Aeolus Level 2b Processor

Top Level Design

Version 0.8

9th May 2008

Aeolus Level 2B Processor Top Level Design (NWP SAF style).

This documentation was developed within the context of the ESA ADM-Aeolus L2B project. It provides documentation in the style used by the EUMETSAT Satellite Application Facility on Numerical Weather Predication (NWP SAF) to facilitate NWP user implementation.

The partners in the Aeolus L2B processor project are the ECMWF, KNMI, and Météo-France.

Change Record

Issue.	Date	Modified pages/sections	Observations	Name
0.1	27-Jun-2007	--	Draft	J. de Kloe
0.2	10-Jul-2007	--	1st revision	A. Stoffelen
0.3	16-Jul-2007	section 2	split the description in section 2 in 2 parts, one for the standalone and one for the subroutine version of the L2BP	J. de Kloe
0.4	18-Jul-2007	section 4, as a standalone discussion document	Initial version: TLD for L2BP subroutine	J. de Kloe
0.41	30-Jul-2007	section 4	Made hidden variables explicit in dataflow (RBC in particular). Expanded TLD to include all cases: reading from file or memory the L1B and AMD data, writing to file and/or memory the output L2B data. Took out from primary processing the matching of AMD so that primary processing may be done in parallel.	P. Poli
0.42	31-Jul-2007	section 4	some small comments	J. de Kloe
0.43	21-Aug-2007	section 4	Separate the document in 2 chapters to include current aeolus L2BP design	P. Poli
0.44	22-Aug-2007	section 4	Modify the current design so that the first tasks of L2BP_primary are done inside the new routine aeolus_l2bp_prepare_primary. Note that AMD and iBRC are removed from the list of necessary arguments for L2BP_primary, while WorkingData is now required	P. Poli
0.45	29-Aug-2007	section 4	Remove L1B Working_Array as in David Tan's branch Aug29	P. Poli
0.46	04-Sep-2007	section 4	Modified dataflow for L1B data and handling of L1B and L2B headers. Added import possibility for L1B and L1B_Headers. Added export possibility for L1B, L1B_Headers, and L2B_Headers.	P. Poli
0.47	16-Oct-2007	section 4	import/export possibility for AMD data and AMD_Headers. Added export possibility for L2B data. Modified list of arguments and contents of routine aeolus_l2bp_fill_L2B_Headers (JdK).	P. Poli
0.48	19-Oct-2007	section 4	Removed target design section. Retained final design part only.	P. Poli
0.5	26-Oct-2007	section 4 inserted in main document	Merged sections 1-3 and new section 4	J. de Kloe
0.6	30-Nov-2007	section 4	Added detailed list of arguments with variable types. Updated top level design in line with latest L2BP version.	P. Poli
0.7	19-Dec-2007	Section 4	Added an example program running the subroutine version, which also allows perfect comparison of the results with the standalone version	P. Poli
0.8	9 May 2008	Section 4 Reference documents	Update design and interfaces in line with L2BP version 1.33 Update (comments from ESA)	P. Poli

Table of Contents

1. Introduction and scope

This document defines the Top Level Design (TLD) of the Aeolus Level 2B Processor (L2BP) software package, in accordance with the requirements of the NWP SAF.

Note that this ESA project has been documented according to the ECSS guidelines, and to minimise the amount of double documentation, the NWP SAF documents will refer to these ECSS documentation produced for ESA as much as possible.

The TLD should show "the modular structure of the deliverable, including the function of each module and the data flow between modules".

The original purpose of a TLD (together with the Module Design) is to give the programmer sufficient information to start implementation of the software.

However, since most of the software has already been implemented, and documented according to the standards imposed by ESA, this is no longer applicable. Therefore this document refers to what has already been documented, and describes the interfaces of the subroutines that are needed to use the L2BP from within an IFS.

1.1Reference documents

[RD-1] ECSS-E-40B Space System Software Engineering, draft version, 28-June-2000

[RD-2] AE-DD-ECMWF-L2BP-001_20070223_DD_Iss1.0.pdf

[RD-3] MFG_L2BP_fortran_structures_20080509.doc

[RD-4] NWP SAF Aeolus Level 2b Processor Product Specification, version 0.3, 16 July 2007

[RD-5] AE-IF-ECMWF-L2BP-002_20080116_ExtICD_Iss1.32.pdf

[RD-6] AE_TN_ECMWF_L2BP_0023_20070223_ATBD_V2.1.pdf

[RD-7] AE-TN-ECMWF-L2BP-0072_20070223_EE2BUFRGuide_Iss1.1.pdf

[RD-8] AE-MA-ECMWF-L2BP-001_20080229_SUM_Iss1.33.pdf

[RD-9] ADM Aeolus Mission: Guide on applying L2B processed winds

1.2Acronyms

ECSS	European Cooperation for Space Standardization
IFS	Integrated Forecasting System
L2BP	ADM-Aeolus Level 2B Processor
TLD	Top Level Design

2. L2BP Software Design

The detailed software design, including description of the functional sets of modules, sorted by subdirectory in the L2BP source tree, is described in [RD-2]. The description of all scientific algorithms used by the L2BP is described in [RD-6]. Its use and an overview of its functionality is detailed in [RD-8].

This software may be used in two different ways:

- as a standalone program, using files for input and output of data. See section 2.1 below for details.
- as a subroutine within a larger software system (like the ECMWF IFS), using data structures in memory for input and output of data. See section 2.2 below for details.

[RD-9] provides an overview of the user considerations for applying L2B winds in NWP, including a flow-diagram to help deciding which version of the software is most appropriate for a specific usage.

2.1 Standalone L2BP version

2.1.1 File Specifications

References to all documents related to specifications of input and output files are given in the Product Specification document [RD-4], section 5.

2.1.2 Data Flows

Data flows in the form of input and output files, and output to stdout and stderr are detailed in [RD-5].

2.1.3 File Specifications

A listing of all needed input files and produced output files, and references to all documents related to specifications of input and output files are given in the Product Specification document [RD-4], section 5.

2.2 Subroutine L2BP version

2.2.1 Data Structures

All information on data structures needed to integrate the L2BP into an IFS are detailed in [RD-3].

2.2.2 Set-up Functions

When integrating the L2BP software in an IFS a number of setup, working and closing subroutines need to be called in a specific order. Details on these subroutines and their use are given in section 4 below.

2.2.3 Data Flows and interfaces

Data flows in the form of input and output data structures are detailed in [RD-3]. The output to stdout and stderr is detailed in [RD-5]. These data flows form the interface used by the subroutine version of the software.

2.2.4 File Specifications

References to all documents related to specifications of input and output files are given in the Product Specification document [RD-4], section 5.

3.EE2BUFR Conversion Tool Software Design

This tool is intended to be used only as standalone tool, so no integration in an IFS system is foreseen. Therefore no description of Set-up functions or data structures is needed. Its use and functionality is detailed in [RD-7]. This tool consists largely of modules that where written for the L2BP. Therefore a description of the functional sets of modules, sorted by subdirectory in the L2BP source tree, are described in [RD-2].

3.1 Data Flows

Data flows in the form of input and output files, and output to stdout and stderr are detailed in [RD-7].

3.2 File Specifications

References to all documents related to specifications of input and output files are given in the Product Specification document [RD-4], section 5.

4.Design and interfaces

4.1 Introduction

This section details the TLD of the following aeolus L2BP configuration: client_rm7_CY32R3_Dec21

4.2 Program L2B_processor (standalone L2BP)

The program described below is the actual L2BP program calling the L2BP subroutines. The Fortran90 definitions of the variables involved to perform this processing are listed in an array below, and detailed in [RD3].

The program below works for standalone processing and the import (export) subroutines as indicated here will read the input data from (write the output data to) EE files.

Variables :

Type	Name
DATASTRUCTURES	
TYPE(L1B_HDR_DataType)	L1B_Headers
TYPE(L1B_BRC_DataType)	L1B
TYPE(L2BC_HDR_DataType)	L2B_Headers
TYPE(L2BC_BRC_DataType), dimension(:), pointer	L2B
TYPE(AMD_HDR_DataType)	AMD_Headers
TYPE(AMD_BRC_DataType), dimension(:), pointer	AMD
TYPE(RBC_DataType)	RBC
TYPE(WorkingDataType)	WorkingData
TYPE(JobOrderData_type)	JobOrderData
TYPE(L2BP_Settings_type)	L2BP_Settings
LOOP COUNTERS	
integer	**AMD_file**
integer	**iBRC**
integer	AMD_iBRC
integer	L2B_iBRC
INTERNAL VARIABLES: INTEGERS, CHARACTER STRINGS, DATES	
integer	Num_BRC
integer	Num_AMD_BRC
integer(IntAus)	Num_L2B_BRCs
integer	error_flag
integer(IntAI)	lat_start

integer(IntAl)	lat_stop
integer(IntAl)	lon_start
integer(IntAl)	lon_stop
integer(IntAuc)	M_Meas0
integer(IntAuc)	M_Mie0
integer(IntAuc)	M_Rayleigh0
integer(IntAuc)	N_Meas_to_write
integer(IntAuc)	N_Obs_Mie_to_write
integer(IntAuc)	N_Obs_Rayleigh_to_write
character(len=256)	filename_JobOrder
type(DateTimeType)	DateTimeStart
type(DateTimeType)	DateTimeStop
INTERNAL VARIABLES : LOGICALS	
logical	do_standalone_processing=.TRUE.or .FALSE.
logical	do_primary_processing=.TRUE.
logical	ProcessBRC
logical	IsWithinTimeRange

Contents:

```
! 1.1 initialization, get L2BP settings and filenames, load auxiliary instrument data (calibration, look-up
! tables etc..)

do_standalone_ processing = .TRUE.

call aeolus_l2bp_setup (filename_JobOrder, JobOrderData, L2BP_Settings, RBC, error_flag,
        standalone_version=do_standalone_processing)

! 2.0 loop on all the AMD files

AMD_file_loop: DO AMD_file = 1, L2BP_Settings%NumNWPfiles

        ! 2.1 prepare processing for AMD file number AMD_file: get L1B_Headers,
        ! allocate L2B array, and read AMD data

        call InitLatLonDateTimeRanges ()

        call aeolus_l2bp_import_L1B_Headers (L1B_Headers,
                Num_BRC, error_flag,
                L1B_filename= L2BP_Settings%L1Bfile%name)

        allocate (L2B(Num_BRC))

        DO iBRC = 1, Num_BRC

                call Init_L2BC_BRC_DataStructure (L2B(iBRC), 0, 0, 0, 0, error_flag)

        ENDDO

        call aeolus_l2bp_import_AMD_Headers (AMD_Headers,
                Num_AMD_BRC, error_flag,
                AMD_filename=L2BP_Settings%NWPfile(AMD_file)%name)

        allocate (AMD(Num_AMD_BRC))
```

```
DO AMD_iBRC = 1, Num_AMD_BRC
        call aeolus_l2bp_import_one_AMD_BRC (AMD_Headers,
                AMD(AMD_iBRC),
                AMD_iBRC, error_flag,
                AMD_filename= L2BP_Settings%NWPfile(AMD_file)%name)
ENDDO
! 2.2 start loop processing for all BRCs corresponding to AMD file number AMD_file
L2B_iBRC = 0
brc_loop: DO iBRC = 1, Num_BRC
        ! 2.2.1 get the instrument data for that particular BRC, and verify time matching
        call aeolus_l2bp_import_one_L1B_BRC (L1B_Headers, L1B, iBRC,
                error_flag, L1B_filename=L2BP_Settings%L1Bfile%name)
        ProcessBRC = .TRUE.
        call CheckWithinTimeRange
                (L1B%Geoloc_ADS%start_of_obs_datetime,
                L2BP_Settings%L1Bfile%start, L2BP_Settings%L1Bfile%stop,
                IsWithinTimeRange, error_flag)
        IF (.not.(IsWithinTimeRange)) ProcessBRC=.FALSE.
        call CheckWithinTimeRange
                (L1B%Geoloc_ADS%start_of_obs_datetime,
                L2BP_Settings%NWPfile(AMD_file)%start,
                L2BP_Settings%NWPfile(AMD_file)%stop,
                IsWithinTimeRange, error_flag)
        IF (.not.(IsWithinTimeRange)) ProcessBRC=.FALSE.
        ! 2.2.2 do actual processing
        IF (ProcessBRC) THEN
                ! 2.2.2.1 start L2B processing, including finding a matching profile for the BRC
                ! in the meteorological data
                L2B_iBRC = L2B_iBRC + 1
                call aeolus_l2bp_prepare_primary (L1B, L2B(L2B_iBRC),
                        AMD, iBRC,
                        L2BP_Settings%L2B_AuxPar,
                        WorkingData, error_flag)
                call L2BP_primary (L1B, L2B(L2B_iBRC), WorkingData,
                        RBC, L2BP_Settings%L2B_Auxpar, error_flag)
                ! 2.2.2.2 update information regarding the date/time ranges
                call DetermineLatLonDateTimeRanges (L1B, error_flag)
        ENDIF
        ! 2.2.3 clear instrument data from memory
        call aeolus_l2bp_export_one_L1B_BRC (L1B, error_flag)
ENDDO brc_loop
! 2.3 gather statistics on the processed BRCs and fill L2B_Headers
Num_L2B_BRCs = int(L2B_iBRC, IntAus)
call GetLatLonRanges (lat_start, lat_stop, lon_start, lon_stop, error_flag)
```

```
call GetDateTimeRange (DateTimeStart, DateTimeStop, error_flag)
call aeolus_l2bp_fill_L2B_Headers(L1B_Headers, L2B, L2B_Headers,
        L2BP_Settings,
        Num_L2B_BRCs, DateTimeStart, DateTimeStop,
        lat_start, lat_stop, lon_start, lon_stop,
        M_Meas0, M_Mie0, M_Rayleigh0, error_flag)
N_Meas_to_write = M_Meas0
N_Obs_Mie_to_write = M_Mie0
N_Obs_Rayleigh_to_write = M_Rayleigh0
! 2.4 write the L2B data to file
call WriteL2BCfile (L2B_iBRC, L2B_Headers, L2B, N_Meas_to_write,
        N_Obs_Mie_to_write, N_Obs_Rayleigh_to_write,
        L2BCresultdir=L2BP_Settings%L2Bresultdir,
        L1Bfilename=L2BP_Settings%L1Bfile%name,
        NWPfilename=L2BP_Settings%NWPfile(AMD_file)%name,
        RBCfilename=L2BP_Settings%RBCfilename,
        CLMfilename="[undefined]", CALfilename="[undefined]",
        ParamFilename=L2BP_Settings%L2BAuxPar_filename,
        error_flag=error_flag)
! 2.5 clear L1B_Headers, L2B_Headers, and all AMD data from memory
call aeolus_l2bp_export_L1B_Headers(L1B_Headers, error_flag)
call aeolus_l2bp_export_L2B_Headers (L2B_Headers, error_flag)
IF (associated(L2B)) THEN
        DO iBRC = 1, size(L2B)
                call aeolus_l2bp_export_one_L2B_BRC (L2B(iBRC), error_flag)
        ENDDO
ENDIF
deallocate (L2B)
call aeolus_l2bp_export_AMD_Headers (AMD_Headers, error_flag)
IF (associated(AMD)) THEN
        DO AMD_iBRC = 1, size(AMD)
                call aeolus_l2bp_export_one_AMD_BRC (AMD(AMD_iBRC),
                        error_flag)
        ENDDO
ENDIF
deallocate (AMD)
ENDDO AMD_file_loop
! 3.0 clean-up remaining items from memory
call aeolus_l2bp_unsetup (JobOrderData, L2BP_Settings, RBC)
```

4.3 Program application_client_example (subroutine version of the L2BP)

A user may want to call the L2BP from inside his own application directly. In such case it is very likely that he wants all the input data (auxiliary data, instrument data, meteorological data) to be loaded before calling the L2BP, rather than loading them as the processingprogresses.

In such case one can refer to the example program called application_client_example.

The only differences with respect to the standalone program L2B_processor are that:

1. All features only relevant for the ground segment nominal processing are removed for clarity (ability to run a pre-processing, to generate a product list etc…)
2. In the program area *1.1* the variable do_standalone_processing is set to .FALSE.
3. The L1B and AMD data are loaded in one pass by adding the following call in what would be area *1.2* of the program L2B_processor:

   call aeolus_l2bp_load_data_to_vdas (L2BP_Settings, error_flag)

   The L1B and AMD data are hereby stored and saved in a module “virtual_das” (for virtual data assimilation system) and then further available to the following routines:

   - aeolus_l2bp_import_L1B_Headers
   - aeolus_l2bp_import_one_L1B_BRC
   - aeolus_l2bp_import_AMD_Headers
   - aeolus_l2bp_import_one_AMD_BRC

   Consequently the later calls to these four routines in the respective program areas *2.1*, *2.1*, *2.2.1* and *2.2* do not invoke the optional argument “filename=”.

   For symmetry there is also an additional call to clear the “virtual_das” memory in what would be area *3.1* of the program L2B_processor:

   call aeolus_l2bp_clear_vdas (error_flag)

   Should a user want to export the output L2B data to his own application (rather than getting an output L2B file) he needs to modify

   - aeolus_l2bp_export_L2B_Headers
   - aeolus_l2bp_export_one_L2B_BRC

Furthermore, the following optional points-of-exit are available, should one want to export to the application the datastructures used in input, with the help of the following routines:

- aeolus_l2bp_export_L1B_Headers
- aeolus_l2bp_export_one_L1B_BRC
- aeolus_l2bp_export_AMD_Headers
- aeolus_l2bp_export_one_AMD_BRC

4.4 aeolus_l2bp_setup

Arguments:

Type	Intent	Name
character(len=256)	out	filename_JobOrder
TYPE(JobOrderData_type)	out	JobOrderData

TYPE(L2BP_Settings_type)	out	L2BP_Settings
TYPE(RBC_DataType)	out	RBC
integer	out	error_flag
logical, optional	in	standalone_version

Contents:

call Init_L2B_Proc_Settings (L2BP_Settings, error_flag)

filename_JobOrder = 'JobOrder.AeolusL2BP.xml'

IF (is_standalone_version) call Get_JobOrderName_from_CmdLine (filename_JobOrder, error_flag)

call Init_JobOrderData (JobOrderData)

call ReadJobOrderFile (JobOrderData, filename_JobOrder, Task_Number, error_flag,
workaround=.TRUE.)

call Copy_JobOrderData_Items (JobOrderData, L2BP_Settings, error_flag)

call InitRead_L2B_AuxPar_fileModule (error_flag)

call Read_L2B_AuxPar_file (L2BP_Settings%L2B_AuxPar,
L2BP_Settings%L2BAuxPar_filename, error_flag)

IF (is_standalone_version) call Get_Additional_Cmdline_Options (L2BP_Settings,
error_flag)

call Set_Derived_L2B_Proc_Settings (L2BP_Settings, error_flag)

call Init_Modules (L2BP_Settings, error_flag)

IF (do_primary_processing) THEN

call Load_Rayl_Brill_Tables (RBC, L2BP_Settings%RBCfilename, error_flag)

ENDIF

4.5 aeolus_l2bp_import_L1B_Headers

Arguments:

Type	Intent	Name
TYPE(L1B_HDR_DataType)	out	L1B_Headers
integer	out	Num_BRC
integer	out	error_flag
character(len=*), optional	in	L1B_filename

Externals:

If no file name is provided, this routine assumes that the L1B data are already loaded in memory in the module virtual_das. In that case, the following structure is expected to be available: DAS_L1B_Headers.

See section 4.9 to see how to that structure is filled.

Contents:

```
IF (present(L1B_filename) THEN
        call Init_L1B_HDR_DataStructure (L1B_Headers, error_flag)
        ! read from file
        call ReadL1Bfile (L1B_Headers, L1B_filename, error_flag,
                which_DSR=1, ReadHeaders=.TRUE.)
ELSE
        ! import from memory
        IF (.not.associated(DAS_L1B_Headers)) THEN
                call logmsg(log_error,"aeolus_l2bp_import_L1B_Headers: ERROR!! "//&
                        "L1B_Headers data are not available in the virtual DAS")
                error_flag = error_programming
                return
        ELSE
                L1B_Headers%headers_are_available = DAS_L1B_Headers%headers_are_available
                L1B_Headers%FH_is_available = DAS_L1B_Headers%FH_is_available
                L1B_Headers%MPH_is_available = DAS_L1B_Headers%MPH_is_available
                L1B_Headers%SPH_is_available = DAS_L1B_Headers%SPH_is_available
                L1B_Headers%FH  => DAS_L1B_Headers%FH
                L1B_Headers%MPH => DAS_L1B_Headers%MPH
                L1B_Headers%SPH => DAS_L1B_Headers%SPH

        ENDIF
ENDIF
Num_BRC=L1B_Headers%SPH%Geolocation_DSD%nr_of_dataset_records
```

4.6 aeolus_l2bp_import_AMD_Headers

Arguments:

Type	Intent	Name
TYPE(AMD_HDR_DataType)	out	AMD_Headers
integer	out	Num_AMD_BRC
integer	out	error_flag
character(len=*), optional	In	AMD_filename
integer, optional	in	iAMD_file

Externals:

If no file name is provided, this routine assumes that the L1B data are already loaded in memory in the module virtual_das. In that case, the following structure is expected to be available:

DAS_AMD_Headers.

See section 4.9 to see how to that structure is filled.

Contents:

```
IF (present(AMD_filename)) THEN
        call Init_AMD_HDR_DataStructure (AMD_Headers, error_flag)
        ! read from file
        call ReadAMDfile (AMD_Headers, AMDfilename, error_flag,
                which_DSR = 1, ReadHeaders=.TRUE.)
ELSE
        !  import from memory
        IF (.not.associated(DAS_AMD_Headers)) THEN
                call logmsg(log_error,"aeolus_l2bp_import_AMD_Headers: ERROR!!
                        "//&"AMD_Headers data are not available in the virtual DAS")
                error_flag = error_programming
                return
        ELSE IF (.not.present(iAMD_file)) THEN
                call logmsg(log_error,"aeolus_l2bp_import_AMD_Headers: ERROR!! "//&
                        "iAMD_file argument is missing")
                error_flag = error_programming
                return
        ELSE
                IF (iAMD_file > size(DAS_L1B_BRCs)) THEN
                        call logmsg(log_error,"aeolus_l2bp_import_AMD_Headers: ERROR!! "//&
                                "requested iAMD_file is beyond available range")
                        error_flag = error_programming
                        return
                ELSE
                        AMD_Headers%FH_is_available  =
                             DAS_AMD_Headers(iAMD_file)%FH_is_available
                        AMD_Headers%MPH_is_available =
                             DAS_AMD_Headers(iAMD_file)%MPH_is_available
                        AMD_Headers%SPH_is_available =
                             DAS_AMD_Headers(iAMD_file)%SPH_is_available
                        AMD_Headers%FH  => DAS_AMD_Headers(iAMD_file)%FH
                        AMD_Headers%MPH => DAS_AMD_Headers(iAMD_file)%MPH
                        AMD_Headers%SPH => DAS_AMD_Headers(iAMD_file)%SPH

                ENDIF
ENDIF
call GetIntAI (AMD_Headers%SPH%GeoADS1_DSD%num_dsr,
```

Temp_Num_AMD_BRC, error_flag)

Num_AMD_BRC = int(Temp_Num_AMD_BRC)

4.7 aeolus_l2bp_import_one_L1B_BRC

Arguments:

Type	Intent	Name
TYPE(L1B_HDR_DataType)	inout	L1B_Headers
TYPE(L1B_BRC_DataType)	out	L1B
integer	in	iBRC
integer	out	error_flag
character(len=*), optional	in	L1B_filename
logical, optional	in	flag_init_only

Externals:

If no file name is provided, this routine assumes that the L1B data are already loaded in memory in the module virtual_das. In that case, the following structure is expected to be available: DAS_L1B_BRCs.

See section 4.9 to see how to that structure is filled.

Contents:

```
do_init_only = .FALSE.
IF (present(flag_init_only)) do_init_only=flag_init_only

IF (do_init_only.or.present(L1B_filename)) THEN
        ! initialise the L1B BRC datastructure (no headers, datasets only)
        call Init_L1B_BRC_DataStructure (L1B, error_flag)
ENDIF
IF (.not. (do_init_only)) THEN
        IF (present(L1B_filename)) THEN
                ! read from file
                call ReadL1Bfile (L1B_Headers, L1B_filename, error_flag,
                        L1B=L1B, which_DSR=iBRC, ReadHeaders=.FALSE.,
                        ReadGeoloc=.TRUE., ReadPCD=.TRUE., ReadGWD=.TRUE.,
                        ReadMeas=.TRUE., ReadUS=.TRUE., ReadWV=.TRUE.,
                        ReadCal=.TRUE.)
        ELSE
                !  import from memory
                IF (.not.associated(DAS_L1B_BRCs)) THEN
                        call logmsg(log_error,"aeolus_l2bp_import_one_L1B_BRC: ERROR!! "//& "L1B
                                data are not available in the virtual DAS")
```

```
                error_flag = error_programming
                return
            ELSE
                IF (iBRC > size(DAS_L1B_BRCs)) THEN
                    call logmsg(log_error,"aeolus_l2bp_import_one_L1B_BRC: ERROR!!
                      "//&
                      "requested BRC is beyond available range")
                    error_flag = error_programming
                    return
                ELSE
                    L1B%Geoloc_ADS => DAS_L1B_BRCs(iBRC)%Geoloc_ADS
                    L1B%PCD_ADS => DAS_L1B_BRCs(iBRC)%PCD_ADS
                    L1B%GWD_ADS => DAS_L1B_BRCs(iBRC)%GWD_ADS
                    L1B%Meas_ADS => DAS_L1B_BRCs(iBRC)%Meas_ADS
                    L1B%Cal_ADS => DAS_L1B_BRCs(iBRC)%Cal_ADS
                    L1B%US_MDS => DAS_L1B_BRCs(iBRC)%US_MDS
                    L1B%WV_MDS => DAS_L1B_BRCs(iBRC)%WV_MDS

                ENDIF
            ENDIF
        ENDIF
ENDIF
```

4.8 aeolus_l2bp_import_one_AMD_BRC

Arguments:

Type	Intent	Name
TYPE(AMD_HDR_DataType)	inout	AMD_Headers
TYPE(AMD_BRC_DataType)	out	AMD
integer	in	AMD_iBRC
integer	out	error_flag
character(len=*), optional	in	AMD_filename
integer, optional	in	iAMD_file
logical, optional	in	flag_init_only

Externals:

If no file name is provided, this routine assumes that the L1B data are already loaded in memory in the module virtual_das. In that case, the following structure is expected to be available: DAS_AMD_BRCs.

See section 4.9 to see how to that structure is filled.

Contents:

```
do_init_only = .FALSE.
IF (present(flag_init_only)) do_init_only=flag_init_only

IF (do_init_only.or.present(AMD_filename)) THEN
        call Init_AMD_BRC_DataStructure (AMD, 0, error_flag)
ENDIF

IF (.not.(do_init_only)) THEN
        IF (present(AMD_filename)) THEN
                ! read from file
                call ReadAMDfile (AMD_Headers, AMD_filename, error_flag,
                        AMD=AMD, whichDSR=AMD_iBRC, ReadHeaders=.FALSE.,
                        Read_Geoloc_offnadir=.TRUE.,
                        ReadGeoloc_nadir=. TRUE,
                        ReadMet_offnadir=. TRUE.,
                        ReadMet_nadir=. TRUE.)
        ELSE
                !  import from memory
                IF (.not.associated(DAS_AMD_BRCs)) THEN
                        call logmsg(log_error,"aeolus_l2bp_import_one_AMD_BRC: ERROR!! "//&
                        "AMD_BRCs data are not available in the virtual DAS")
                        error_flag = error_programming
                        return
                ELSE IF (.not.present(iAMD_file)) THEN
                        call logmsg(log_error,"aeolus_l2bp_import_one_AMD_BRC: ERROR!! "//&
                        "iAMD_file argument is missing")
                        error_flag = error_programming
                        return
                ELSE
                        IF (iAMD_file > size(DAS_AMD_BRCs,1)) THEN
                                call logmsg(log_error,"aeolus_l2bp_import_one_AMD_BRC: ERROR!!
                                    "//&
                                    "requested iAMD_file is beyond available range")
                                error_flag = error_programming
                                return
                        ELSE
                                IF (AMD_iBRC > size(DAS_AMD_BRCs,2)) THEN
                                        call logmsg(log_error,"aeolus_l2bp_import_one_AMD_BRC:
                                          ERROR!! "//& "requested AMD_iBRC is beyond available
                                          range")
```

```
                    error_flag = error_programming
                    return
                ELSE
                    AMD%Geoloc_ADS_off_nadir => DAS_AMD_BRCs
                        (iAMD_file,AMD_iBRC)%Geoloc_ADS_off_nadir
                    AMD%Geoloc_ADS_nadir => DAS_AMD_BRCs
                        (iAMD_file,AMD_iBRC)%Geoloc_ADS_nadir
                    AMD%Met_MDS_off_nadir => DAS_AMD_BRCs
                        (iAMD_file,AMD_iBRC)%Met_MDS_off_nadir
                    AMD%Met_MDS_nadir => DAS_AMD_BRCs
                        (iAMD_file,AMD_iBRC)%Met_MDS_nadir

                ENDIF
            ENDIF
        ENDIF
    ENDIF
ENDIF
```

4.9 aeolus_l2bp_load_data_to_vdas

Arguments:

Type	Intent	Name
TYPE(L2BP_Settings_type)	in	L2BP_Settings
integer	out	error_flag

Externals:

The module virtual_das is used. This module contains the following declarations:

```
TYPE(L1B_HDR_DataType), pointer, save :: DAS_L1B_Headers
TYPE(L1B_BRC_DataType), dimension(:), pointer, save :: DAS_L1B_BRCs
TYPE(AMD_HDR_DataType), dimension(:), pointer, save :: DAS_AMD_Headers
TYPE(AMD_BRC_DataType), dimension(:,:), pointer, save :: DAS_AMD_BRCs
logical, save :: virtual_das_initialised
```

Contents:

```
! read the L1B Headers and transfer to virtual DAS
allocate(DAS_L1B_Headers)
call aeolus_l2bp_import_L1B_Headers (DAS_L1B_Headers, DAS_Num_BRC, error_flag,
        L1B_filename= L2BP_Settings%L1Bfile%name)
```

```
! read the L1B BRCs and transfer to virtual DAS
allocate(DAS_L1B_BRCs(DAS_Num_BRC))
DO iBRC = 1, DAS_Num_BRC
        call aeolus_l2bp_import_one_L1B_BRC (DAS_L1B_Headers, DAS_L1B(iBRC), iBRC,
                error_flag, L1B_filename=L2BP_Settings%L1Bfile%name)

ENDDO

! read the AMD Headers and transfer to virtual DAS
allocate(DAS_AMD_Headers(L2BP_Settings%NumNWPfiles))
allocate(DAS_Num_AMD_BRC(L2BP_Settings%NumNWPfiles))
DO AMD_file = 1, L2BP_Settings%NumNWPfiles
        call aeolus_l2bp_import_AMD_Headers (DAS_AMD_Headers(AMD_file),
                DAS_Num_AMD_BRC(AMD_file), error_flag,
                AMD_filename=L2BP_Settings%NWPfile(AMD_file)%name)

ENDDO

! read the AMD BRCs and transfer to virtual DAS
allocate(DAS_AMD_BRCs(L2BP_Settings%NumNWPfiles, maxval(DAS_Num_AMD_BRC)))
DO AMD_file = 1, L2BP_Settings%NumNWPfiles
        DO AMD_iBRC = 1, DAS_Num_AMD_BRC(AMD_file)

                call aeolus_l2bp_import_one_AMD_BRC (DAS_AMD_Headers(AMD_file),
                        DAS_AMD(AMD_file,AMD_iBRC),AMD_iBRC, error_flag,
                        AMD_filename= L2BP_Settings%NWPfile(AMD_file)%name)

        ENDDO
ENDDO

virtual_das_initialised = .true.
```

4.10 aeolus_l2bp_prepare_primary

Arguments:

Type	Intent	Name
TYPE(L1B_BRC_DataType)	inout	L1B
TYPE(L2BC_BRC_DataType)	out	L2B
TYPE(AMD_BRC_DataType), dimension(:),		AMD

pointer		
integer	in	iBRC
TYPE(L2B_AuxPar_Type)	in	L2B_AuxPar
TYPE(WorkingDataType)	inout	WorkingData
integer	out	error_flag

Contents:

N_meas = size(L1B%Geoloc_ADS%Meas_AOCS)

call Init_L2BC_BRC_DataStructure (L2B,
N_meas, N_Obs_Max_Mie,N_Obs_Max_Rayleigh,
int(N_range_gates), error_flag, Init_Headers=.FALSE.)

call InitWorkingData (WorkingData, int(N_range_gates,IntAs), int(N_Meas,IntAs),
int(N_Obs_Max_Mie,IntAs),int(N_Obs_Max_Rayleigh,IntAs), error_flag)

call Copy_L1B_fields2WorkingArays (L1B,WorkingData)

call Screen_L1B_Data (L1B, WorkingData, L2B%PCD_ADS%L1B_Input_Screening,
L2B_AuxPar%Data%Screening_Params%L1B_Screening_Params, error_flag)

call determine_vcog (WorkingData, error_flag)

call Find_Matching_AMD_Profile(AMD, WorkingData,
L2B_AuxPar%Data%WVM_Params%AMD_Matchup_Params,
L2B%PCD_ADS%L2B_AMD_Screening%L2B_AMD_Collocations,
which_AMD_DSR, iBRC, error_flag)

IF (which_AMD_DSR >0) THEN

from_off_nadir_flag = .TRUE.

call Copy_AMD_to_WorkingData (from_off_nadir_flag, AMD(which_AMD_DSR),
L1B, WorkingData,error_flag)

ENDIF

4.11 L2BP_primary

Arguments:

Type	Intent	Name
TYPE(L1B_BRC_DataType)	inout	L1B
TYPE(L2BC_BRC_DataType)	inout	L2B
TYPE(WorkingDataType)	inout	WorkingData
TYPE(RBC_DataType)	inout	RBC
TYPE(L2B_AuxPar_Type)	in	L2B_Auxpar
integer	out	error_flag

Contents:

call Match_Mie_And_Rayleigh_Bins (WorkingData, error_flag)

IF (Extinction_Needed) call Calc_Extinction (L1B, WorkingData, MolExt_Method,
PartExt_Method, error_flag, use_lite_test_data,
calibr_on_upper_bin=.not.(use_lite_test_data))

call Calc_BackscatterRatio (L1B, WorkingData, error_flag, Method=
L2B_AuxPar%Data%WVM_Params%Optical_Properties_Params%ScatRatio_Method,
FallBackMethod=
L2B_AuxPar%Data%WVM_Params%Optical_Properties_Params%ScatRatio_Method2)

call Classify (WorkingData, L2B_AuxPar%Data%WVM_Params%Classification_Params,
error_flag)

call Select_And_Weigh_Rayleigh_Meas (WorkingData, L2B, error_flag)

call Select_And_Weigh_Mie_Meas (WorkingData, L2B, error_flag)

call Construct_Rayleigh_Obs (WorkingData, L2B, error_flag)

call Construct_Mie_Obs (WorkingData, L2B, error_flag)

call Process_Rayleigh_Observations (L1B, WorkingData, RBC, L2B, error_flag)

call Process_Mie_Observations (L1B, WorkingData, L2B_AuxPar, L2B, error_flag)

call Format_and_Copy_WD_to_L2B_Outp (WorkingData, L2B, error_flag)

L2B%PCD_ADS%N_Obs_Mie_Actual = L2B%Mie_MDS%N_Obs_Mie_Actual

L2B%PCD_ADS%N_Obs_Rayleigh_Actual = L2B%Rayleigh_MDS%N_Obs_Rayleigh_Actual

L2B%Geoloc_ADS%N_Obs_Mie_Actual = L2B%Mie_MDS%N_Obs_Mie_Actual

L2B%Geoloc_ADS%N_Obs_Rayleigh_Actual = L2B%Rayleigh_MDS%N_Obs_Rayleigh_Actual

call DeleteWorkingData (WorkingData)

4.12 aeolus_l2bp_export_one_L1B_BRC

Arguments:

Type	Intent	Name
TYPE(L1B_BRC_DataType)	inout	L1B
integer	out	error_flag

Externals:

The module virtual_das is used. See section 4.9for a description of that module.

Contents:

! → you can also export the L1B data for a given BRC to your application here

IF (virtual_das_initialised) THEN

nullify(L1B%Geoloc_ADS)

nullify(L1B%PCD_ADS)

nullify(L1B%GWD_ADS)

nullify(L1B%Meas_ADS)

nullify(L1B%Cal_ADS)

nullify(L1B%US_MDS)

nullify(L1B%WV_MDS)

ELSE

call Delete_L1B_BRC_DataStructure (L1B)

ENDIF

4.13 aeolus_l2bp_fill_L2B_Headers

Arguments:

Type	Intent	Name
TYPE(L1B_HDR_DataType)	in	L1B_Headers
TYPE(L2BC_BRC_DataType), dimension(:), pointer		L2B
TYPE(L2BC_HDR_DataType)	out	L2B_Headers
TYPE(L2BP_Settings_type)	in	L2BP_Settings
integer(IntAus)	in	Num_L2B_BRCs
type(DateTimeType)	in	DateTimeStart
type(DateTimeType)	in	DateTimeStop
integer(IntAl)	in	lat_start
integer(IntAl)	in	lat_stop
integer(IntAl)	in	lon_start
integer(IntAl)	in	lon_stop
integer(IntAuc)	out	M_Meas0
integer(IntAuc)	out	M_Mie0
integer(IntAuc)	out	M_Rayleigh0
integer	out	error_flag

Contents:

call Init_L2BC_HDR_DataStructure (L2B_Headers, error_flag)

call Fill_L2BC_FH (L2B_Headers, L2B_AuxPar%Data%FH_Default_Fields,

DateTimeStart, DateTimeStop, error_flag)

call Fill_L2B_MPH_From_L1B_MPH(L1B_Headers%MPH,
L2B_Headers%MPH, error_flag)

call Fill_Remaining_L2BC_MPH_Items (L2B_Headers%MPH, L2B_Headers%FH,
L2BP_Settings,

L2BP_Settings%L2B_AuxPar%Data%MPH_Default_Fields,
DateTimeStart, DateTimeStop,
error_flag)

call GetStatisticsL2BMieDS (L2B, int(Num_L2B_BRCs), M_Mie0,
Num_Valid_Mie_Profiles, Num_Clear_Mie_Profiles,
Num_Cloud_Mie_Profiles, Num_Mie_Prof_Warm_Pulses,
Num_Profiles_Surface_Mie, Num_Invalid_Meas_Prof_L2B_Mie,
Num_Invalid_Meas_Bins_L2B_Mie,Num_Valid_Obs_L2B_Mie,
error_flag)

call GetStatisticsL2BRayleighDS (L2B, int(Num_L2B_BRCs), M_Rayleigh0,
Num_Valid_Rayleigh_Profiles, Num_Clear_Rayleigh_Profiles,
Num_Cloud_Rayleigh_Profiles, Num_Rayleigh_Prof_Warm_Pulses,
Num_Profiles_Surface_Ray, Num_Invalid_Meas_Prof_L2B_Ray,
Num_Invalid_Meas_Bins_L2B_Ray,Num_Valid_Obs_L2B_Ray,
error_flag)

Num_Valid_Obs_Prof_L1B_Mie=0

Num_Valid_Obs_Prof_L1B_Ray=0

Num_Valid_Obs_Bins_L1B_Mie=0

Num_Valid_Obs_Bins_L1B_Ray=0

Num_Valid_Meas_Prof_L1B_Mie=0

Num_Valid_Meas_Prof_L1B_Ray=0

Num_Valid_Meas_Bins_L1B_Mie=0

Num_Valid_Meas_Bins_L1B_Ray=0

Num_Invalid_Meas_Prof_L1B_Mie=0

Num_Invalid_Meas_Prof_L1B_Ray=0

Num_Invalid_Meas_Bins_L1B_Mie=0

Num_Invalid_Meas_Bins_L1B_Ray=0

call Fill_L2BC_SPH (L2B_Headers, int(Num_L2B_BRCs), lat_start, lat_stop, lon_start,
lon_stop, Sat_Track, M_Mie0, M_Rayleigh0, M_Meas0,
Num_Valid_Mie_Profiles, Num_Valid_Rayleigh_Profiles,
Num_Clear_Mie_Profiles, Num_Clear_Rayleigh_Profiles,
Num_Cloud_Mie_Profiles, Num_Cloud_Rayleigh_Profiles,
Num_Mie_Prof_Warm_Pulses, Num_Rayleigh_Prof_Warm_Pulses,
Num_Profiles_Surface_Mie, Num_Profiles_Surface_Ray,
Num_Valid_Obs_Prof_L1B_Mie,Num_Valid_Obs_Prof_L1B_Ray,
Num_Valid_Meas_Prof_L1B_Mie, Num_Valid_Meas_Prof_L1B_Ray,
Num_Valid_Obs_Bins_L1B_Mie,Num_Valid_Obs_Bins_L1B_Ray,
Num_Valid_Meas_Bins_L1B_Mie,
Num_Valid_Meas_Bins_L1B_Ray,
Num_Invalid_Meas_Prof_L1B_Mie,
Num_Invalid_Meas_Prof_L1B_Ray,
Num_Invalid_Meas_Prof_L2B_Mie,
Num_Invalid_Meas_Prof_L2B_Ray,
Num_Invalid_Meas_Bins_L1B_Mie,
Num_Invalid_Meas_Bins_L1B_Ray,
Num_Invalid_Meas_Bins_L2B_Mie,
Num_Invalid_Meas_Bins_L2B_Ray,Num_Valid_Obs_L2B_Mie,
Num_Valid_Obs_L2B_Ray, error_flag, L2C_flag=.FALSE.)

4.14 aeolus_l2bp_export_L1B_Headers

Arguments:

Type	Intent	Name
TYPE(L1B_HDR_DataType)	inout	L1B_Headers
integer	out	error_flag

Externals:

The module virtual_das is used. See section 4.9for a description of that module.

Contents:

! → you can also export the L1B_Headers to your application here

```
IF (virtual_das_initialised) THEN
        ! the pointers inside L1B_Headers currently point to the virtual DAS
        ! so all we have to do is nullify them.
        ! The actual deallocation is done in subroutine aeolus_l2bp_clear_vdas
        nullify(L1B_Headers%FH)
        nullify(L1B_Headers%MPH)
        nullify(L1B_Headers%SPH)
ELSE
        call Delete_L1B_HDR_DataStructure (L1B_Headers)
ENDIF
```

4.15 aeolus_l2bp_export_L2B_Headers

Arguments:

Type	Intent	Name
TYPE(L2BC_HDR_DataType)	inout	L2B_Headers
integer	out	error_flag

Contents:

! → you can also export the L2B_Headers to your application here

```
call Delete_L2BC_HDR_DataStructure (L2B_Headers)
```

4.16 aeolus_l2bp_export_one_L2B_BRC

Arguments:

Type	Intent	Name

TYPE(L2BC_BRC_DataType)	inout	L2B
integer	out	error_flag

Contents:

! → you can also export the L2B data for a given BRC to your application here

call Delete_L2BC_BRC_DataStructure (L2B, error_flag)

4.17 aeolus_l2bp_export_AMD_Headers

Arguments:

Type	Intent	Name
TYPE(AMD_HDR_DataType)	inout	AMD_Headers
integer	out	error_flag

Externals:

The module virtual_das is used. See section 4.9for a description of that module.

Contents:

! → you can also export the AMD_Headers to your application here

```
IF (virtual_das_initialised) THEN
        ! the pointers inside AMD_Headers currently point to the virtual DAS
        ! so all we have to do is nullify them.
        ! The actual deallocation is done in subroutine aeolus_l2bp_clear_vdas
        nullify(AMD_Headers%FH)
        nullify(AMD_Headers%MPH)
        nullify(AMD_Headers%SPH)
ELSE
        call Delete_AMD_HDR_DataStructure (AMD_Headers)
ENDIF
```

4.18 aeolus_l2bp_export_one_AMD_BRC

Arguments:

Type	Intent	Name
TYPE(AMD_BRC_DataType)	inout	AMD
integer	out	error_flag

Externals:

The module virtual_das is used. See section 4.9for a description of that module.

Contents:

! → you can also export the AMD data for a given BRC to your application here

IF (virtual_das_initialised) THEN

! the pointers inside AMD currently point to the virtual DAS

! so all we have to do is nullify them.

! The actual deallocation is done in subroutine aeolus_l2bp_clear_vdas

nullify(AMD%Geoloc_ADS_off_nadir)

nullify(AMD%Geoloc_ADS_nadir)

nullify(AMD%Met_MDS_off_nadir)

nullify(AMD%Met_MDS_nadir)

ELSE

call Delete_AMD_BRC_DataStructure (AMD)

ENDIF

4.19 aeolus_l2bp_unsetup

Arguments:

Type	Intent	Name
TYPE(JobOrderData_type)	out	JobOrderData
TYPE(L2BP_Settings_type)	out	L2BP_Settings
TYPE(RBC_DataType)	inout	RBC

Contents:

call Close_Modules ()

IF (do_primary_processing) call Unload_Rayl_Brill_Tables (RBC)

call Delete_L2B_Proc_Settings (L2BP_Settings)

call Delete_JobOrderData (JobOrderData)

4.20 aeolus_l2bp_clear_vdas

Arguments:

Type	Intent	Name
integer	out	error_flag

Externals:

The module virtual_das is used. See section 4.9for a description of that module.

Contents:

virtual_das_initialised = .false.

```
IF (associated(DAS_L1B_Headers)) THEN
        call Delete_L1B_HDR_DataStructure(DAS_L1B_Headers)
        deallocate(DAS_L1B_Headers)
ENDIF
IF (associated(DAS_L1B_BRCs))THEN
        Num_BRCs = size(DAS_L1B_BRCs)
        DO iBRC=1,Num_BRCs
                call Delete_L1B_BRC_DataStructure (DAS_L1B_BRCs(iBRC))
        END DO
        deallocate(DAS_L1B_BRCs)
ENDIF
IF (associated(DAS_AMD_BRCs))THEN
        Num_AMD_files = size(DAS_AMD_BRCs(:,1))
        Num_AMD_BRCs  = size(DAS_AMD_BRCs(1,:))
        DO iAMD_file=1,Num_AMD_files
                DO iAMD_BRC=1,Num_AMD_BRCs
                        call Delete_AMD_BRC_DataStructure
                                (DAS_AMD_BRCs (iAMD_file, iAMD_BRC))
                END DO
        END DO
        deallocate(DAS_AMD_BRCs)
ENDIF

IF (associated(DAS_AMD_Headers)) THEN
        Num_AMD_files = size(DAS_AMD_Headers)
        DO iAMD_file=1,Num_AMD_files
                call Delete_AMD_HDR_DataStructure(DAS_AMD_Headers(iAMD_file))
        END DO
        deallocate(DAS_AMD_Headers)
ENDIF
```