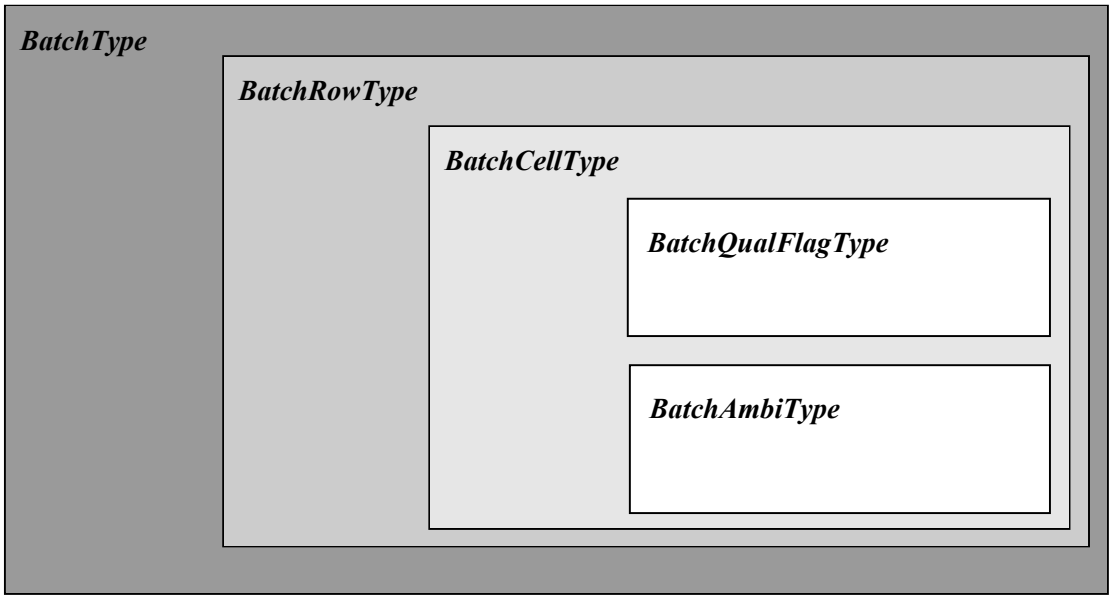


Figure 6.2 Schematic representation of the batch data structure.

BatchType

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

BatchRowType

Attribute	Type	Description
RowNr	Integer	Row number within orbit
NrCells	Integer	Number of cells in batch (max 76)
Cell	*BatchCellType*	Array of cells within row

BatchCellType

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

BatchAmbiType

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

Table 6.3 Batch data structures.

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

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

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

Table 6.4 Batch quality flag attributes.

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

Table 6.5 Routines of module *BatchMod*.

6.4 The KNMI 2DVar scheme

6.4.1 Introduction

The purpose of the KNMI 2DVar scheme is to make an optimal selection provided the (modeled) likelihood of the ambiguities and the (modeled) uncertainty of the background surface wind field. First, an optimal estimated surface wind vector field is determined based on variational principles

(2DVar). 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 minimizing the (RMS) distance to the analysis wind vector. For details of the KNMI 2DVar scheme formulation the reader is referred to [*Stoffelen et al.,* 2004; *de Vries et al.,* 2004; *de Vries and Stoffelen,* 2000]. These three documents may be downloaded from the EUMETSAT website, www.eumetsat.int.

The calculation of the cost function and its gradient is rather complex matter. The reader who is only interested in how the 2DVar scheme is assembled into the genscat module class *ambrem* is referred to subsection 6.4.2. Readers interested in the details of the cost function calculations and the minimization should also read the subsequent subsections. Subsection 6.4.3 forms an introduction to the cost function. It is recommended to first read this section, because it provides necessary background information to understand the code. Subsection 6.4.8 on the actual minimization and subsection 6.4.9 on Fast Fourier Transforms are in fact independent of the cost function itself. The reader might skip these subsections.

Remarks:

- The 2DVar scheme in SDP is in essence the same as that of QDP. However the implementation largely differs.

6.4.2 Data structure, interface and initialisation

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

Routine	Call	Description
Do2DVARonBatch	*DoAmbrem*	Apply 2DVar scheme on batch
BatchInput2DVAR	*Do2DVARonBatch*	Fills the 2DVar data structure with input
BatchOutput2DVAR	*Do2DVARonBatch*	Fills the batch data structure with output
SetAlpha	*BatchInput2DVAR*	Sets the observation orientation
GetBatchSize2DVAR		Determine maximum size of batch
latlon2xyz	*SetAlpha*	Coordinate transformation
rotuv	*BatchInput2DVAR*	Calculates the rotation of the (u,v) wind field

Table 6.6 Routines of module *Ambrem2DVAR*.

These routines are sufficient to couple the 2DVar scheme to the processor. The actual 2DVar processing is done by the routines of module *TwoDvar* itself. These routines are listed in table 6.7. Figures B2.1-B2.7 show the complete calling tree of the AR routines.

Routine	Call	Description
InitTwodvarModule		Initialization of module *TwoDvar*
Do2DVAR	*Do2DVARonBatch*	Cost function minimization
Init2DVARmeth	*Do2DVARonBatch*	Initialize the 2DVar scheme

Table 6.7 Routines of module *TwoDvar*.

The *Obs2dvarType* data type is the main data structure for the observed winds. Its attributes are listed in table 6.8. This data structure is implemented in module *TwoDvarData* and the routines working with the *Obs2dvarType* structure are listed in table 6.9. The quality status of an instance of *Obs2dvarType* is indicated by the attribute *QualFlag* which is an instance of *TwoDvarQualFlagType*. The attributes of this flag are listed in table 6.10.

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

Table 6.8 The 2DVAR data structure.

Routine	Call	Description
Init2dvarSettings	not used	Initialization of settings
InitObs2dvar	*BatchInput2DVAR, BatchOutput2DVAR*	Allocation of observations array
DeallocObs2dvar	*BatchOutput2DVAR*	Deallocation of observations array
InitOneObs2dvar	*InitObs2dvar*	Initialization of single observation
TestObs2dvar	*Do2DVAR*	Test single observation
PrintObs2dvar	*BatchInput2DVAR*	Print a single 2DVar observation
Prn2DVARQualFlag	*Do2DVAR*	Print observation quality flag
set2DVARQualFlag	*TestObs2DVAR*	Convert observation quality flag to integer
get2DVARQualFlag	not used	Convert integer to observation quality flag

Table 6.9 Routines in module *TwoDvarData*.

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

Table 6.10 Attributes of 2DVar observation quality flag.

6.4.3 Reformulation and transformation

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

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

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

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

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

Such a formulation does not closely match the code of the 2DVar scheme. In fact, for scientific and technical reasons several transformations are applied to reformulate the minimization problem. Description of these transformations is essential to understand the different procedures within the code. Details on these transformations can be found in the literature. Here follow brief descriptions:

Incremental approach: Rather than working with the surface wind field v itself, the 2DVar scheme works with the increment wind field given by

$$d(v) = v - v_{bg} \quad . \tag{6.3}$$

Structure function orientation: The background error covariance structure is specified in terms of structure functions. The technical definition of these functions requires that the corresponding increment wind vectors are specified as if it was a wind vector towards the pole. The rotation involved is calculated by the genscat routine *rotuv()*.

Stream function and velocity potential: The background error structures are better specified in terms of the stream function, ψ , and the velocity potential, χ (often written as ϕ), see e.g. {*Stoffelen,* 1998]. These two scalars are defined as

$$\begin{aligned} u &= -\frac{\partial \chi}{\partial x} = -\frac{\partial \psi}{\partial y} \\ v &= -\frac{\partial \chi}{\partial y} = \frac{\partial \psi}{\partial x} \end{aligned} \quad , \tag{6.4}$$

Therefore, the 2DVar scheme minimization is formulated in terms of ψ and χ rather than u (eastings) and v (northings).

Fourier transformation: The structure functions specify the background error covariance structure of the (ψ,χ) field (stream function and velocity potential) in terms of 2D Fourier modes rather than real values. Fourier transformations within the 2DVar scheme are done with a 2D FFT method.

Regular grid: To apply the 2D FFT transformation on the wind field data, the data have to be interpolated on a regular grid. Weight factors have to be set to control the mapping .

Preconditing: For larger dimensions of the wind vector field, the inversion of the matrix B is computationally prohibitive. To circumvent the calculation of B^{-1}, or rather its equivalent in terms of the 2D Fourier components of the (ψ,χ) field, the 2DVar minimization problem is preconditioned. That is, the control vector is transformed with the linear operator A, defined by the Cholesky decomposition

$$B = AA^T \quad . \tag{6.5}$$

For covariance matrices the Cholesky decomposition always exists.

Reorganization: Especially the FFT method requires a specific organization of its input data. Therefore, the data of the control vector are reorganized on several occasions.

To summarize, the actual control array of the minimization problem in the 2DVar ambiguity removal scheme contains the preconditioned 2D Fourier coefficients of the gridded (stream function, velocity potential) increment field.

6.4.4 Module *CostFunc*

Module *CostFunc* contains the main procedure for the calculation of the cost function and its gradient. It also contains the minimization procedure. Table 6.11 provides an overview of the routines. To keep track of the different transformations, some variable names are introduced for the different fields, see e.g. the transformation routine *z2y()*.

Routine	Call	Description
Jt	*FuncAndGrad*	Total cost function
Jb	*Jt*	Background term of cost function
Jo	*Jt*	Observational term of cost function
JoScat	*Jo*	Single observation contribution to the cost function
JoScatTest	*JoScat*	Test of contribution to cost function
JoScatTry	not used	Test routine
x2z	*Jb, Jo*	Unpack of control vector
z2x	*Jb, Jo*	Pack of control vector (and gradient)
z2y	*Jo*	Several transformations of control vector
z2y_Adj	*Jo*	Adjoint of *z2y*
z2y_Test	not used	Test of *z2y* and *z2y_Adj*
convolute	*z2y*	Convolution of control vector with structure functions
convolute_adj	*z2y_adj*	Adjoint of *convolute*
convolute_Test	not used	Test of *convolute* and *convolute_adj*
frq2grd	*z2y*	Backward FFT transformation
frq2grd_adj	*z2y_adj*	Adjoint of *frq2grd*
frq2grd_Test	not used	Test of *frq2grd* and *frq2grd_adj*
pc2uv	*z2y*	Conversion from (ψ,χ) to (u,v)
pc2uv_Adj	*z2y_adj*	Adjoint of *pc2uv*
pc2uv_Test	not used	Test of *pc2uv_adj*
SetMuNu	*pc2uv*	Support routine for *pc2uv*
minimise	*Do2DVAR* (*TwoDvar*)	Minimization
FuncAndGrad	*minimise*	Cost function and gradient calculation

Table 6.11 Routines of module *CostFunc*.

x*,*g: *x* is the packed control vector. The packing procedure consists of (1) joining the real parts of

the 2D Fourier coefficients of the stream function, ψ, and the velocity potential, χ, in one vector and (2) omitting the corresponding imaginary parts, because they are equal to zero by definition. The packed control vector is passed to the minimization routine. Therefore, the final gradient *g* is also specified in terms of the format of *x*.

z: *z* is the unpacked control vector. It is an instance of *PCgridType* and it contains the Fourier coefficients of the stream function - velocity potential (ψ,χ) representation. The imaginary parts equal zero. In fact, the imaginary parts are not even stored in the control vector *x*.

zc: *zc* is the result of the convolution of *z* with the structure functions. The error covariance information is set by the structure functions and stored in the global variable *cov*. Note that the ψ and χ covariance are independent. *zc* is again an instance of the *PCgridType*.

yc: *yc* is the result of the conversion of *zc* to (*u*,*v*) frequency space.

y : Backward FFT of *yc* results in the specification of the control vector in the real (*u*,*v*) space. These are still gridded data. The weights within the observation point data structure are used to obtain values at the observation location.

Due to preconditioning the calculation of the background term of the cost function and its gradients has become relatively straightforward. First, the control vector must be unpacked by routine *x2z()*. Next, the norm of the control vector is calculate using the complex linear algebra routine *Cnorm2D()*. The corresponding adjoint routine *Cnorm2D_adj()* is used to calculate the gradient of this term. This gradient of course must be packed with a call to routine *z2x()*.

The calculation of the observational term is relatively complex. Again, first the control vector must be unpacked using *x2z()*. Next, a transformation is applied using *x2y()* to get real (*u*,*v*) increments. The weights specified for each observation point are used to calculate the contribution of the observations to the total gradient of the observational term. The routines *z2_adj()* and *z2x()* are respectively called to obtain a packed gradient for this term which can by added to the gradient of the background term to form the gradient of the total cost function.

6.4.5 Adjoint method

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

In brief, the reformulations and transformations of the cost function (see subsections 6.4.3 and 6.4.4) may be denoted as

$$J(y) = J[G(x)] \quad , \tag{6.6}$$

with

$$G = G_m \otimes G_{m-1} \otimes \cdots \otimes G_2 \otimes G_1 \quad , \tag{6.7}$$

Where the G_i's are assumed to be the formal differential representations of transformations involved. The chain rule for derivatives states is

$$\frac{\partial J}{\partial x_k} = \sum_{j=1}^{n} \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial x_k} \quad . \tag{6.8}$$

Since

$$y = G(x) \quad , \quad \delta y = G' \delta x \quad , \tag{6.9}$$

where δx and δy are infinitesimal perturbations in x and y, respectively, and G' is the Jacobian of G, we may write for the gradient of J with respect to x

$$\nabla_x J = G' \nabla_y J \quad , \tag{6.10}$$

where

$$G' = G'_m \otimes G'_{m-1} \otimes \cdots \otimes G'_2 \otimes G'_1 \quad . \tag{6.11}$$

In other words, we may calculate the final gradient by the subsequent application of the transposed Jacobians of the individual transformations. $\nabla_y J$ is supposed to be available as a simple expression. The actual coding of adjoint method must be developed directly from the actual code of J. There exist formal (even automatic) methods to do so, see e.g. [*Giering*, 1997]. It is good practice to test adjoint code by applying finite difference approximations. This can be done not only for G' but also for sequences of G'_k. Results of adjoint testing for the 2DVar scheme are given in [*Bonekamp et al.,* 2004].

6.4.6 Structure Functions

Module *StrucFunc* contains the routines to calculate the covariance matrices for the stream function, ψ, and the velocity potential, χ. Its routines are listed in table 6.12.

Routine	Call	Description
PrintStrcFuncPars	not used	
SetCovMat	*Do2DVAR*	Calculate the covariance matrices
InitStrucFunc	*SetCovMat*	Initialize the structure functions
StrucFuncPsi	*SetCovMat*	Calculate ψ
StrucFuncChi	*SetCovMat*	Calculate χ

Table 6.12 Routines of module *StrucFunc*.

6.4.7 Complex linear algebra

Module *CompLinAlg* contains some routines to perform some elementary linear calculations on a complex two-dimensional field. Instead of using the standard Fortran complex type, the data type *Ctype* is introduced with the real attributes *re* and *im* for respectively the real and imaginary parts. Table 6.13 lists the routines in module *CompLinAlg*.

Routine	Call	Description
Cnorm2D	*Jb*	Norm of 2D complex field
Cnorm2D_adj	*Jb*	Adjoint of *Cnorm2D*
Cnorm2D_Test	not used	Test of *Cnorm2D* and *Cnorm2D_adj*
Cdot2D	*Cnorm2D*	Dot product of two 2D complex fields
Cdot2D_adj	*Cnorm2D_adj*	Adjoint of *Cdot2D*
Cdot2D_Test	not used	Test of *Cdot2D* and *Cdot2D_adj*
Cprod2D	*Cdot2D*	Complex product of two 2D complex fields
Cprod2D_adj	*Cdot2D_adj*	Adjoint of *Cprod2D*
Cprod2D_Test	not used	Test of *Cprod2D* and *Cprod2D_adj*

Table 6.13 Routines in module *CompLinAlg*.

6.4.8 Minimisation

The minimization routine used is *LBFGS*. This is a quasi Newton method with a variable rank for the approximation of the Hessian written by J. Nocedal. A detailed description of this method is given by *Liu and Nocedal* [1989]. Routine LBFGS is freeware and can be obtained from web page www.netlib.org/opt/index.html, file `lbfgs_um.shar`. The original Fortran77 code has been adjusted to compile under Fortran90 compilers. Routine LBFGS and its dependencies are located in module `BFGSMod.F90` in directory `genscat/support/BFGS`. Table 6.14 provides an overview of the routines in this module.

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

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

Table 6.14 Routines in module *BFGSMod*.

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).

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 Fortran90. 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 Fortran90 intrinsic function *dot_product.*

6.4.9 MultiFFT

Module *MultFFT* in directory `genscat/support/multifft` contains the multi-variate complex Fourier routines needed in the 2DVar scheme. Actually there are two methods (implementations) available. These are the simple method, which is straightforward implementation of the 2D transform, and the fast method, which is an implementation of the Cooley-Tukey algorithm. The fast method is default in SDP, but the simple method is useful for testing purposes.

Routine	Call	Description
Fourier2DForward	*SetCovMat*	Forward 2D Fourier transform
Fourier2DForward_adj	not used	Adjoint of *Fourier2DForward*
Fourier2DBackward	*frq2grd*	Inverse 2D Fourier transform
Fourier2DBackward_adj	*frq2grd_adj*	Adjoint of *Fourier2DBackward*
Simple2DFT	*Fourier2DForward, Fourier2DBackward*	Basic 2D Fourier routine
Simple2DFT_adj	*Fourier2DForward_adj, Fourier2DBackward_adj*	Adjoint of *Simple2DFT*
Simple2DFT_test	not used	Test of *Simple2DFT*
Simple1DFT	not used	Basic 1D FT routine
Simple1DFT_adj	not used	Adjoint of *Simple1DFT*
Simple1DFT_test	not used	Test of *Simple1DFT*
Fast2DFT	*Fourier2DForward, Fourier2DBackward*	Fast 2D Fourier routine
Fast2DFT_adj	*Fourier2DForward_adj, Fourier2DBackward_adj*	Adjoint of *Fast2DFT*
Fast2DFT_test	not used	Test of *Fast2DFT*
FastFT	*Fast2DFT*	Basic 1D FFT routine
Fourier2DBackward_adj_test	not used	Adjoint test
Fourier2DForward_adj_test	not used	Adjoint test
PrintCompare2D	not used	Support routine for test
NormCompare2D	not used	Support routine for test

Table 6.15 Fourier transform routines.

Table 6.15 gives an overview of the available routines. Figure B.2.11 shows the calling tree of the FT routines.

6.5 The PreScat scheme

The PreScat ambiguity removal scheme is not used within SDP. More information on this scheme can be found in [*Stoffelen et al.,* 2004].

Chapter 7

Module *BufrMod*

Module *BufrMod* is part of the genscat support modules. The current version is a Fortran90 wrapper around the ECMWF BUFR library (see www.ecmwf.int). The goal of this support module is to provide a comprehensive interface to BUFR data for every Fortran90 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.

7.1 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 centers. 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 (www.wmo.ch) and of the European Center for Medium-range Weather Forecasts ECMWF (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 Fortran90 program. Section 7.2 describes the routines in module *BufrMod*. The public available data structures are described in section 7.3. *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 7.4.

7.2 Routines

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

Routine	Call	Description
InitAndSetNrOfSubsets	SDP	Initialization routine
set_BUFR_fileattributes	SDP	Initialization routine
open_BUFR_file	SDP	Opens a BUFR file
get_BUFR_nr_of_messages	SDP	Inquiry of BUFR file
get_BUFR_message	SDP	Reads instance of *BufrDataType* to file
get_expected_BUFR_msg_size	*get_BUFR_message*	
ExpandBufrMessage	*get_BUFR_message*	Convert from *BufrMessageType* to *BufrSectionsType*
PrintBufrErrorCode	*ExpandBufrMessage*	
CheckBufrTables	*ExpandBufrMessage*	Control
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	*get_BUFR_message*	Convert from *BufrSectionsType* to *BufrDataType*
close_BUFR_file	SDP	Closes a BUFR file
BufrReal2Int	SDP	Conversion
save_BUFR_message	not used	Saves instance of *BufrDataType* to file
EncodeBufrData	*save_BUFR_message*	Convert from *BufrSectionsType* to *BufrMessageType*
CheckBufrData	*EncodeBufrData*	Control
FillBufrData	*save_BUFR_message*	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	
BufrInt2Real	not used	
GetFreeUnit	not used	There is also a copy in module *SwsSupport*

Table 7.1 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*. This routine is not used in SDP.

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 both read (*writemode*=.false.) and 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 7.5. 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, with the BUFRIO library acting as an intermediate. Appendix B3 shows the calling trees of the routines in module *BufrMod* that are used in SDP.

7.3 Data structures

The data type closest to the actual BUFR messages in the BUFR files is the *BufrMessageType*, see table 7.2. 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 7.3, which contains the data and meta data in integer values subdivided in these sections.

Attribute	Type	Description
buff	Integer (max_bufr_mess_size)	BUFR message, all sections
size	Integer	Size in bytes of BUFR message
nr_of_words	Integer	Idem, now size in words

Table 7.2 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(64)	Integer	Expanded section 2 (optional)
ksec3(4)	Integer	Expanded section 3 (data description)
ksec4(2)	Integer	Expanded section 4 (data)

Table 7.3 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
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 7.4 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 7.4. 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 7.5..

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

7.4 Libraries

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

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

Appendix D provides an overview of the different routines of this library. From the calling trees in Appendix B3 it can be inferred that only a few routines of the BUFR software library are actually used.

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 7.6. The source file (`bufrio.c`) is located in subdirectory `genscat/support/bufr`.

Routine	Call	Description
bufr_open	*open_BUFR_file*	
bufr_split	*open_BUFR_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 7.6 Routines in library BUFRIO.

7.5 BUFR table routines

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

7.6 Center 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 Fortran90 code several center specific modules are created. These modules are listed in table 7.7. Note that these modules are just cosmetic and not essential for the encoding or decoding of the BUFR data.

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

Table 7.7 Fortran90 BUFR modules.

References

- Dragosavac, M., 1994,
  BUFR User Guide and Reference Manual. ECMWF. (Available via the ECMWF website www.ecmwf.int)
- Draper, D.W. and Long, D.G., 2002,
  An assessment of SeaWinds on QuikSCAT wind retrieval. *Journal of Geophysical Research*, **107**, C12, p. 3212 (2002JC001330).
- Freilich, M.H.,
  SeaWinds algorithm theoretical basis document. Report atbd-sws-01.
- Giering, R., 1997,
  Tangent linear and Adjoint Model Compiler, Users manual. Max-Planck- Institut fuer Meteorologie.
- Leidner, S.M., Hoffman, R.N., and Augenbaum, J., 2000,
  SeaWinds Scatterometer Real Time BUFR Geophysical Data Product User's guide. NOAA NESDIS. (Available from the KNMI site www.knmi.nl/scatterometer/seawinds).
- Liu, D.C., and Nocedal, J., 1989
  On the limited memory BFGS method for large scale optimization methods. *Mathematical Programming,* 45, pp. 503-528.
- Portabella, M., 2002,
  Wind field retrieval from satellite radar systems. PhD thesis, University of Barcelona. (Available via the KNMI site www.knmi.nl/scatterometer/seawinds).
- Portabella, M. and Stoffelen, A., 2001,
  Rain Detection and Quality Control of SeaWinds. *Journal of Atmospheric and Oceanic Technology*, **18** , pp. 1171-1183.
- Portabella, M. and Stoffelen, A., 2002,
  A Comparison of KNMI Quality Control and JPL Rain Flag for SeaWinds. *Canadian Journal of Remote Sensing (special issue on Remote Sensing of Marine Winds),* **28**, 3.

- Portabella, M. and Stoffelen, A., 2004,
  A probabilistic approach for SeaWinds Data Assimilation. *Quarterly Journal of the Royal Meteorological Society*, 130, pp. 1-26.
- SCAT group, 2005,
  SDP Test Report, version 1.2.
  KNMI, March 2006.
- Stoffelen, A., de Haan, S., Quilfen, Y., and Schyberg, H., 2004,
  ERS scatterometer ambiguity removal scheme comparison. KNMI/ EUMETSAT Ocean and Sea Ice report. (Available from the EUMETSAT website, www.eumetsat.int).
- Stoffelen, A., de Vries, J., and Voorrips, A., 2000,
  Towards the real-time use of QuikScat winds. Beleidscommissie Remote Sensing, report nr. USP-2/00-26.
- Stoffelen, A.C.M., 1998,
  Scatterometry. PhD thesis, University of Utrecht, ISBN 90-393-1708-9. (Available via the KNMI site www.knmi.nl/scatterometer/seawinds).
- 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.
- de Vries, J. and Stoffelen, A., 2000,
  2D Variational Ambiguity Removal. KNMI, Feb 2000. (Available from the EUMETSAT website, www.eumetsat.int).
- de Vries, J., Stoffelen, A., and Beysens, J., 2004,
  Ambiguity Removal and Product Monitoring for SeaWinds. KNMI. (Available from the EUMETSAT website, www.eumetsat.int).
- de Vries, J., et al., 2004,
  QuikScat Data Processor User Manual, KNMI. (Available from the KNMI website, www.knmi.nl/scatterometer/sw_proc).
- Wentz, and Smith, D.K., 1999,
  A model function for the ocean normalized cross section at 14 GHz derived from NSCAT observations. *Journal of Geophysical Research*, **104**, C5, pp. 11499-11507.

Appendix A

Calling tree for SDP

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

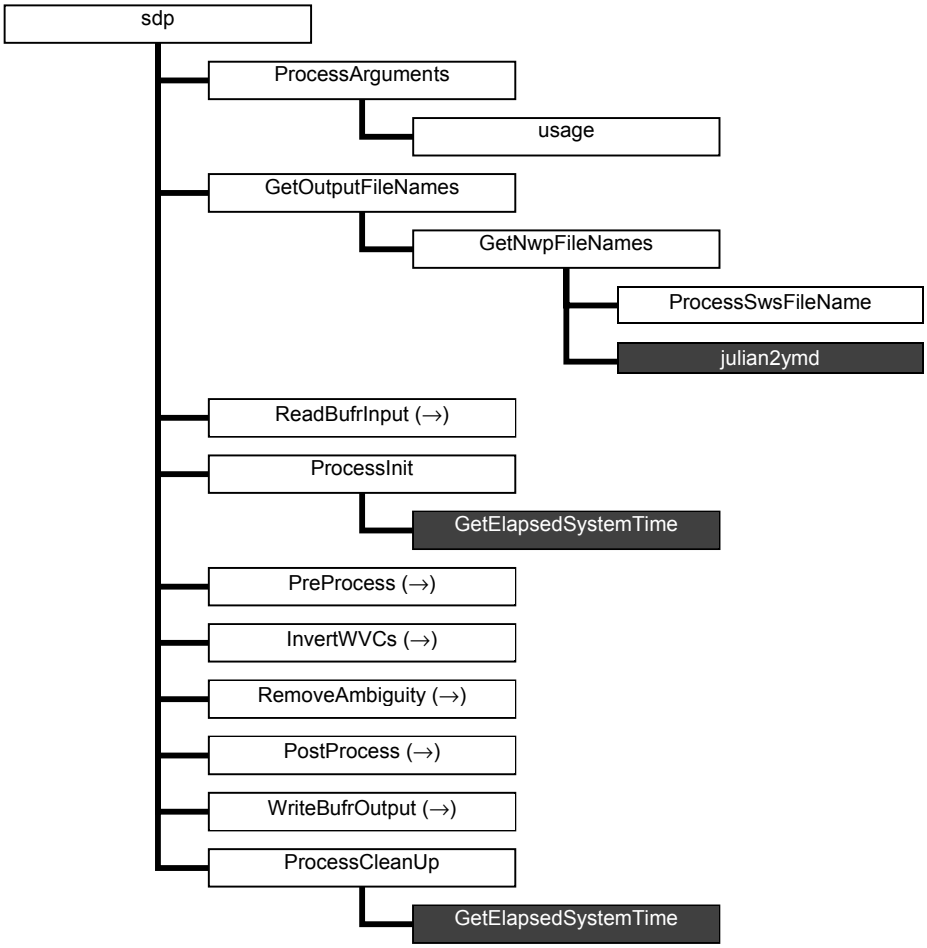


Figure A.1 Calling tree for program *sdp* (top level). Light grey boxes are cut here and will be continued in one of the first level or second level calling trees in the next figures. Black boxes with light text indicate genscat routines.

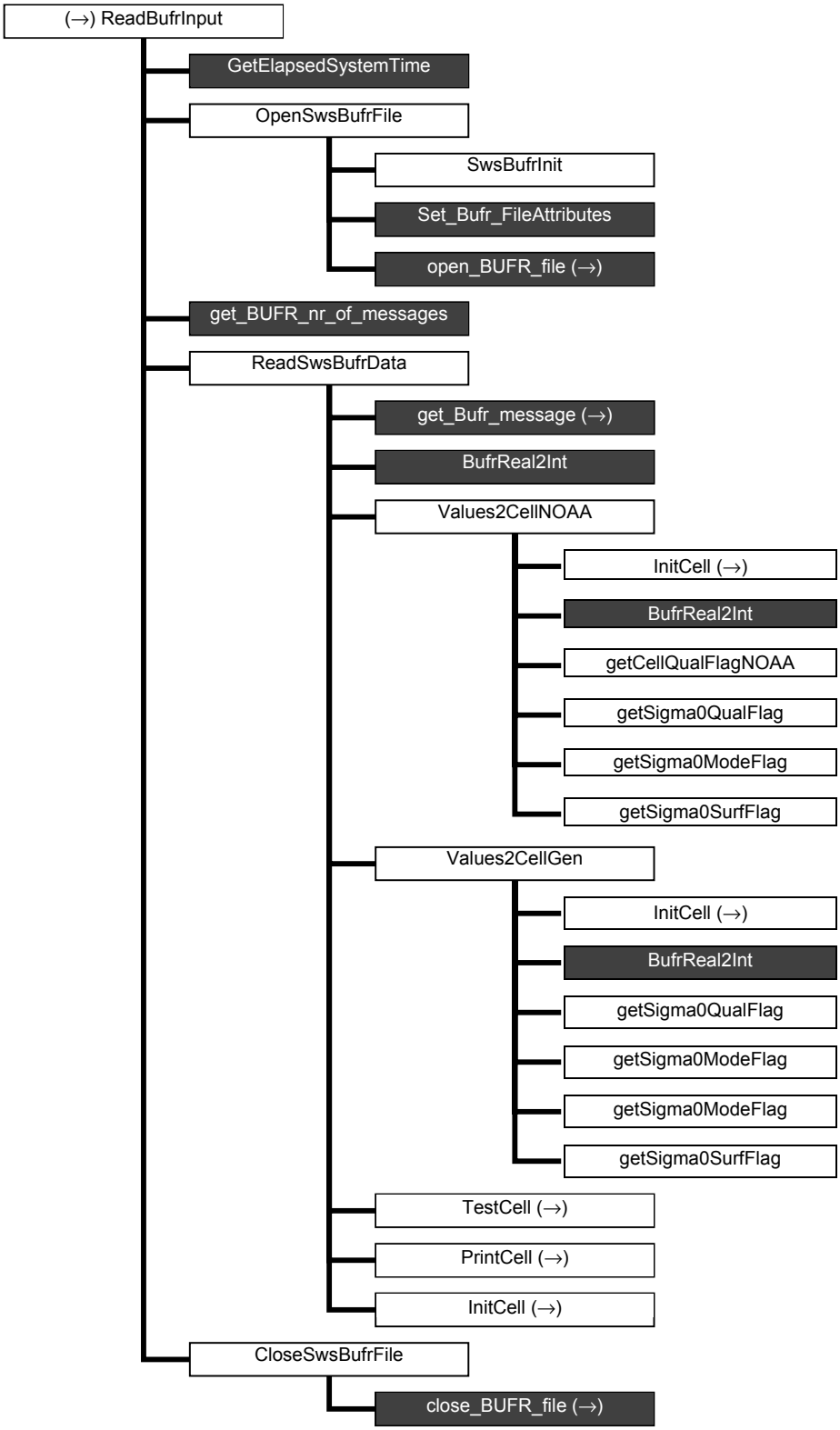


Figure A.2 Calling tree for routine *ReadBufrInput* (first level).

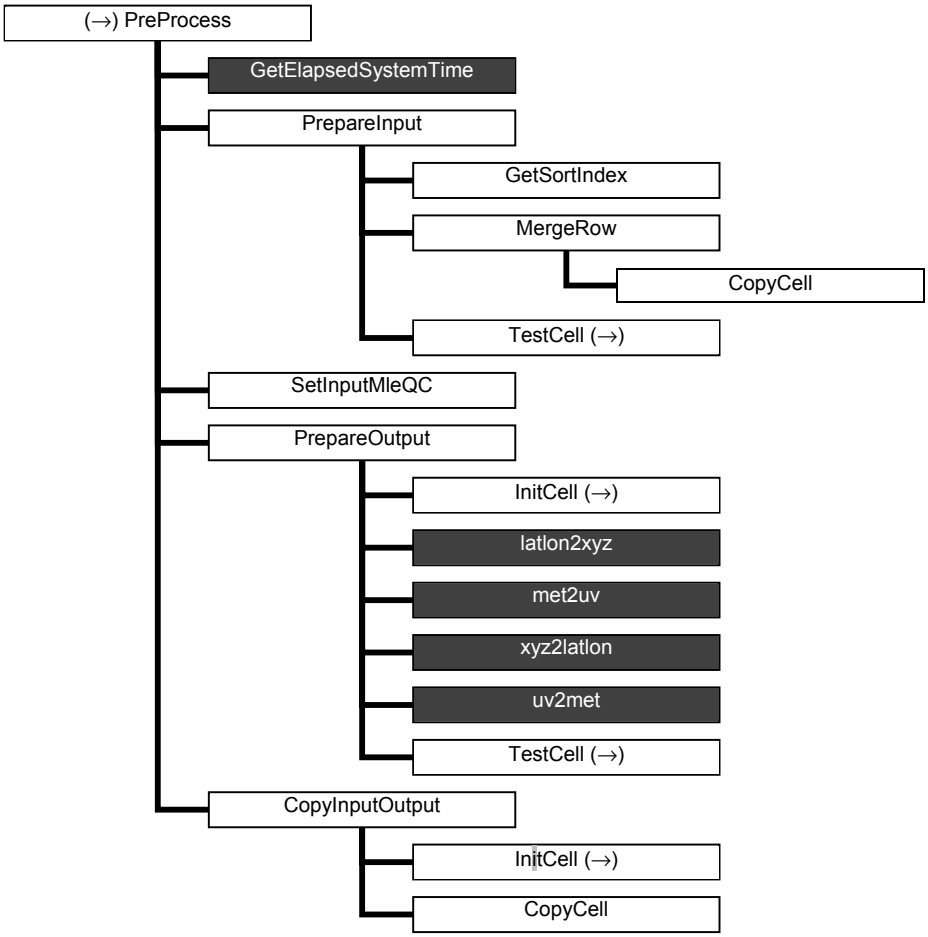


Figure A.3 Calling tree for routine *PreProcess* (first level).

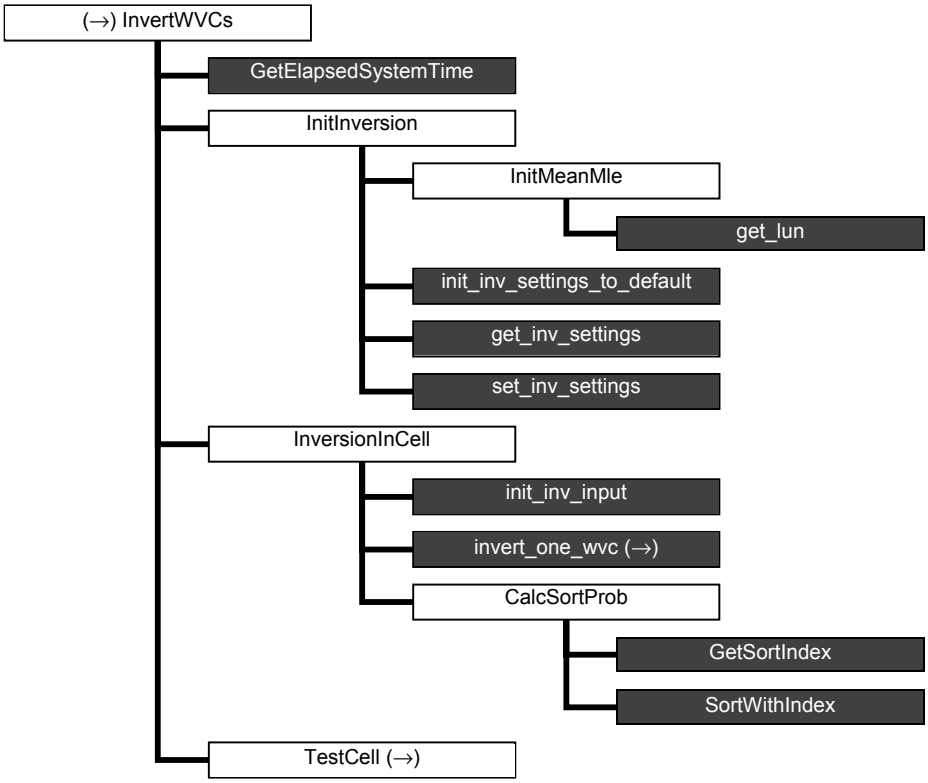
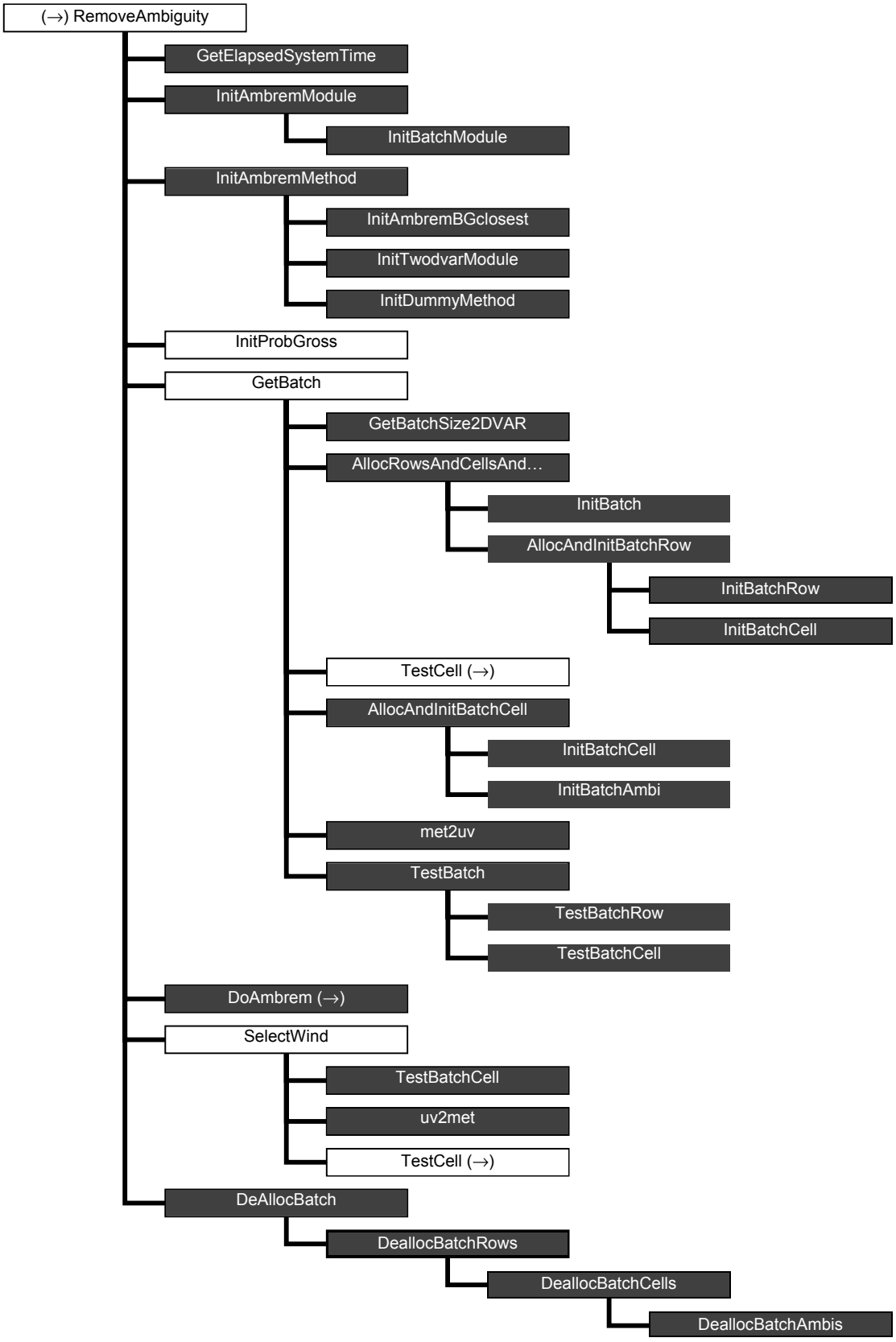


Figure A.4 Calling tree for routine *InvertWVCs* (first level).

Figure A.5 (next page) Calling tree for routine *RemoveAmbiguity* (first level). The full name of the 12th routine is *AllocRowsAndCellsAndInitBatch*.


(→) RemoveAmbiguity
GetElapsedSystemTime
InitAmbremModule
InitBatchModule
InitAmbremMethod
InitAmbremBGclosest
InitTwodvarModule
InitDummyMethod
InitProbGross
GetBatch
GetBatchSize2DVAR
AllocRowsAndCellsAnd…
InitBatch
AllocAndInitBatchRow
InitBatchRow
InitBatchCell
TestCell (→)
AllocAndInitBatchCell
InitBatchCell
InitBatchAmbi
met2uv
TestBatch
TestBatchRow
TestBatchCell
DoAmbrem (→)
SelectWind
TestBatchCell
uv2met
TestCell (→)
DeAllocBatch
DeallocBatchRows
DeallocBatchCells
DeallocBatchAmbis

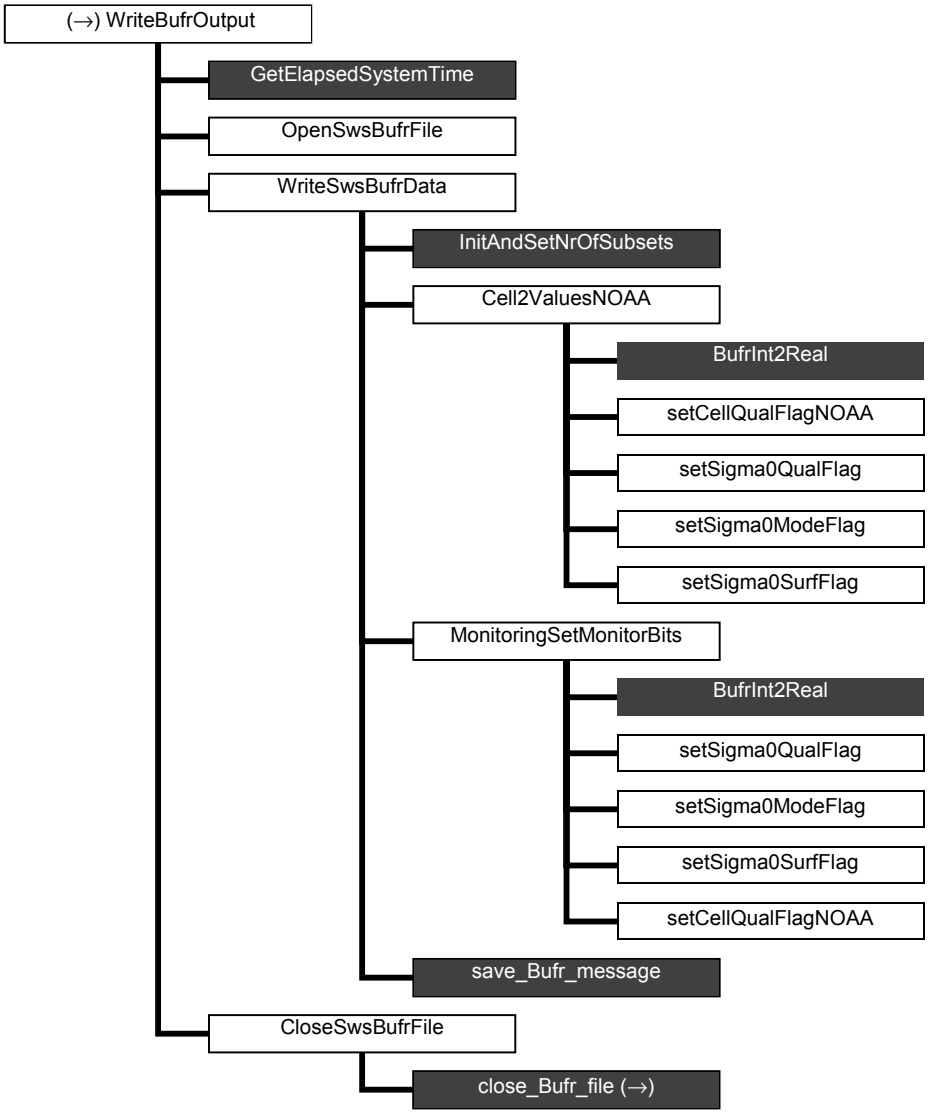


Figure A.6 Calling tree for routine *WriteBufrOutput* (first level).

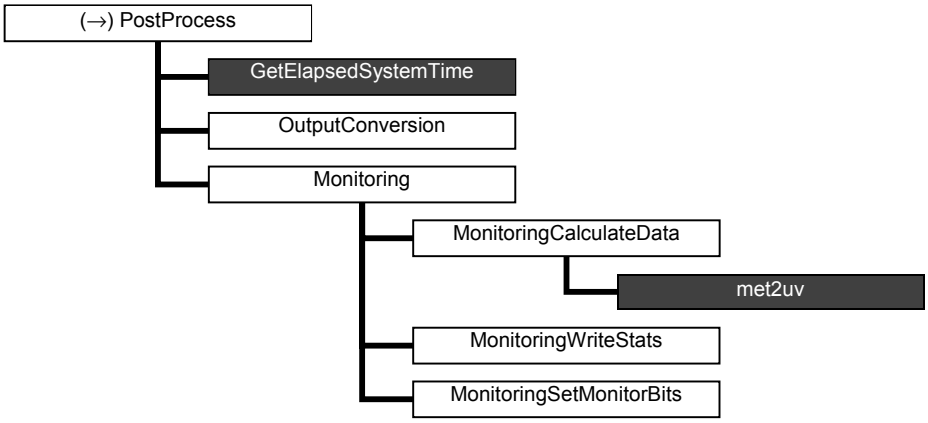


Figure A.7 Calling tree for routine *PostProcess* (first level).

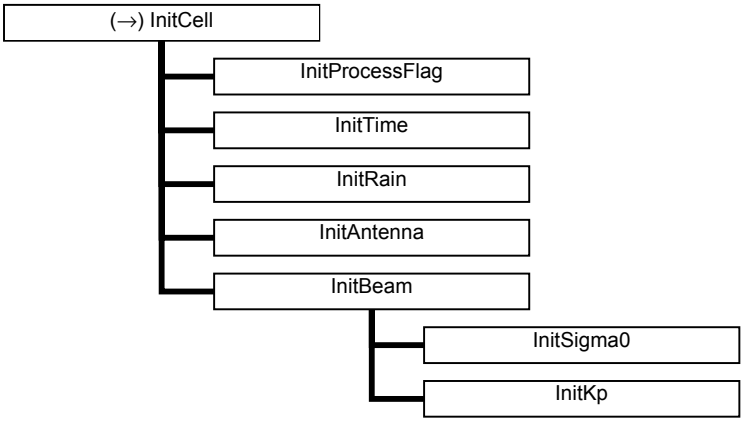


Figure A.8 Calling tree for routine *InitCell* (second level).

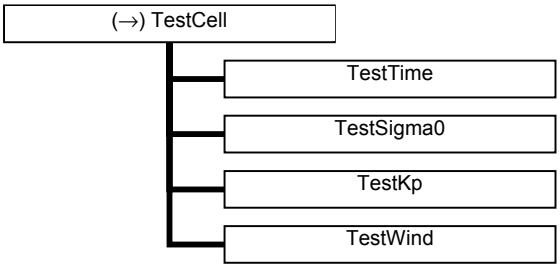


Figure A.9 Calling tree for routine *TestCell* (second level).

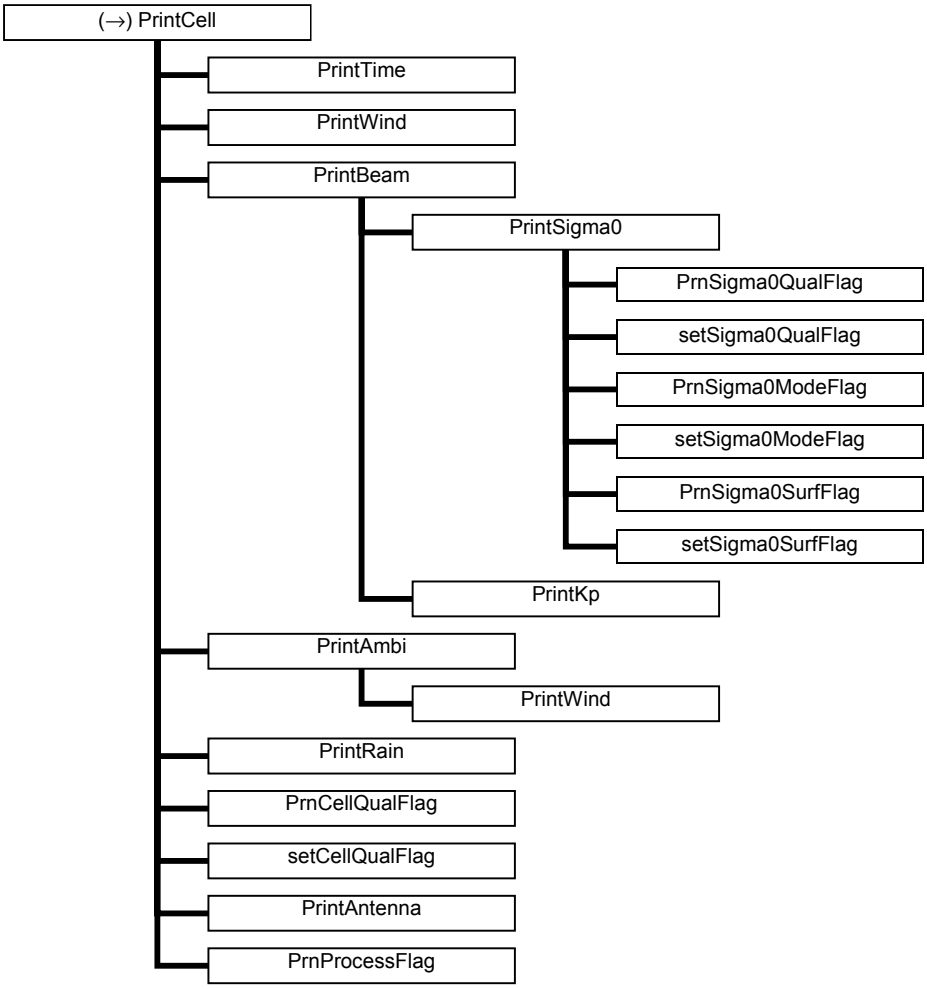


Figure A.10 Calling tree for routine *PrintCell* (second level).

Appendix B1

Calling tree for inversion routines

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

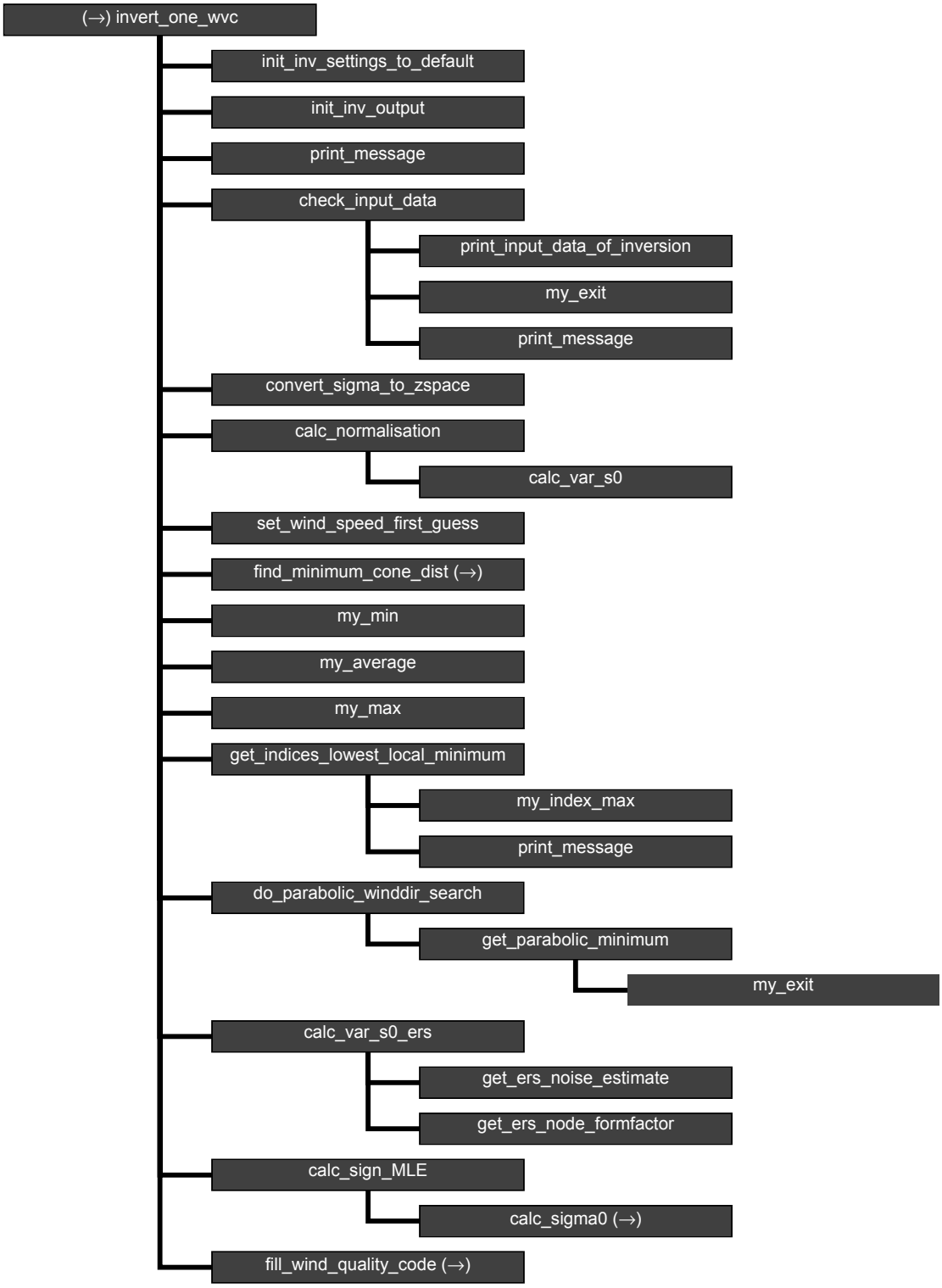


Figure B1.1 Calling tree for inversion routine *invert_one_wvc*.

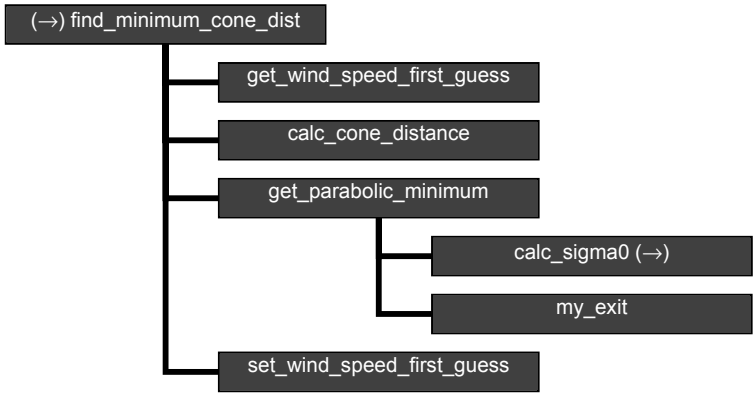


Figure B1.2 Calling tree for inversion routine *find_minimum_cone_dist*

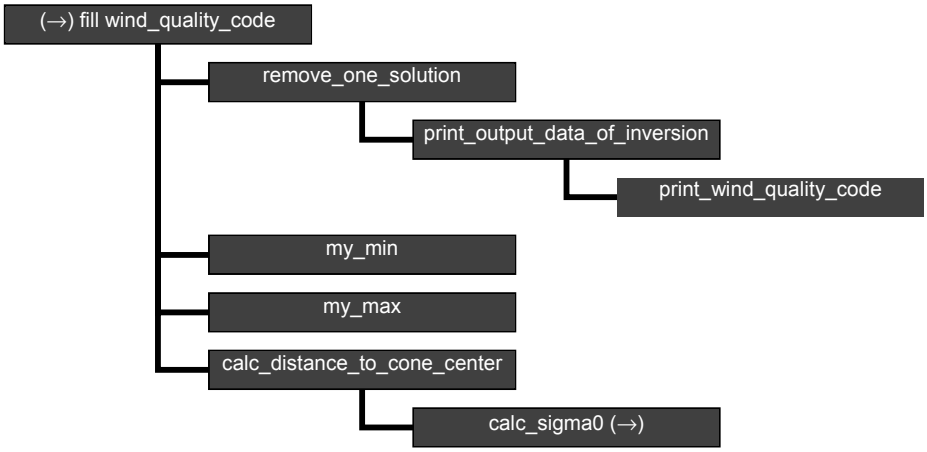


Figure B1.3 Calling tree for inversion routine *fill_wind_quality_code*.

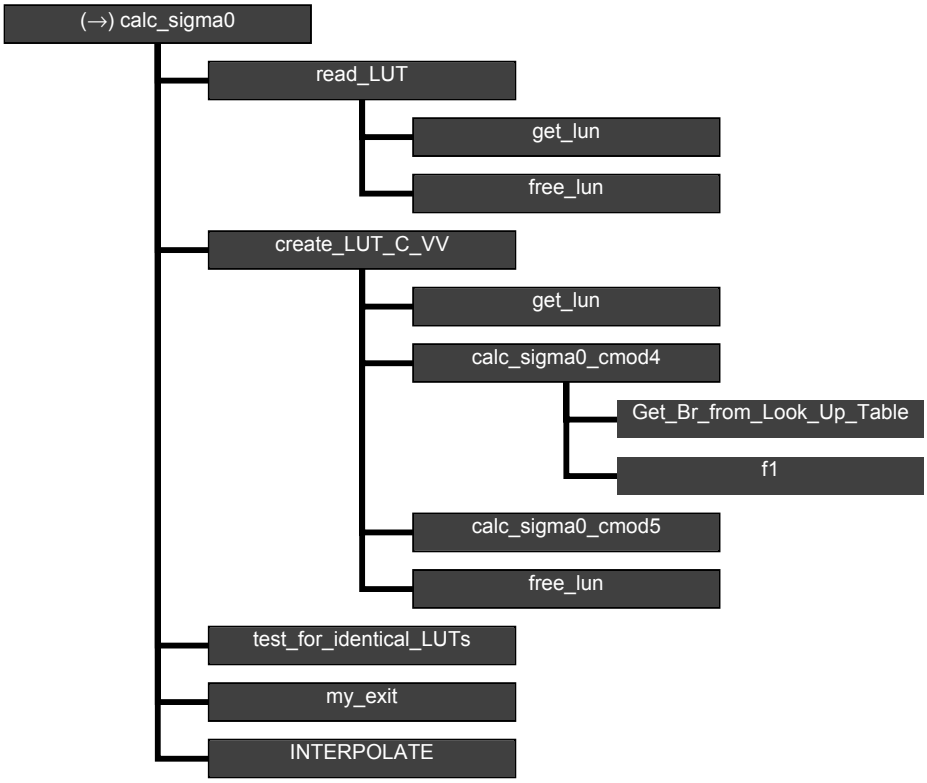


Figure B1.4 Calling tree for inversion routine calc_sigma0. Routine *INTERPOLATE* is an interface that can have the values *interpolate1d*, *interpolate2d*, *interpolate2dv* or *interpolate3d*.

Appendix B2

Calling tree for AR routines

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

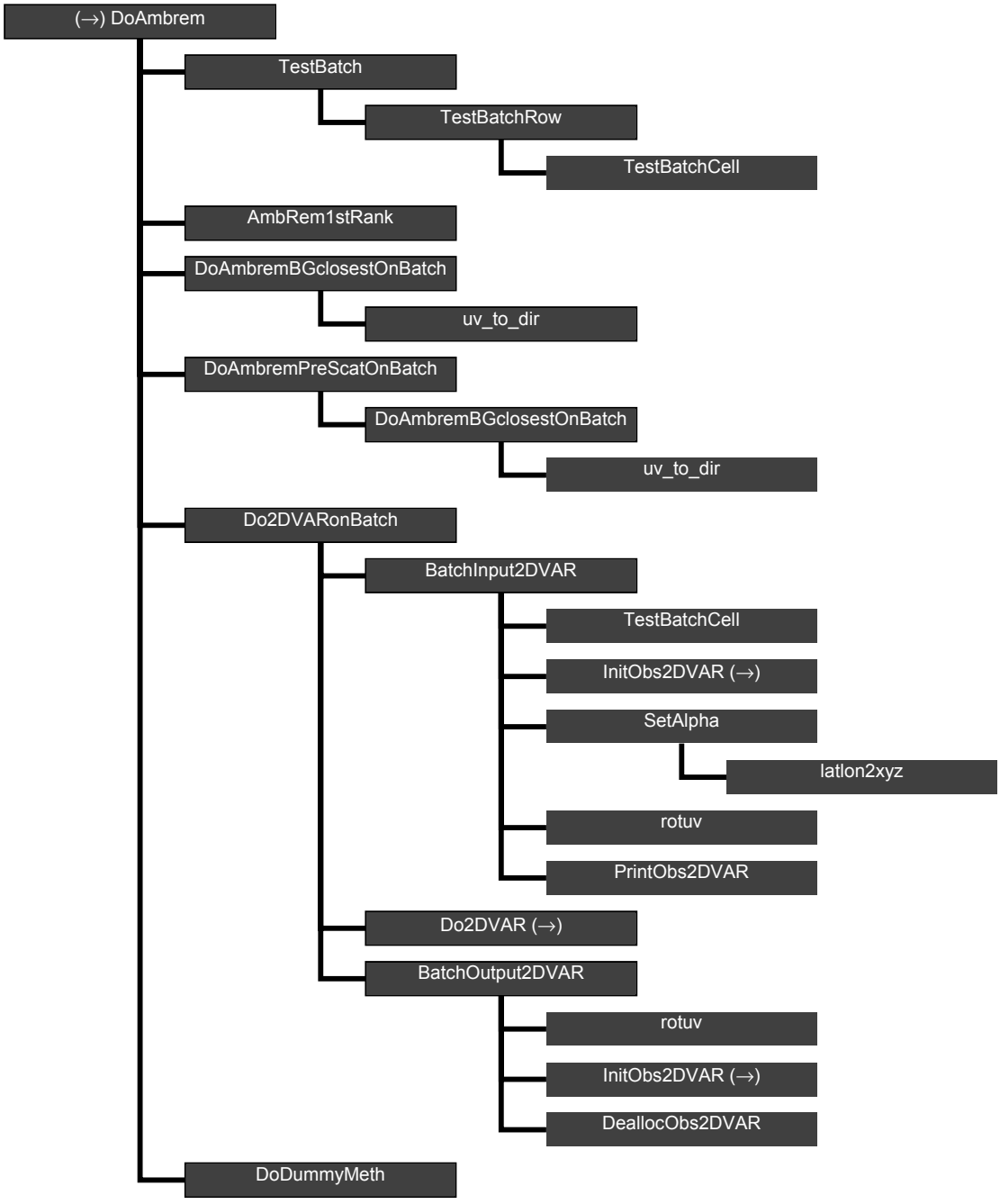


Figure B2.1 Calling tree for AR routine *DoAmbrem*.

Figure B2.2 Calling tree for AR routine *InitObs2DVAR*.

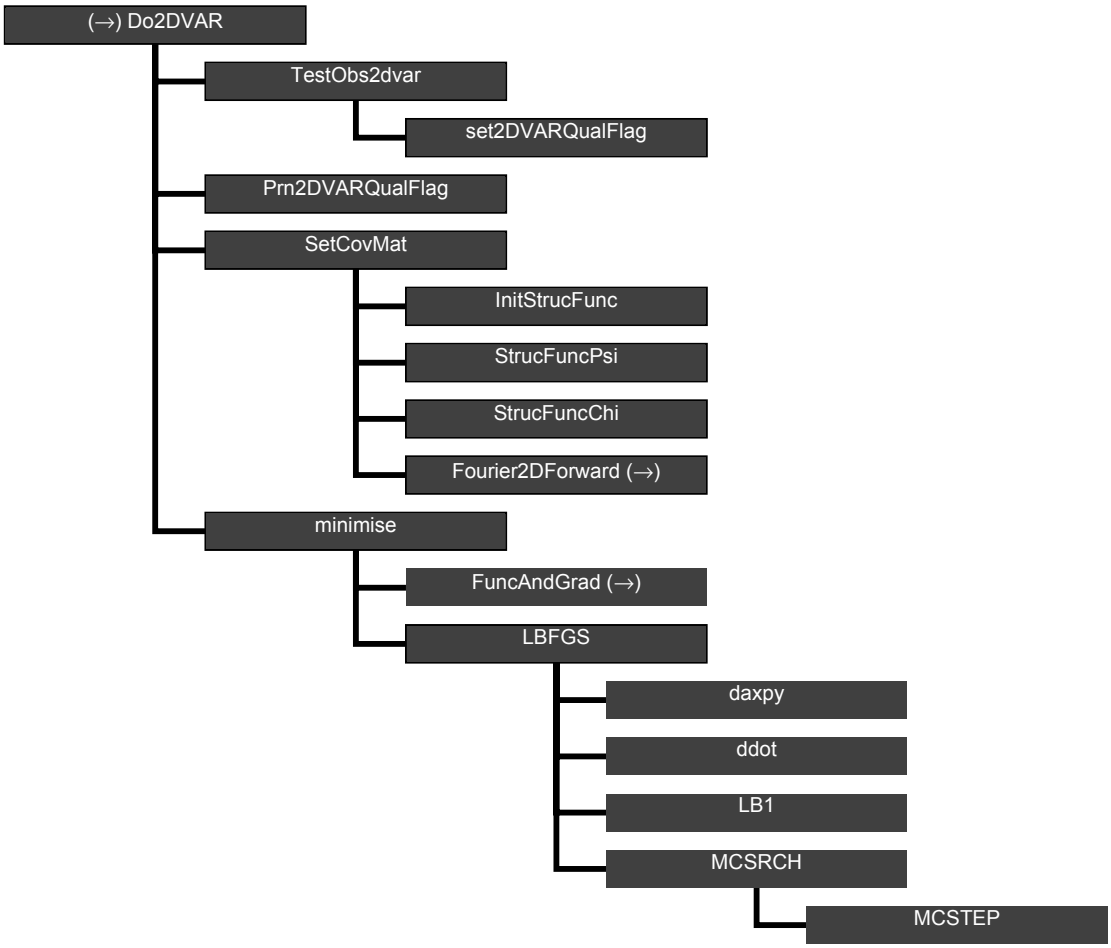


Figure B2.3 Calling tree for AR routine *Do2DVAR*.

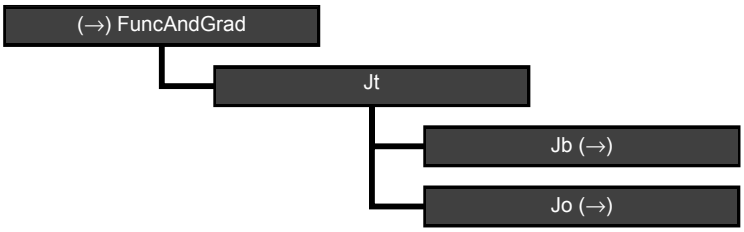


Figure B2.4 Calling tree for AR routine *FuncAndGrad*.

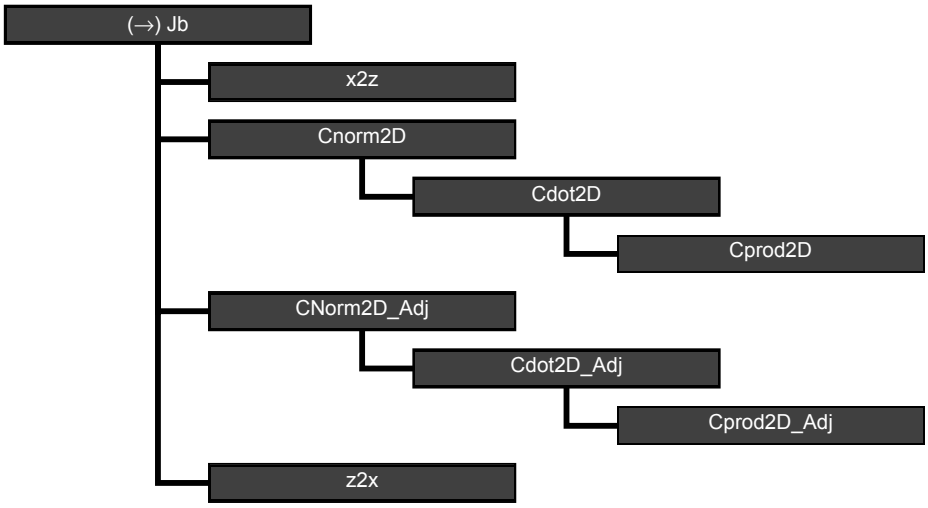


Figure B2.5 Calling tree for AR routine *Jb* (calculation of cost function).

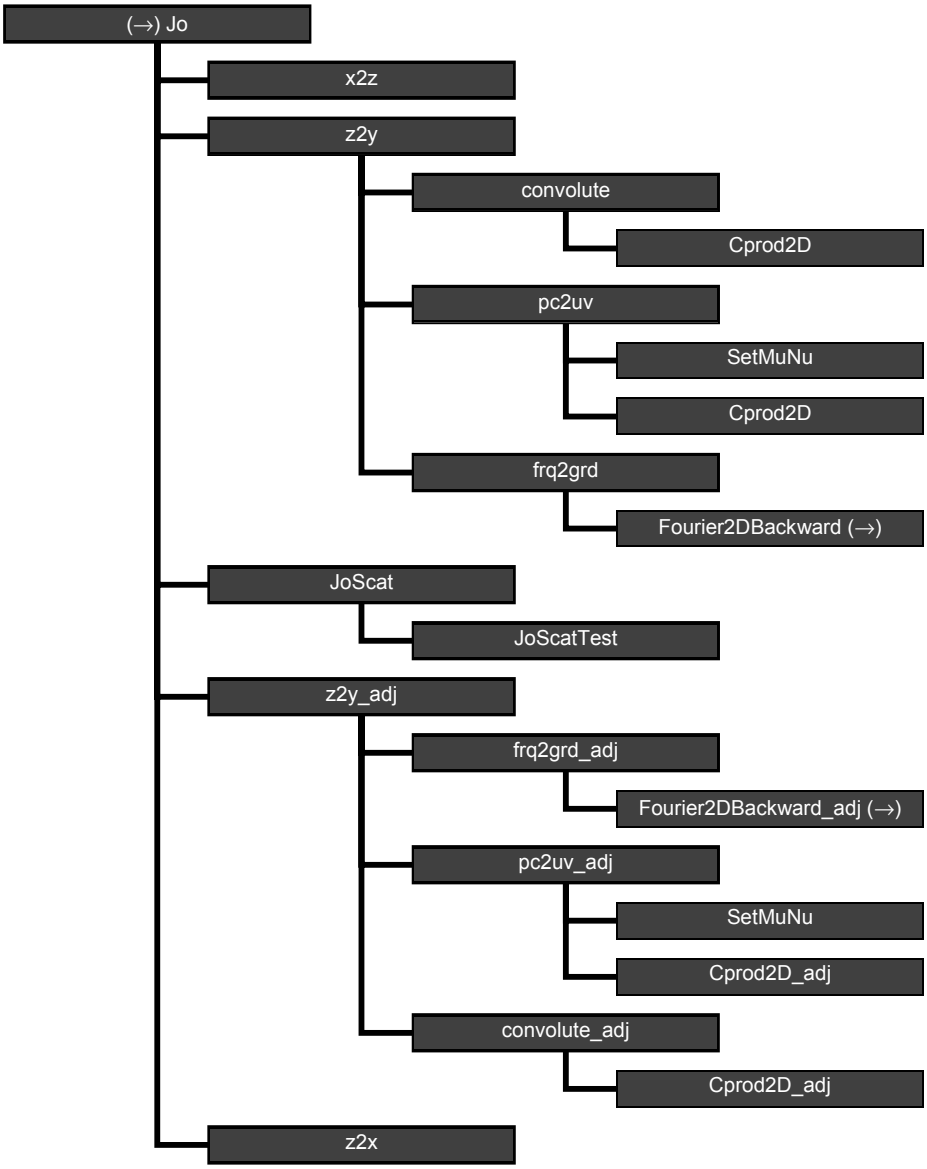


Figure B2.6 Calling tree for AR routine Jo (calculation of cost function).

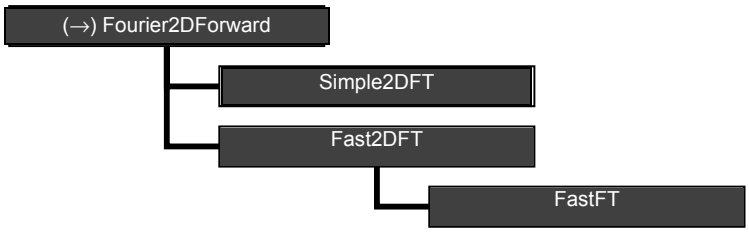


Figure B2.7 Calling tree for AR routine *Fourier2DForward*. The calling tree for routine *Fourier2DBackward* is identical.

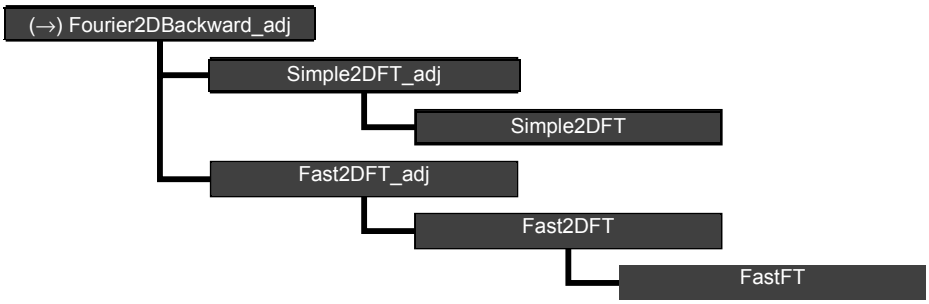


Figure B2.8 Calling tree for AR routine *Fourier2DBackward_adj*.

Appendix B3

Calling tree for BUFR routines

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

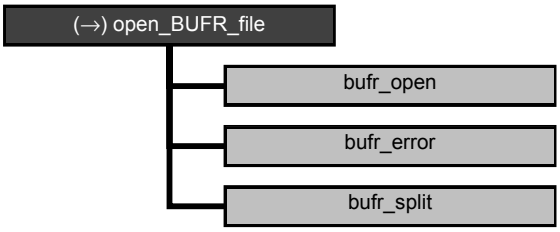


Figure B3.1 Calling tree for BUFR file handling routine *open_BUFR_file*.

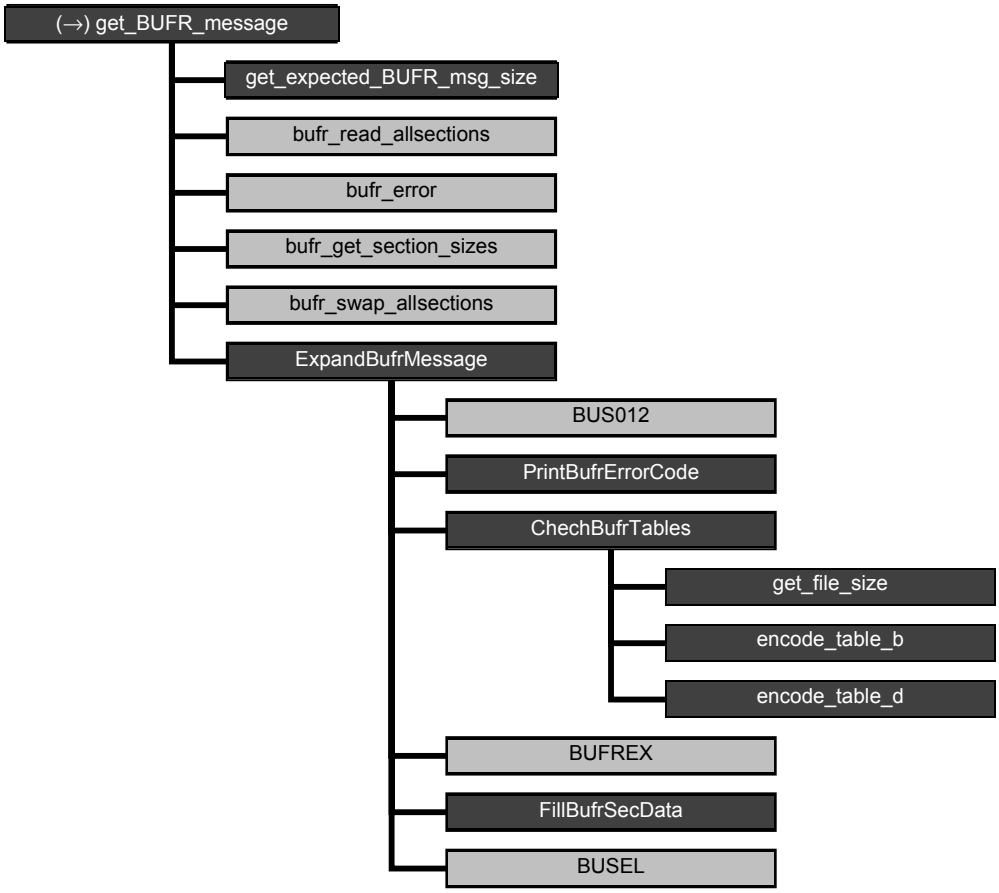


Figure B3.2 Calling tree for BUFR handling routine *get_BUFR_message*.

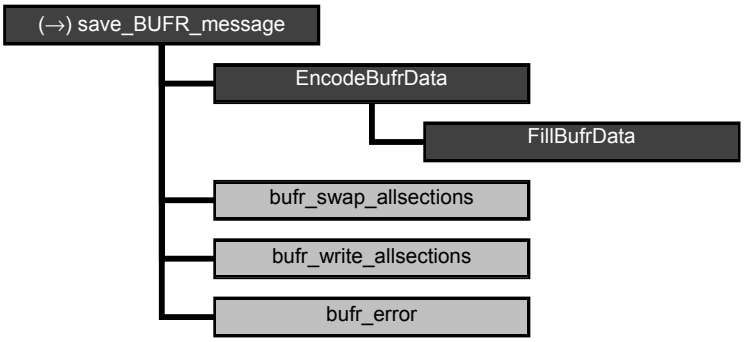


Figure B3.3 Calling tree for BUFR file handling routine *save_BUFR_file*.

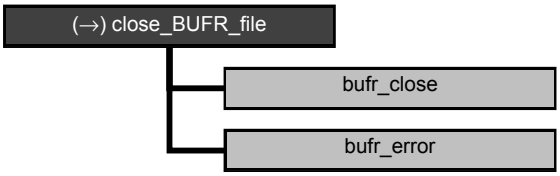


Figure B3.4 Calling tree for BUFR handling routine *close_BUFR_file*.

Appendix C

SeaWinds BUFR data descriptors

Number	Parameter	Unit	Descriptor
001	Satellite Identifier	CODE TABLE	(01007)
002	Direction of Flight	DEGREE TRUE	(01012)
003	Satellite Instrument Identifier	CODE TABLE	(02048)
004	Wind Scatterometer GMF	CODE TABLE	(21119)
005	Software Identification	NUMERIC	(25060)
006	Cross Track Resolution	M	(02026)
007	Along Track Resolution	M	(02027)
008	Orbit Number	NUMERIC	(05040)
009	Year	YEAR	(04001)
010	Month	MONTH	(04002)
011	Day	DAY	(04003)
012	Hour	HOUR	(04004)
013	Minute	MINUTE	(04005)
014	Second	SECOND	(04006)
015	Latitude (Coarse Accuracy)	DEGREE	(05002)
016	Longitude (Coarse Accuracy)	DEGREE	(06002)
017	Time Difference Qualifier	CODE TABLE	(08025)
018	Time to Edge	S	(04001)
019	Along Track Row Number	NUMERIC	(05034)
020	Cross Track Cell Number	NUMERIC	(06034)
021	Seawinds Wind Vector Cell Quality Flag	FLAG TABLE	(21109)
022	Model Wind Direction At 10 M	DEGREE TRUE	(11081)
023	Model Wind Speed At 10 M	M/S	(11082)
024	Number of Vector Ambiguities	NUMERIC	(21101)

Continued on next page

Continued from previous page

Number	Parameter	Unit	Descriptor
025	Index of Selected Wind Vector	NUMERIC	(21102)
026	Total Number of Sigma0 Measurements	NUMERIC	(21103)
027	Seawinds Probability of Rain	NUMERIC	(21120)
028	Seawinds NOF Rain Index	NUMERIC	(21121)
029	Intensity Of Precipitation	KG/M**2/SEC	(13055)
030	Attenuation Correction On Sigma-0 (from Tb)	dB	(21122)
031	Wind Speed At 10 M	M/S	(11012)
032	Formal Uncertainty In Wind Speed	M/S	(11052)
033	Wind Direction At 10 M	DEGREE TRUE	(11011)
034	Formal Uncertainty In Wind Direction	DEGREE TRUE	(11053)
035	Likelihood Computed for Wind Solution	NUMERIC	(21104)
036	Wind Speed At 10 M	M/S	(11012)
037	Formal Uncertainty In Wind Speed	M/S	(11052)
038	Wind Direction At 10 M	DEGREE TRUE	(11011)
039	Formal Uncertainty In Wind Direction	DEGREE TRUE	(11053)
040	Likelihood Computed for Wind Solution	NUMERIC	(21104)
041	Wind Speed At 10 M	M/S	(11012)
042	Formal Uncertainty In Wind Speed	M/S	(11052)
043	Wind Direction At 10 M	DEGREE TRUE	(11011)
044	Formal Uncertainty In Wind Direction	DEGREE TRUE	(11053)
045	Likelihood Computed for Wind Solution	NUMERIC	(21104)
046	Wind Speed At 10 M	M/S	(11012)
047	Formal Uncertainty In Wind Speed	M/S	(11052)
048	Wind Direction At 10 M	DEGREE TRUE	(11011)
049	Formal Uncertainty In Wind Direction	DEGREE TRUE	(11053)
050	Likelihood Computed for Wind Solution	NUMERIC	(21104)
051	Antenna Polarisation	CODE TABLE	(02104)
052	Total Number w.r.t. accumulation or average	NUMERIC	(08022)
053	Brightness Temperature	K	(12063)
054	Standard Deviation Brightness Temperature	K	(12065)
055	Antenna Polarisation	CODE TABLE	(02104)
056	Total Number w.r.t. accumulation or average	NUMERIC	(08022)
057	Brightness Temperature	K	(12063)
058	Standard Deviation Brightness Temperature	K	(12065)
059	Number of Inner-Beam Sigma0 (fwd of sat.)	NUMERIC	(21110)
060	Latitude (Coarse Accuracy)	DEGREE	(05002)
061	Longitude (Coarse Accuracy)	DEGREE	(06002)
062	Attenuation Correction On Sigma-0	dB	(21118)
063	Radar Look (Azimuth) Angle	DEGREE	(02112)
064	Radar Incidence Angle	DEGREE	(02111)
065	Antenna Polarisation	CODE TABLE	(02104)
066	Normalized Radar Cross Section	NUMERIC	(21105)
067	Kp Variance Coefficient (Alpha)	NUMERIC	(21106)
068	Kp Variance Coefficient (Beta)	NUMERIC	(21107)
069	Kp Variance Coefficient (Gamma)	dB	(21114)
070	Seawinds Sigma-0 Quality Flag	FLAG TABLE	(21115)
071	Seawinds Sigma-0 Mode Flag	FLAG TABLE	(21116)
072	Seawinds Land/Ice Surface Flag	FLAG TABLE	(08018)

Continued on next page

Continued from previous page

Number	Parameter	Unit	Descriptor
073	Sigma-0 Variance Quality Control	NUMERIC	(21117)
074	Number of Outer-Beam Sigma0 (fwd of sat.)	NUMERIC	(21111)
075	Latitude (Coarse Accuracy)	DEGREE	(05002)
076	Longitude (Coarse Accuracy)	DEGREE	(06002)
077	Attenuation Correction On Sigma-0	dB	(21118)
078	Radar Look (Azimuth) Angle	DEGREE	(02112)
079	Radar Incidence Angle	DEGREE	(02111)
080	Antenna Polarisation	CODE TABLE	(02104)
081	Normalized Radar Cross Section	NUMERIC	(21105)
082	Kp Variance Coefficient (Alpha)	NUMERIC	(21106)
083	Kp Variance Coefficient (Beta)	NUMERIC	(21107)
084	Kp Variance Coefficient (Gamma)	dB	(21114)
085	Seawinds Sigma-0 Quality Flag	FLAG TABLE	(21115)
086	Seawinds Sigma-0 Mode Flag	FLAG TABLE	(21116)
087	Seawinds Land/Ice Surface Flag	FLAG TABLE	(08018)
088	Sigma-0 Variance Quality Control	NUMERIC	(21117)
089	Number of Inner-Beam Sigma0 (aft of sat.)	NUMERIC	(21112)
090	Latitude (Coarse Accuracy)	DEGREE	(05002)
091	Longitude (Coarse Accuracy)	DEGREE	(06002)
092	Attenuation Correction On Sigma-0	dB	(21118)
093	Radar Look (Azimuth) Angle	DEGREE	(02112)
094	Radar Incidence Angle	DEGREE	(02111)
095	Antenna Polarisation	CODE TABLE	(02104)
096	Normalized Radar Cross Section	NUMERIC	(21105)
097	Kp Variance Coefficient (Alpha)	NUMERIC	(21106)
098	Kp Variance Coefficient (Beta)	NUMERIC	(21107)
099	Kp Variance Coefficient (Gamma)	dB	(21114)
100	Seawinds Sigma-0 Quality Flag	FLAG TABLE	(21115)
101	Seawinds Sigma-0 Mode Flag	FLAG TABLE	(21116)
102	Seawinds Land/Ice Surface Flag	FLAG TABLE	(08018)
103	Sigma-0 Variance Quality Control	NUMERIC	(21117)
104	Number of Outer-Beam Sigma0 (aft of sat.)	NUMERIC	(21113)
105	Latitude (Coarse Accuracy)	DEGREE	(05002)
106	Longitude (Coarse Accuracy)	DEGREE	(06002)
107	Attenuation Correction On Sigma-0	dB	(21118)
108	Radar Look (Azimuth) Angle	DEGREE	(02112)
109	Radar Incidence Angle	DEGREE	(02111)
110	Antenna Polarisation	CODE TABLE	(02104)
111	Normalized Radar Cross Section	NUMERIC	(21105)
112	Kp Variance Coefficient (Alpha)	NUMERIC	(21106)
113	Kp Variance Coefficient (Beta)	NUMERIC	(21107)
114	Kp Variance Coefficient (Gamma)	dB	(21114)
115	Seawinds Sigma-0 Quality Flag	FLAG TABLE	(21115)
116	Seawinds Sigma-0 Mode Flag	FLAG TABLE	(21116)
117	Seawinds Land/Ice Surface Flag	FLAG TABLE	(08018)
118	Sigma-0 Variance Quality Control	NUMERIC	(21117)

Table C.1 List of data descriptors.

Appendix D

ECMWF BUFR data routines

Function	Description
bbuprs0.F	Print BUFR section 0 (?)
bbuprs1.F	Print BUFR section 1 (?)
bbuprs2.F	Print BUFR section 2 (?)
bbuprs3.F	Print BUFR section 3 (?)
bbuprt.F	Print BUFR (?)
bbuprtbox.F	Print BUFR box (?)\\
buaug.F	Update augmented BUFR table B
bubox.F	??
bucomp.F	Pack number of subsets in a compressed form
bucrkey.F	Extract elements needed for RDB key definition(update)
bucrekey.F	Extract elements needed for RDB key definition
buedd.F	Expand section 3 of BUFR message
buens0.F	Pack section 0 of BUFRsage
buens1.F	Pack section 1 of BUFR message
buens2.F	Pack section 2 of BUFR message
buens3.F	Pack section 3 of BUFR message
buens4.F	Pack preliminary items/data of sect.4 of BUFR message
buens5.F	Pack sect.5 of BUFR message
buepmrk.F	Process marker operator, replace with table B descriptor
buepmrkc.F	Process marker operator, replace with table B descriptor
buepwt.F	Updates working tables (name, unit, scale, ref, data width)
buepwtc.F	Updates working tables (name, unit, scale, ref, data width)
buerr.F	Print error code
buetab.F	Load BUFR table B, D and C according to BUFR code
buetd.F	Expand sect.3 of BUFR message

Continued on next page

Continued from previous page

Function	Description
buetdr.F	Solve BUFR table D reference
buevar.F	Initialize constants and variables
buexs0.F	Expand section 0 of BUFR message
buexs1.F	Expand section 1 of BUFR message
buexs2.F	Expand section 2 of BUFR message
buexs3.F	Expand section 3 of BUFR message
buexs3p.F	Expand section 3 of BUFR message (preliminary items)
buexs4.F	Expand section 4 of BUFR message
buexs5.F	Expand section 5 of BUFR message
bufren.F	Encode BUFR message
bufrex.F	Decode BUFR message into fully expanded form
bugbts.F	Load BUFR table B, D and C according to BUFR code
bugetbm.F	Create bit map to resolve marker operators
buivar.F	Initialize constants and variables
bunexs.F	Sets word/bit pointers at the start of next BUFR sect
bunpck.F	Unpack bit string
bunpks.F	Unpack bit string of KSIZE bits
buoctn.F	Calculate number of octets from bit position
buoper.F	Process BUFR operator
buoperc.F	Process BUFR operator
bupck.F	Pack value *KS* in *KSI* bits
bupkey.F	Pack local ECMWF information (rdb key)
bupks.F	Pack bit string of KSIZE bits
bupmrk.F	Process marker operator, replace with table B descriptor
buprco.F	Process BUFR operator
buprq.F	Sets variable KPMISS,KPRUS into common block
buprs0.F	Print section 0 of BUFR message
buprs1.F	Print section 1 of BUFR message
buprs2.F	Print section 2 of BUFR message (expanded RDB key)
buprs3.F	Print section 3 of BUFR message
buprt.F	Print expanded BUFR message
buprtbox.F	Print boxed expanded BUFR message
burep.F	Resolve data descriptor replication problem
burepc.F	Resolve data descriptor replication problem
burqc.F	Create parameters needed for partial expansion of BUFR
burquc.F	Create parameters needed for partial expansion of BUFR
bus012.F	Expands sections 0, 1, and 2 of BUFR message
busel.F	Returns Data Descriptors as in Section 3 of BUFR
buset.F	Set flags in common block (?)
busrp.F	Resolve data descriptor replication problem
busrq.F	Set BUFR table B references for partial expansion
bustdr.F	Solve BUFR table D reference
buuatb.F	Update augmented BUFR table B
buukey.F	Expands local ECMWF information from sect.2
buunp.F	Unpack bit string of KSIZE bits
buunps.F	Unpack bit string of KSIZE bits
buupwt.F	Updates working tables (name, unit, scale, ref, data width)
buxdes.F	Expand data descriptors to show user's template

Continued on next page

Continued from previous page

Function	Description
fmmh.F	Find max/min latitude/longitude
setlalo.F	Return indices for latitude and longitude

Table D.1 List of ECMWF BUFR routines.

Appendix E

Acronyms

Name	Description
AR	Ambiguity Removal
BUFR	Binary Universal Form for the Representation of data
C-band	Radar wavelength at about 5 cm
ECMWF	European Center for Medium-range Weather Forecasts
EUMETSAT	European Organization for the Exploitation of Meteorological Satellites
GMF	Geophysical model function
HIRLAM	High resolution Local Area Model
KNMI	Koninklijk Nederlands Meteorologisch Instituut (Royal Dutch Meteorological Institute)
Ku-band	Radar wavelength at about 2 cm
L1b	Level 1b product
LUT	Look up table
MLE	Maximum Likelihood Estimator
MSS	Multiple Solution Scheme
NRCS	Normalized radar cross-section (σ_0)
NWP	Numerical Weather Prediction
QC	Quality Control
RFSCAT	Rotating fan beam scatterometer
RMS	Root mean square
SAF	Satellite Application Facility
WVC	Wind vector cell, also called node or cell

Table E.1 List of acronyms.