SOFTWARE USER MANUAL

SMOS Artificial Scene Library

Art-Sce-Lib

Code:	ASL-SUM-DME
Issue:	1.2
Approval Date:	14/10/2020

Prepared by: Daniel Barros / DME Software Engineer

Reviewed by: Gonçalo Lopes / DME Software Engineer

Approved by: José Barbosa / Project Manager

Approval Signature:

This page intentionally left blank

Document Status Log

Issue	Section	Change Description	Date
1.0	All document	First version of the document.	04/02/2019
1.1	All document	• Removed RHEL 5 from list of supported operating systems • Removed procgsl library from the list of compilation dependencies • Added gomp library to the list of compilation dependencies • Added new enumerator to define the ASL polarizations • Updated the UI description of the compute_artificial_scene function • Updated the examples according to the new UI of the compute_artificial_scene	12/07/2019
1.2	All document	• Updated in section 4.1 the Artificial Scene Computation interface and description • Updated in subsections 4.2.1 and 4.2.2 the code examples according to the interface update in section 4.1	14/10/2020

Table of Contents

1. INTRODUCTION 6
1.1. Purpose and Scope of the Document 6
1.2. Applicability 6
1.3. Document Overview 6
1.4. Acronyms and Abbreviations 6
2. RELATED DOCUMENTS 8
2.1. Applicable Documents 8
2.2. Reference Documents 8
3. DIRECTORY STRUCTURE AND SYSTEM INSTALLATION 9
3.1. Software and Hardware Pre-Requisites 9
3.1. Installation Instructions 10
3.2. Delivery Package Structure 10
4. PRODUCT DESCRIPTION 12
4.1. Functions List 12
4.2. Library's Usage 14
4.2.1. Example code 1 – How to read a single snapshot 15
4.2.2. Example code 2 – How to read a L1b product 18

List of Tables

Table 1: Applicable documents 8
Table 2: Reference documents 8
Table 3: Supported Operating Systems 9
Table 4: Software dependencies 9

1. INTRODUCTION

1.1. Purpose and Scope of the Document

This document is the User Manual of the Artificial Scene Library (ASL), a software library produced in the frame of the IDEAS_SMOS project. This library provides the necessary functions to cope with the changes in the Level 1B of the SMOS L1 Operational Processor (L1OP).

Since L1OP v7.1.0, the Gibbs 2 algorithm, implemented based on the description in [RD.01], is supported. This algorithm subtracts an estimated base image to the measurements, with the Sea modelled by a fixed Fresnel model and a fixed Land temperature, to decrease the overall radiometric bias and smooth the land-sea transitions bias. Therefore the L1b products are missing this component (which is added back in L1c processing).

The ASL library functions allow the users to use directly the L1b products by mimicking the Gibbs-2 image reconstruction correction algorithm as it is implemented in the L1OP. It allows to produce the same estimated scene as seen by the instrument so that it can be added directly to L1b data.

The purpose of this document is to provide all users of the SMOS Artificial Scene Library a user manual to learn its functions and capabilities.

1.2. Applicability

This document is intended for use by knowledgeable L1B users of the SMOS Artificial Scene Library software. Its scope is limited to the functionality and operations that can be done with the SMOS Artificial Scene Library.

1.3. Document Overview

The sections included in this manual are as follows:

- Section 3. Details the pre-requisites and how to link the library to user's code
- Section 4. Describes the libraries functionalities and commands

1.4. Acronyms and Abbreviations

The acronyms and abbreviations used in this document are the following ones:

AD: Applicable Document
ADF: Auxiliary Data File
API: Application Programming Interface
ASL: Artificial Scene Library
CFI: Customer Furnished Item
COTS: Commercial Off-The-Shelf
DME: DEIMOS Engenharia
DMS: DEIMOS Space
DPGS: Data Processing Ground Segment
ESA: European Space Agency
GSL: General Software Library
HW: Hardware

I/F:	Interface
ICD:	Interface Control Document
I/O:	Input/Output
L1OP:	SMOS Level 1 Operational Processor
L1PP:	SMOS Level 1 Prototype Processor
NIR:	Noise Injection Radiometers
NRT:	Near Real Time
O/S:	Operating System
PDPC:	Payload Data Processing Centre
PDR:	Preliminary Design Review
PFW:	Processing Framework
RD:	Reference Document
RPF:	Reference Processing Facility
DSR:	Data Set Record
DTA:	Data Transfer Agent
FAT:	Final Acceptance Tests
SDV:	SMOS Data Viewer
SCoT:	SMOS Comparison Tool
SMOS:	Soil Moisture and Ocean Salinity
SPRT:	Software Problem Report Test
SPR:	Software Problem Report
ST:	System Test
SVVP:	Software Verification & Validation Plan
SW:	Software
SUM:	Software User Manual
TBC:	To Be Confirmed
TC:	Test Case
TDS:	Test Data Set
UT:	Unit Test
VTS:	Verification/Validation Test Specification
VTP:	Verification/Validation Test Procedure
VTR:	Verification/Validation Test Report
V&V:	Verification & Validation
VVP:	Verification & Validation Plan
VVR:	Verification & Validation Report

2. RELATED DOCUMENTS

2.1. Applicable Documents

The following table specifies the documents applicable during project development.

Table 1: Applicable documents

Reference	Code	Title	Issue	Date
[AD 01]	SO-DS-DME-L1PP-0008	SMOS L1 Processor L1a to L1b Data Processing Model	2.26	17/04/2020
[AD 02]	SO-TN-IDR-GS-0005	SMOS Level 1 and Auxiliary Data Products Specifications	6.11	03/04/2020

2.2. Reference Documents

The following table specifies the reference documents that shall be considered during project development.

Table 2: Reference documents

Reference	Code	Title	Version	Date
[RD 01]	SO-TN-CBAP-GS-053.1	Implementation of the GIBBS 2 correction algorithm in the SMOS L1a to L1b data processor: Definition and assessment of its stability	1.e	RD.25

3. DIRECTORY STRUCTURE AND SYSTEM INSTALLATION

3.1. Software and Hardware Pre-Requisites

The library was tested in the three 64 bits platforms described below. The main verification and validation procedures, however, where done in the machine where L1OP is installed, Earth4 (CentOS 7) server, with the following specifications:

- DELL PowerEdge R720XD with the following specifications:
  - CPU: Intel® Xeon® Processor E5-2643 (10M Cache, 3.30 GHz, 8.00 GT/s Intel® QPI)
  - RAM: 64GB RDIMMs, 1600 MHz

The ASL is delivered as a shared and static library programmed in C++ language and has been successfully compiled and tested with the following versions of the operating system and third party software. One of these compatible versions must be present in the platform where the ASL is installed for its correct usage and operation:

Table 3: Supported Operating Systems

Name	Description	System Kernel and Standard Libraries
Community Enterprise Operating System 6 (CentOS 6)	Operating System	• Kernel 2.6.32-754.10.1.el6.x86_64 • glibc.x86_64 2.12-1.212.el6 • gcc-c++.x86_64 4.4.7-23.el6 • cmake.x86_64 2.8.12.2-4.el6
Community Enterprise Operating System 7 (CentOS 7)	Operating System	• Kernel 3.10.0-327.el7.x86_64 • glibc_x86_64 2.17-196.el7 • gcc-c++.x86_64 4.8.5-16.el7_4.2 • cmake.x86_64 2.8.12.2-2.el7

If installing ASL v1.0.0 or higher, the following libraries are required to be set as external dependencies in the user's building system. This means that the ASL requires these libraries to be present during the user's compilation and linkage procedure. There is no need for the user to install them separately because they are already delivered within the ASL installation package.

Table 4: Software dependencies

Name	Description	Location
General Software Library v2.0	A library that provides general logging functions used by the DPGS subsystems within the PDPC Core. These messages are part of the interface of the processors such as L1OP.	Delivered inside ASL package
Xerces-C++ v2.7.0	Validation XML parser written in a portable subset of C++. Provides the ability to read and write XML data.	Delivered inside ASL package
FFTW3	A C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST)	Delivered inside ASL package
XMLRWAPI	The XML Read/Write API is a library to access SMOS products in Earth Explorer format. It is mainly used by the developers of the SMOS components but it is designed and presented as a general public library.	Delivered inside ASL package
CMake	CMake is an open-source, cross-platform family of tools designed to build, test and package software.	System

3.1. Installation Instructions

To install the SMOS Artificial Scene library it is only necessary to unzip the file to the target directory, "ASL-V.V.V-linux64.elX.tar.gz", where V.V.V corresponds to the library's version and X to the OS version where the package was compiled and tested.

3.2. Delivery Package Structure

After decompressing the ASL in the target directory the following folder's structure should be created:

```
|-- SM_OPER_AUX_DGG____20050101T000000_20500101T000000_300_003_3
|-- SM_OPER_AUX_LSMASK_20050101T000000_20500101T000000_300_003_3
|-- SM_TEST_AUX_FRSNEL_20050101T000000_20500101T000000_720_001_0
|-- SM_TEST_MIR_SC_F1B_20110502T025659_20110502T025704_730_001_1
|-- examples
|-- include
|-- l1b_freqs.txt
|-- libart_scene.a
|-- libart_scene_dyn.so
|-- libfftw3.a
|-- libinfo_unzip.a
|-- libinfo_zip.a
|-- libprocgsl.so
|-- libxerces-c.so -> libxerces-c.so.27.0
|-- libxerces-c.so.27 -> libxerces-c.so.27.0
|-- libxerces-c.so.27.0
|-- libxerces-depdom.so -> libxerces-depdom.so.27.0
|-- libxerces-depdom.so.27 -> libxerces-depdom.so.27.0
|-- libxerces-depdom.so.27.0
|-- libxrwa.a
|-- smos
`-- xml_rw_api.usr_conf.xml
```

As previously referred, the ASL is delivered in two compilation modes:

- *libart_scene.a – ASL statically compiled.*
- *libart_scene_dyn.so – ASL dynamically compiled.*

All the dependencies needed by the library for the supported OS listed in section 3.1, are also delivered:

- *libfftw3.a*
- *libinfo_unzip.a*
- *libinfo_zip.a*
- *libprocgsl.so*
- *libxerces-c.so -> libxerces-c.so.27.0*
- *libxerces-c.so.27 -> libxerces-c.so.27.0*
- *libxerces-c.so.27.0*

- *libxerces-depdom.so -> libxerces-depdom.so.27.0*
- *libxerces-depdom.so.27 -> libxerces-depdom.so.27.0*
- *libxerces-depdom.so.27.0*
- *libxrwa.a*

The package also contains the library's header files with the functions prototypes inside the "include" folder. Furthermore, two examples are provided to show the library's usage. They are intended for the user to understand its building procedures and coding functionalities:

```
examples
|-- CMakeLists.txt
|-- Utilities.cmake
|-- build-examples.sh
|-- ut01
|   |-- CMakeLists.txt
|   |-- makefile
|   |-- target.ut01.mk
|   `-- ut01.cc
`-- ut02
    |-- CMakeLists.txt
    |-- L1BProduct.cc
    |-- L1BProduct.h
```

Three ADF, a MIR_SC_F1B product and an ASCII file are also provided. They are ingested and used by the code in the examples provided:

- l1b_freqs.txt
- SM_OPER_AUX_DGG____20050101T000000_20500101T000000_300_003_3
- SM_OPER_AUX_LSMASK_20050101T000000_20500101T000000_300_003_3
- SM_TEST_AUX_FRSNEL_20050101T000000_20500101T000000_720_001_0
- SM_TEST_MIR_SC_F1B_20110502T025659_20110502T025704_730_001_1

The XML Read/Write API library is used to read the DGG, LSMASK and FRSNEL auxiliary data files. This library needs the product's schemas to be placed in the directory where the XML_RW_API_HOME environment variable is pointing to. This library also needs the *xml_rw_api.usr_conf.xml* file to be present in the place where the user's program is going to executed. Both are also provided in the ASL package:

- smos – folder's structure containing the XML schemas
- xml_rw_api.usr_conf.xml – Pure XML file needed by the XML R/W library

4. PRODUCT DESCRIPTION

4.1. Functions List

The three main functions implemented by the library are:

Preprocessing - ingests the Auxiliary Data and pre-computes optimized data structures

- **Syntax:** asl::Artificial_scene::preprocessing(const string &file_dgg, const string &file_lsmask, const string &file_frsnel)

Artificial Scene Computation - provides the Artificial Scene for 1 polarization or for the all the 4 polarizations, in the extended star domain, for a given position of the spacecraft and polarization of the scene. For correct usage, the user needs to define the following important parameters: the scene polarization as well as the output polarization, a lock so that the user can call this function under a multithreading safe environment and finally a flag to control the usage of ***fftw_cleanup().***

- **Syntax:** asl:: Artificial_scene::compute_artificial_scene(const double matrix_BFP2EF[3][3], const double position[3], const EPolarization pol_scene, omp_lock_t* const lock, const EPolarization pol_output, t_asl_output* const sum_output_data, **const bool fftw_cleanup_on**)

Important Note !!

- The function argument ***fftw_cleanup_on*** must always be set, either *true* or *false*
- ***fftw_cleanup_on*** is meant to enable or disable the call of the fftw_cleanup(). This function resets the FFTW structures to a pristine state.
- If this flag is set as ***true***, all FFTW persistent information in memory is cleared
- If this flag is set as ***false***, all FFTW persistent information continues to persist in memory
- Typically, in cases where no other FFTW instances are set, is recommended to set the flag as true.
- **However**, in cases where other FFTW instances are set in memory and the ASL function is called with this flag ON, the program might end with a **catastrophic** and unexpected behaviour. In such cases the user should set the flag as **false.**

Artificial Scene Addition – adds the Artificial Scene to the L1b measurements by expanding them to the extended star domain

- **Syntax:** asl::Artificial_scene::sum_artificial_scenes_to_measurements(const t_nrt_doublecomplex* const art_sce_freqs const double* const L1b_freqs, t_nrt_doublecomplex* const fc_freqs)

The interfaces of the three main functions of the Artificial Scene Library are available in the header file of the library "Artificial_scene.h" in the /include folder of the installation directory. As can be seen in the interface specifications, the name space of the library is *asl*.

The type of polarization enumeration will be called *EPolarization*, to obtain an artificial scene a user must select the scene polarization as an input which must be *H_POL* or *V_POL*, and as an output the user may select one polarization: *H_POL*, *V_POL* or *F_POL* or can select all polarizations with the option *ALL_POL*, as seen bellow:

```
typedef enum EPolarization
{
   H_POL,
```

```
    V_POL,
    F_POL,
    ALL_POL,

    EPOLARIZATION_SIZE
} EPolarization;
```

The output of the function that computes the artificial scene for a given spacecraft position will be a structure called *t_asl_output* containing the artificial scene temperatures frequencies for the 4 polarizations as well as the water fraction for each L1OP's convention xi-eta pixel (196x196 hex grid), non-earth pixels are printed with a default value of -9999, as seen bellow:

```
typedef struct t_asl_output
{
   t_nrt_doublecomplex artificial_scene_freqs_H[NON_RED_BASELINES_256*2-1];
   t_nrt_doublecomplex artificial_scene_freqs_V[NON_RED_BASELINES_256*2-1];
   t_nrt_doublecomplex artificial_scene_freqs_F[NON_RED_BASELINES_256*2-1];

   std::pair<double,double> xi_eta_list[N_GMAT_NEW_SQR]; // 196x196
   double water_fraction_list[N_GMAT_NEW_SQR]; //196x196
} t_asl_output;
```

It can been observed that the struct contains five arrays with fixed sizes. The artificial scene frequencies have 10873 values for each polarization. The output format of the addition will be a structure which will contain the final L1b frequencies, ready to be apodized and geolocated:

```
typedef struct { double r, i; } t_nrt_doublecomplex;
```

The NON_RED_BASELINES_256 and N_GMAT_NEW_SQR are system constants, the first corresponds to the number of non-redundant baselines for a star with 4 times more frequencies. It corresponds to half of that star (5437 values) while the second is the square value of the size of xi eta list size with value of 196x196 (38416).

Two more functions are also provided:

(U,V) Star Grid – provides the (U,V) grid associated to the L1b measurements star domain

- **Syntax:** const std::vector< std::pair<double, double> >& asl::Artificial_scene::getUVStarGrid() const

(U,V) Denser Star Grid – provides the (U,V) grid associated to the extended star domain

- **Syntax:** const std::vector< std::pair<double, double> >& asl::Artificial_scene::getUVDenserStarGrid() const

To incorporate the library into an existing code it is necessary to add to the user's makefile the path to the ASL include director, e.g., "/path/to/ASL/installation/dir/include" the list of the COTS and name of the flavor of the library to incorporate, art_scene (static) and art_scene_dyn (dynamic), as well as the path to the files: "/path/to/ASL/installation/dir".

4.2. Library's Usage

In this section two examples will be given to illustrate how to compute the addition of the Artificial Scene frequencies to the frequencies retrieved by L1B's products.

First, the user must decide whether to link his program statically or dynamically according to his necessities. If he chooses to link statically, he must take into account that his program will be bigger and harder to upgrade, but probably easier to deploy. To use this option, he has to explicitly add the *art_scene.a* to the list of linkage libraries. If it links dynamically, his program will be smaller, easier to upgrade, but harder to deploy. In this case it has to explicitly add the *art_scene_dyn.so* to the linker's libraries list. Either way, one of the variants must be explicitly included in the user's building system to enable the usage of the libraries capabilities.

List of libraries that **must** be included in the user's building system:

- art_scene/ art_scene_dyn
- fftw3
- xrwa
- info_zip
- info_unzip
- xerces-c
- pthread
- gomp

Because of the XML R/W API dependency API (also provided by the ASL), the user **must** be aware of two **important** notes:

1. The environment variable XML_RW_API_HOME is **mandatory** to be set. It should point to the place where the XML schemas are deployed. The ASL already provides them in the "smos" folder. This library expects the schemas to be inside a tree like the one provided in "smos" folder.
2. The *xml_rw_api.usr_conf.xml* file **must** be placed in the directory where the user's program is going to executed.

Before using the examples given by the ASL package the user needs to compile both test units. And for that, it is **mandatory** the user to have the CMake building system installed in his machine. The CMake is a standard and cross-platform building system so a simple "sudo *yum install cmake*" suffices to have the system properly installed and ready to be used in a LINUX machine. Please see section 3.1 for more details. **No other environment variable or setup is needed to build and execute the examples.**

Then, the user needs to enter the following directory: "/path/to/ASL/installation/dir /examples" and type the command: *./build-examples.sh*.

In the directory "/path/to/ASL/installation/dir" (just one folder above) two executables, *example01* and *example02*, will be created.

After running each test an output folder will be created with all the generated product, *ut01-output* and *ut02-output* respectively.

4.2.1. Example code 1 – How to read a single snapshot

In this section a full example will be given to illustrate how to compute the addition of the L1b frequencies to the artificial scene.

```
// System
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <stdexcept>
#include <sys/stat.h>
#include <cerrno>
#include <omp.h>

// Local: ASL library
#include "Artificial_scene.h"

// Number of non-redundant baselines
#define N_STAR_BT_FREQS_HALF  1396

// Number of BT frequencies to build an entire star (2791 values)
#define N_STAR_BT_FREQS  (2*N_STAR_BT_FREQS_HALF-1)

// Number of entries, real + imaginary, to build an entire (U,V) star (2 x 2791 values)
#define N_STAR_BT_FREQS_DOUBLED (2*N_STAR_BT_FREQS)

// Number of non-redundant baselines for a star 4 times more dense
#define NON_RED_BASELINES_4_TIMES_DENSE  5437

/**
 * fprintfADouble_2: Print double to file with a specified separation character
 */
int fprintfADouble_2( FILE*         pFile,
                      const double  theDouble,
                      const char*   separation )
{
   int n = 0;

   if ( !isnan( theDouble ) )
   {
      fprintf( pFile,
               "%.15f%s ",
               theDouble,
               separation );
   }
   else
   {
      n++;
      fprintf( pFile,
               "%e%s",
               0.0,
               separation );
   }

   return n;

}


////////////////////////////////////////////////////
/// String Paths Querying
////////////////////////////////////////////////////


// Breaks spath in its sub-directory elements.
//
inline std::vector< std::string > ListDirPaths( const std::string &spath )
{
    static const std::string separator = "/\\";

    std::vector< std::string > v;
    std::string::size_type npos = 0;

    while ( (npos = spath.find_first_of( separator, npos ) ) != std::string::npos )
        v.push_back( spath.substr( 0, ++npos ) );

    return v;
}

////////////////////////////////////////////////////
/// String Paths Manipulation
////////////////////////////////////////////////////

// Turns a path into a directory path, if necessary.
//
```

```
// Appends the final "/" to a path, if it does not exist, to
// be distinct from a filename and to allow direct concatenation
// of a filename.
// Returns
//      sdir, if sdir has a final "/"
//      or
//      sdir += "/"
//
inline std::string& CorrectDirPath( std::string& sdir )
{
    if (*sdir.rbegin() != '\\' && *sdir.rbegin() != '/')
        sdir += "/";
    return sdir;
}


// Builds a directory path from a given path, if necessary.
// See CorrectDirPath
//
inline std::string MakeDirPathFrom( const std::string& sdir )
{
    std::string tmp_sdir( sdir) ;
    return CorrectDirPath( tmp_sdir );
}

//////////////////////////////////////////////////
/// Disk (physical) Paths Manipulation
//////////////////////////////////////////////////
inline bool MakeSubDirectory( const std::string& sdir, bool bMustNotExist = false )
{
    return  ( mkdir( MakeDirPathFrom( sdir ).c_str(), S_IRWXU | S_IRGRP | S_IROTH) == 0 )
            ||
#if defined (__APPLE__)
            ( !bMustNotExist && ( errno == EEXIST || errno == EISDIR ) );
#else
          ( !bMustNotExist && errno == EEXIST );
#endif
}


// Creates sdir on disk ensuring all its sub-directories are also
// explicitly created
//
inline bool MakeDirectory( const std::string &sdir )
{
    std::vector< std::string > v = ListDirPaths( MakeDirPathFrom( sdir) );
    size_t size = v.size();
    for ( size_t i_subdir = 0; i_subdir < size; ++i_subdir )
        if ( !MakeSubDirectory(v[i_subdir]) )
            return false;

    return true;
}


/**
 * MAIN: Load L1B half-orbit and use the ASL to retrieve the final
 * BT frequencies ready to be apodized and geolocated
 */
int main()
{

try
{
    // Check for XML_RW_API_HOME environment variable
    if (!getenv("XML_RW_API_HOME"))
        setenv( "XML_RW_API_HOME", ".", 1 );

    // Declaration of the necessary variables
    std::string file_DGG    = "SM_OPER_AUX_DGG____20050101T000000_20500101T000000_300_003_3";
    std::string file_LSMASK = "SM_OPER_AUX_LSMASK_20050101T000000_20500101T000000_300_003_3";
    std::string file_FRSNEL = "SM_TEST_AUX_FRSNEL_20050101T000000_20500101T000000_720_001_0";

    // For further ascii file writing
    bool iscreated = MakeDirectory("ut01-output");
    if(!iscreated)
        throw std::runtime_error("Error making ut01-output folder in current directory");

    // Set rotation matrix from BFP to Earth Frame
    double matrix_BFP2EF[3][3] = {{-0.6778658246148082, -0.7349128728081019, -0.020024814611593916},
                                  {0.7122958797950691, -0.6497732772772872, -0.26538513101397},
                                  {0.1820233596169856, -0.19415910361679206, 0.9639345097238035}};

    // Set spacecraft position (X, Y, Z)
    double position[3] = {-2943939.0306312982, -4097999.6158383028, 5042026.9284043992};

    // Declare BT frequencies array for a single snapshot
    double L1b_freqs_in[N_STAR_BT_FREQS];
```

```
    // Construct the artificial scene objects needed to execute the ASL functions
    asl::Artificial_scene        art_scene;
    asl::t_asl_output            cas_output;
    asl::t_nrt_error error       = asl::nrtpNoError();
    // Declare sum_artificial_scenes_to_measurements output array
    asl::t_nrt_doublecomplex  asl_final_freqs[NON_RED_BASELINES_4_TIMES_DENSE*2-1];

    // Decare and initialize omp lock for multithreading
    omp_lock_t lock;
    omp_init_lock(&lock);

    // Open file where the L1b freqs are stored
    std::ifstream myfile ("l1b_freqs.txt" , ios::in);
    // Load L1B data
    if (myfile.is_open())
    {
        size_t mm = 0;
        while (myfile >> L1b_freqs_in[mm])
            ++mm;
    }
    else
        throw runtime_error("Unable to open file: l1b_freqs.txt");

    // Close
    myfile.close();

   // Convert input data to a single array containing real + imaginary parts in a contiguous way
   double L1b_freqs_conv[N_STAR_BT_FREQS_DOUBLED];

   // Get zero-baseline
   L1b_freqs_conv[0]                = L1b_freqs_in[0];
   L1b_freqs_conv[N_STAR_BT_FREQS]   = 0.;

   for(size_t i_freq = 1; i_freq < static_cast<size_t>(N_STAR_BT_FREQS_HALF); ++i_freq)
   {
       // Build Array with Real Star
       L1b_freqs_conv[i_freq]                            = L1b_freqs_in[i_freq];
       L1b_freqs_conv[i_freq + N_STAR_BT_FREQS_HALF - 1]  = L1b_freqs_in[i_freq];
       // Build Array with Imaginary Star
       L1b_freqs_conv[i_freq + N_STAR_BT_FREQS] = L1b_freqs_in[i_freq + N_STAR_BT_FREQS_HALF - 1];
       L1b_freqs_conv[i_freq + N_STAR_BT_FREQS_HALF - 1 + N_STAR_BT_FREQS]  = -L1b_freqs_in[i_freq + N_STAR_BT_FREQS_HALF -
1];
   }

    // Polarization input and output of ASL
    asl::EPolarization pol_input = asl::H_POL;
    asl::EPolarization pol_ouput = asl::H_POL;

    // Call of the ASL proprocessing function
    error = art_scene.preprocessing(file_DGG, file_LSMASK, file_FRSNEL);
    if( error.error_code != asl::NRT_ERR_CODE_NO_ERROR )
    {
        throw runtime_error("Error in ASL preprocessing function");
    }

    // Call of the computation of the artifcial scenes computation
    error = art_scene.compute_artificial_scene(matrix_BFP2EF,
                                               position,
                                               pol_input,
                                               &lock,
                                               pol_ouput,
                                               &cas_output,
                                               true);

    if( error.error_code != asl::NRT_ERR_CODE_NO_ERROR )
    {
        throw std::runtime_error("Error in ASL compute_artificial_scene function");
    }

    // Call of the sum of the L1B frequencies with the frequencies given by the artificial scene
    error = art_scene.sum_artificial_scenes_to_measurements(&cas_output.artificial_scene_freqs_H[0],
                                                            &L1b_freqs_conv[0],
                                                            asl_final_freqs);
    if( error.error_code != asl::NRT_ERR_CODE_NO_ERROR )
    {
        throw std::runtime_error("Error in ASL sum_artificial_scenes_to_measurements function");
    }

    // Write output ASCII file
    FILE* pFile = NULL;

    // Define breakpoint filename

    // Open file
    pFile = fopen( "ut01-output/l1b_fs_artificial_library_extTempFreqs_000078572242_256_H.txt", "w" );
    if ( pFile == NULL )
    {
```

```
        throw                std::runtime_error("Error                opening                file:                ut01-
output/l1b_fs_artificial_library_extTempFreqs_000078572242_256_H.txt");
    }

    // Get (U,V) denser star grid
    std::vector< std::pair<double, double> > u_v_denser = art_scene.getUVDenserStarGrid();

    // Save Breakpoint
    int nr_nans = 0;
    for ( size_t kk = 0; kk < 2 * NON_RED_BASELINES_4_TIMES_DENSE - 1; ++kk )
    {
        nr_nans += fprintfADouble_2( pFile, u_v_denser[ kk ].first, " " );
        nr_nans += fprintfADouble_2( pFile, u_v_denser[ kk ].second, " " );
        nr_nans += fprintfADouble_2( pFile, asl_final_freqs[ kk ].r, " " );
        nr_nans += fprintfADouble_2( pFile, asl_final_freqs[ kk ].i, "i\n" );
    }

    // // Get (U,V) star grid
    // std::vector< std::pair<double, double> > u_v = art_scene.getUVStarGrid();
    // for ( size_t kk = 0; kk < N_STAR_BT_FREQS; ++kk )
    //     std::cout << u_v[ kk ].first << " " << u_v[ kk ].second << std::endl;

    // Close
    fclose(pFile);

    // Destroy lock
    omp_destroy_lock(&lock);

    return 0;
}
catch( const std::out_of_range &e )
{
    std::cerr << e.what() << '\n';
    return 1;
}
catch( const std::runtime_error &e )
{
    std::cerr << e.what() << '\n';
    return 2;
}
catch( const std::exception &e )
{
    std::cerr << e.what() << '\n';
    return 3;
}
catch( ...  )
{
    std::cerr << "Unexpected exception type, the original error is unknown" << '\n';
    return 4;
}


} // end MAIN()
```

4.2.2. Example code 2 – How to read a L1b product

```
// System
#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>
#include <vector>
#include <stdexcept>
#include <sys/stat.h>
#include <cerrno>
#include <omp.h>

// Local: ASL library
#include "Artificial_scene.h"
#include "L1BProduct.h"

// Number of non-redundant baselines
#define N_STAR_BT_FREQS_HALF  1396

// Number of BT frequencies to build an entire star (2791 values)
#define N_STAR_BT_FREQS  (2*N_STAR_BT_FREQS_HALF-1)

// Number of entries, real + imaginary, to build an entire (U,V) star (2 x 2791 values)
#define N_STAR_BT_FREQS_DOUBLED (2*N_STAR_BT_FREQS)

// Number of non-redundant baselines for a star 4 times more dense
#define NON_RED_BASELINES_4_TIMES_DENSE  5437

/**
 * fprintfADouble_2: Print double to file with a specified separation character
```

```
 */
int fprintfADouble_2( FILE*         pFile,
                      const double  theDouble,
                      const char*   separation )
{
   int n = 0;

   if ( !isnan( theDouble ) )
   {
      fprintf( pFile,
               "%.15f%s ",
               theDouble,
               separation );
   }
   else
   {
      n++;
      fprintf( pFile,
               "%e%s",
               0.0,
               separation );
   }

   return n;

}

//////////////////////////////////////////////////////
/// String Paths Querying
//////////////////////////////////////////////////////


// Breaks spath in its sub-directory elements.
//
inline std::vector< std::string > ListDirPaths( const std::string &spath )
{
    static const std::string separator = "/\\";

    std::vector< std::string > v;
    std::string::size_type npos = 0;

    while ( (npos = spath.find_first_of( separator, npos ) ) != std::string::npos )
        v.push_back( spath.substr( 0, ++npos ) );

    return v;
}

//////////////////////////////////////////////////////
/// String Paths Manipulation
//////////////////////////////////////////////////////

// Turns a path into a directory path, if necessary.
//
// Appends the final "/" to a path, if it does not exist, to
// be distinct from a filename and to allow direct concatenation
// of a filename.
// Returns
//      sdir, if sdir has a final "/"
//      or
//      sdir += "/"
//
inline std::string& CorrectDirPath( std::string& sdir )
{
    if (*sdir.rbegin() != '\\' && *sdir.rbegin() != '/')
        sdir += "/";
    return sdir;
}


// Builds a directory path from a given path, if necessary.
// See CorrectDirPath
//
inline std::string MakeDirPathFrom( const std::string& sdir )
{
    std::string tmp_sdir( sdir) ;
    return CorrectDirPath( tmp_sdir );
}

//////////////////////////////////////////////////////
/// Disk (physical) Paths Manipulation
//////////////////////////////////////////////////////
inline bool MakeSubDirectory( const std::string& sdir, bool bMustNotExist = false )
{
    return  ( mkdir( MakeDirPathFrom( sdir ).c_str(), S_IRWXU | S_IRGRP | S_IROTH) == 0 )
            ||
#if defined (__APPLE__)
            ( !bMustNotExist && ( errno == EEXIST || errno == EISDIR ) );
#else
```

```
          ( !bMustNotExist && errno == EEXIST );
#endif
}


// Creates sdir on disk ensuring all its sub-directories are also
// explicitly created
//
inline bool MakeDirectory( const std::string &sdir )
{
    std::vector< std::string > v = ListDirPaths( MakeDirPathFrom( sdir) );
    size_t size = v.size();
    for ( size_t i_subdir = 0; i_subdir < size; ++i_subdir )
        if ( !MakeSubDirectory(v[i_subdir]) )
            return false;

    return true;
}



/**
 * MAIN: Load L1B half-orbit and use the ASL to retrieve the final
 * BT frequencies ready to be apodized and geolocated
 */
int main()
{

try
{
    // Check for XML_RW_API_HOME environment variable
    if (!getenv("XML_RW_API_HOME"))
        setenv( "XML_RW_API_HOME", ".", 1 );

    // Declaration of the necessary variables
    std::string file_DGG    = "SM_OPER_AUX_DGG___20050101T000000_20500101T000000_300_003_3";
    std::string file_LSMASK = "SM_OPER_AUX_LSMASK_20050101T000000_20500101T000000_300_003_3";
    std::string file_FRSNEL = "SM_TEST_AUX_FRSNEL_20050101T000000_20500101T000000_720_001_0";
    // Set L1B filename to be used
    std::string file_L1B("SM_TEST_MIR_SC_F1B_20110502T025659_20110502T025704_730_001_1");

    // For further ascii file writing
    bool iscreated = MakeDirectory("ut02-output");
    if(!iscreated)
        throw std::runtime_error("Error making ut02-output folder in current directory");

    // Construct the artificial scene objects needed to execute the ASL functions
    asl::Artificial_scene    art_scene;
    asl::t_nrt_error         error = asl::nrtpNoError();

    // Decare and initialize omp lock for multithreading
    omp_lock_t lock;
    omp_init_lock(&lock);

    // Load L1B Product data and generate instance
    L1BProduct l1b_science_data(file_L1B);

    // 1. Call of the ASL preprocessing function
    error = art_scene.preprocessing(file_DGG, file_LSMASK, file_FRSNEL);
    if( error.error_code != asl::NRT_ERR_CODE_NO_ERROR )
        throw runtime_error("Error in ASL preprocessing function");

    // Get the number of scenes
    size_t n_scenes = l1b_science_data.GetNumberOfScenes();

    // To report progress
    int counter = 0;

    // Declare double array of 2*2791 frequencies. First 2791 are the real part
    // while the 2792 onwards are the imaginary part
    double scene_bt_freqs[N_STAR_BT_FREQS_DOUBLED];

    // Loop through the scenes
    for(size_t i_scene = 0; i_scene < n_scenes; ++i_scene)
    {
        // Get L1B scene data
        ut::t_ut_l1b_temp_snapshot scene = l1b_science_data.GetSceneData(i_scene);

        // Declare compute artificial scene function output struct
        asl::t_asl_output        as_freqs_output;
        asl::t_nrt_doublecomplex *artificial_scene_freqs;

        // Pol. string
        std::string pol;

        // Polarization input ASL
        asl::EPolarization pol_input;
        asl::EPolarization pol_ouput = asl::ALL_POL;
```

```
// Check whether the scene is pure/mixed or cross-polar and build the entirely (U,V) star
// Convert L1B product BT frequencies into possible librarie's user type, a double array
// It will be the library's argument to sum the Artificial Scene frequencies back to measurements
if ((scene.flags & 0x03) == ut::L1B_FLAG_POL_HH)
{
    for(size_t i_pixel = 0; i_pixel < static_cast<size_t>(N_STAR_BT_FREQS); ++i_pixel)
    {
        scene_bt_freqs[i_pixel] = scene.scene_freqs[i_pixel].r;
        scene_bt_freqs[i_pixel + static_cast<size_t>(N_STAR_BT_FREQS)] = scene.scene_freqs[i_pixel].i;
    }

    // Point to the further ASL scene frequencies
    artificial_scene_freqs = &as_freqs_output.artificial_scene_freqs_H[0];
    // Set pol.
    pol = 'H';
    pol_input = asl::H_POL;

    // std::cout << "L1B_FLAG_POL_HH" << std::endl;
}
else if ((scene.flags & 0x03) == ut::L1B_FLAG_POL_VV)
{
    for(size_t i_pixel = 0; i_pixel < static_cast<size_t>(N_STAR_BT_FREQS); ++i_pixel)
    {
        scene_bt_freqs[i_pixel] = scene.scene_freqs[i_pixel].r;
        scene_bt_freqs[i_pixel + static_cast<size_t>(N_STAR_BT_FREQS)] = scene.scene_freqs[i_pixel].i;
    }

    // Point to the further ASL scene frequencies
    artificial_scene_freqs = &as_freqs_output.artificial_scene_freqs_V[0];
    // Set pol.
    pol = 'V';
    pol_input = asl::V_POL;

    // std::cout << "L1B_FLAG_POL_VV" << std::endl;
}
else if((scene.flags & 0x03) == ut::L1B_FLAG_POL_HV_REAL)
{
    for(size_t i_pixel = 0; i_pixel < static_cast<size_t>(N_STAR_BT_FREQS); ++i_pixel)
    {
        scene_bt_freqs[i_pixel] = scene.scene_freqs[i_pixel].r;
    }
    // std::cout << "L1B_FLAG_POL_HV_REAL" << std::endl;
    continue; // Because the imaginary part was not yet correctly fullfiled
}
else if((scene.flags & 0x03) == ut::L1B_FLAG_POL_HV_IMAG)
{
    for(size_t i_pixel = 0; i_pixel < static_cast<size_t>(N_STAR_BT_FREQS); ++i_pixel)
    {
        scene_bt_freqs[i_pixel + static_cast<size_t>(N_STAR_BT_FREQS)] = scene.scene_freqs[i_pixel].i;
    }
    // Point to the further ASL scene frequencies
    artificial_scene_freqs = &as_freqs_output.artificial_scene_freqs_F[0];
    // Set pol.
    pol = 'F';

    // std::cout << "L1B_FLAG_POL_HV_IMAG" << std::endl;
}

// Build rotation matrix
double matrix_BFP2EF[3][3];
for(size_t i_row = 0; i_row < 3; ++i_row)
{
    for(size_t i_col = 0; i_col < 3; ++i_col)
    {
        matrix_BFP2EF[i_row][i_col] = scene.matrix_BFP2EF[i_row * 3 + i_col];
    }
} //or memcpy(&matrix_BFP2EF[0][0], &scene.matrix_BFP2EF[0], 9 * sizeof(double));


// 2. Call of the computation of the artifcial scenes computation
error = art_scene.compute_artificial_scene(matrix_BFP2EF,
                                           scene.
                                           position,
                                           pol_input,
                                           &lock,
                                           pol_ouput,
                                           &as_freqs_output,
                                           true);
if( error.error_code != asl::NRT_ERR_CODE_NO_ERROR )
    throw std::runtime_error("Error in ASL compute_artificial_scene function");

// Declare sum_artificial_scenes_to_measurements output array
asl::t_nrt_doublecomplex  asl_final_freqs[NON_RED_BASELINES_4_TIMES_DENSE*2-1];

// 3. Call of the sum of the L1B frequencies with the frequencies given by the artificial scene
error = art_scene.sum_artificial_scenes_to_measurements(artificial_scene_freqs,
                                                        &scene_bt_freqs[0],
```

```
                                                                &asl_final_freqs[0]);
        if( error.error_code != asl::NRT_ERR_CODE_NO_ERROR )
            throw std::runtime_error("Error in ASL sum_artificial_scenes_to_measurements function");

        ////////////////////////
        // Write output ASCII files
        ////////////////////////
        char c_id[64];
        sprintf(c_id,"%d", scene.snapshot_id); //converts to decimal base.
        std::string s_id(c_id);

        // Define ASCII filename
        std::string filename = "ut02-output/l1b_fs_artificial_library_extTempFreqs_0000" + s_id + "_256_" + pol + ".txt";

        // Open file
        FILE* pFile = NULL;
        pFile = fopen(filename.c_str() , "w" );
        if ( pFile == NULL )
            throw                std::runtime_error("Error                opening                file:                ut02-
output/l1b_fs_artificial_library_extTempFreqs_000078572242_256_H.txt");

        // Save Breakpoint
        int nr_nans = 0;
        for ( size_t kk = 0; kk < 2 * NON_RED_BASELINES_4_TIMES_DENSE - 1; ++kk )
        {
            nr_nans += fprintfADouble_2( pFile, asl_final_freqs[ kk ].r, " " );
            nr_nans += fprintfADouble_2( pFile, asl_final_freqs[ kk ].i, "i\n" );
        }
        // Close
        fclose(pFile);


            /* report progress in percentage for L1OP */
           if ( n_scenes > 10   &&
                counter % (n_scenes/10) == 0)
           {
               double partial_process = static_cast<double>(counter + 1 / n_scenes);
               std::cout << "\r" << "Progress: " << static_cast<int>(partial_process*10)  << "%" << std::flush;
           }
           counter++;
    } // end scenes loop

    // Carriage Return
    std::cout << "\n";

    // Destroy lock
    omp_destroy_lock(&lock);

    return 0;
}
catch( const std::out_of_range &e )
{
    std::cerr << e.what() << '\n';
    return 1;
}
catch( const std::runtime_error &e )
{
    std::cerr << e.what() << '\n';
    return 2;
}
catch( const std::exception &e )
{
    std::cerr << e.what() << '\n';
    return 3;
}
catch( ...  )
{
    std::cerr << "Unexpected exception type, the original error is unknown" << '\n';
    return 4;
}


} // end MAIN()
```

End of Document