Norsk Regnesentral
NORWEGIAN COMPUTING CENTER

Note

SnowLab

A System for Automated Snow Product Generation

Software documentation

Note no SAMBA/56/09

Authors Jostein Amlien
Hans Koren
Line Eikvil
Rune Solberg

Date 15 Dec 2009

Norsk Regnesentral

Norsk Regnesentral (Norwegian Computing Center, NR) is a private, independent, non-profit foundation established in 1952. NR carries out contract research and development projects in the areas of information and communication technology and applied statistical modeling. The clients are a broad range of industrial, commercial and public service organizations in the national as well as the international market. Our scientific and technical capabilities are further developed in co-operation with The Research Council of Norway and key customers. The results of our projects may take the form of reports, software, prototypes, and short courses. A proof of the confidence and appreciation our clients have for us is given by the fact that most of our new contracts are signed with previous customers.

Norsk Regnesentral
Norwegian Computing Center
Postboks 114, Blindern
NO-0314 Oslo, Norway

Besøksadresse
Office address
Gaustadalléen 23
NO-0373 Oslo, Norway

Telefon · telephone
(+47) 22 85 25 00
Telefaks · telefax
(+47) 22 69 76 60

Internett · internet
www.nr.no
E-post · e-mail
nr@nr.no

Title	**SnowLab - Software documentation**
Authors	**Jostein Amlien** **Hans Koren** **Line Eikvil** **Rune Solberg**
Date	15 Dec 2009
Year	2009
Publication number	SAMBA/56/09

Abstract

This note is a documentation of the SnowLab software system for the automatic generation of snow related geophysical products from satellite data of moderate resolution. The system is designed for the daily generation of Snow Covered Area (SCA), Snow Temperature Surface (STS), Snow Grain Size (SGS), and Snow Surface Wetness (SSW). In addition, the cloud cover is generated for masking purposes. Currently MODIS and AATSR are supported, but some products are not available for AATSR.

The generated products can be categorized as basic products or as derived products. The basic products are generated from satellite data from one day or one pass only. The derived products are generated from the basic products for several days. The derived products part also includes multi-sensor functionality. The basic products are SCA, STS and SGS. The derived products are SSW and an improved SCA.

The software system is designed as an automatic production chain, implemented in IDL/ENVI, and runs on a Linux platform. It takes care of the complete process from the downloading of satellite data until the final products have been generated.

Keywords	Snow products, geophysical parameters, SCA, STS, SGS, SSW
Target group	Snow hydrologists, hydropower companies
Availability	Open
Project number	
Research field	Earth observation
Number of pages	46

Contents

1 System overview

1.1 Purpose

This report describes and documents a software system for the automated retrieval of various snow products from remote sensing data. The basic idea behind the software system is to consider the retrieval process as a production chain, consisting of certain steps, as in this generic production chain:

- Data download
- Data import and pre-processing
- Basic products generation
- Geo-correction
- Derived products generation
- Product export

The production chain is controlled by a production chain framework, which retrieves data and leads them through the various steps in the production chain.

The actual steps in the chain are defined by the application software. Although the current application is the generation of snow products, the application software could in principle address virtually any production purpose. The application software for the generation of snow products is organized as modules that are closely related to the steps in the production chain.

This document describes the production chain and its individual steps. It documents the various software modules, and how they interact with other modules and with the framework.

1.2 System architecture

This section introduces the main modules in the system, the main process of the production chain, and the main principles for the framework that controls the dataflow through the system.

1.2.1 Modules

As shown in Fig. 1 the application software is grouped into five main modules. Each module provides at least one function that interacts with the framework through a standardized interface.

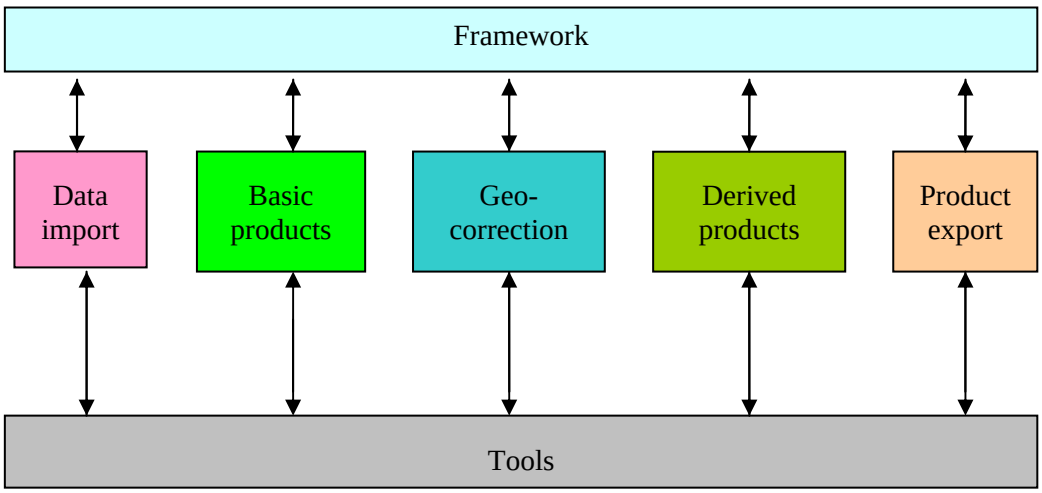


***Figure 1** : The main modules*

- *Data import module*: Provides functions for converting downloaded satellite image data to the ENVI format, which is the internal format in the software system. The result of the import may be as swath (as for MODIS) or grid (as for AATSR). In any case, each location have been observed only once.
- *Basic products module*: Provides snow parameter retrieval functions that derive snow parameters directly from the imported dataset.
- *Geo-correction module*: Provides functions for the projection of the basic products to a grid reference. This module is redundant for data imported as grid data.
- *Derived products module*: Provides functions that combine a set of basic products into a derived snow product. This dataset in typically a multi-temporal one. This module is optional.
- *Product export module*: Converts the products into user specified format. This may also include inclusion of mask data
- *Toolbox*: Provides basic tool functions to the other modules.

1.2.2 Main process

The main process for the retrieval of cryospheric products is shown in Fig 2. The arrows show the sequence of the various functional steps. The colours in the diagram refer to the main modules above. When derived products are not requested, the process proceeds directly to the export step.

For the AATSR process note that the import undertakes the geometrical projection into a map grid. This is done by the BEAM software, which is external to the system. There is therefore no need for the projection step for AATSR data.

For the MODIS process note that the projection step may also be undertaken before the basic product generation. However, this is not required since the system have means to link auxiliary data to swath data.

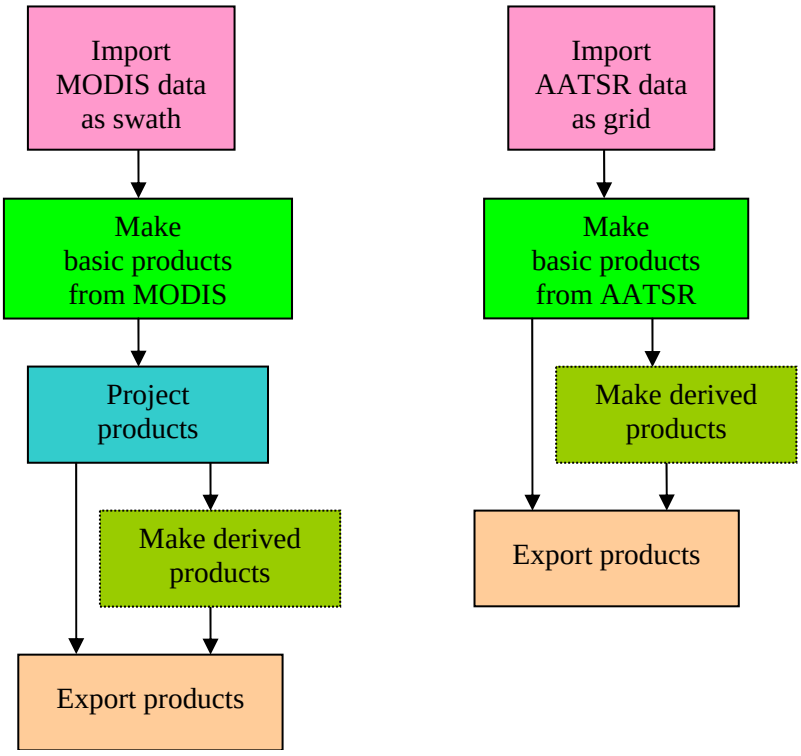


Figure 2. *The main process*

1.2.3 The framework

The framework is intended for automatic processing of remote sensing datasets through a production chain. A simple controller controls the production chain, where one dataset is processed at the time, running the process through the necessary steps.

The main principle behind the interaction between the application software and the framework is that the framework does not need to know the application software. The framework can thus be applied for any application. This principle is obtained by:

- Defining a standardized API (Application Program Interface)
- Saving all processing results in files between each step
- Transferring application specific arguments through text files

No functions performing operations on the data are part of the framework, but should all be given in the application specific modules. Any application can be plugged into the framework, if it is possible to call its functions from the framework's method API, which is described more detailed in section 3.2.

The framework takes care of retrieving the input data from some source and making them available to the application software. It controls the sequence the application functions are being called, and the dataflow through the system.

The framework is written in IDL and C++, and is intended to run on a Linux platform. Fig. 3 gives an overview of the framework.

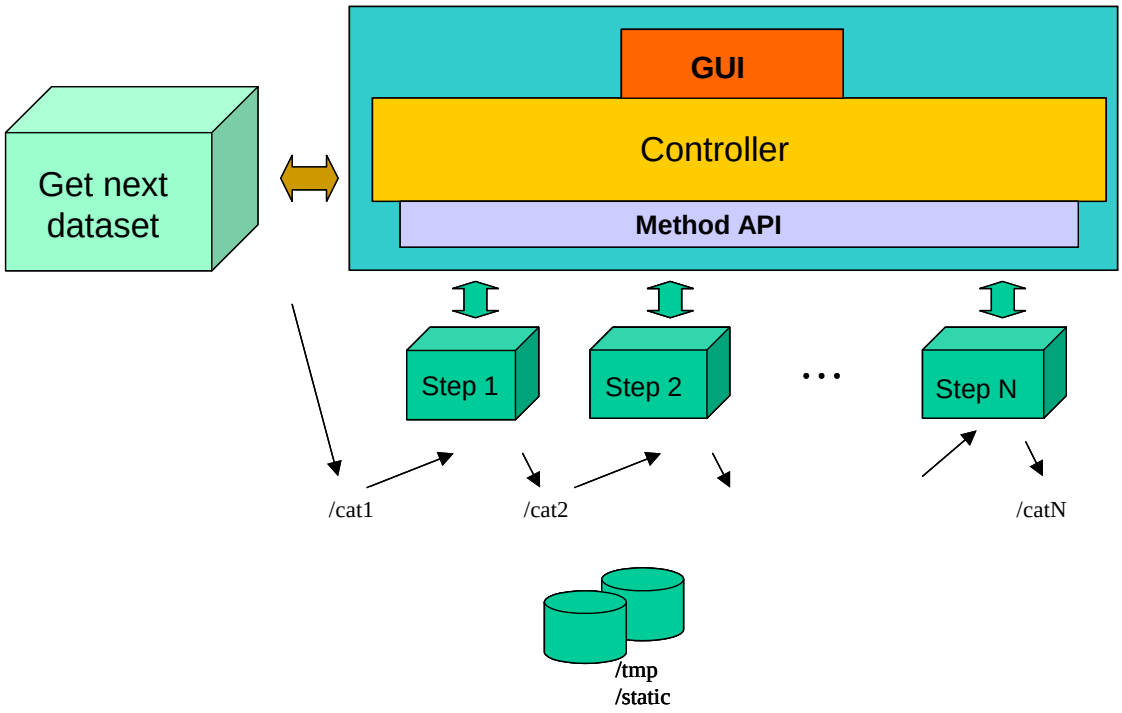


Figure 3. *The framework*

1.3 Overview of the system modules and their main functions

The system consists of the framework, the production modules, and a toolbox of common functions. This section gives an overview of the main functions in the production modules. These functions correspond to the main steps described above, and are callable from the framework, following the requirements outlined in section 1.2.3. Also some functions at a lower level follow these requirements.

1.3.1 Import module

The import module reads data from satellite images and stores them in a set ENVI files comprising a dataset. There is one import function for each satellite sensor:

- Import MODIS data
- Import AATSR data

The MODIS import results in a swath dataset that is a continuous portion of a satellite orbit. The swath may be composed on adjacent scenes. The AATSR import function is a wrapping function around the BEAM function 'mosaic'. All satellite passes during on single day is projected into a grid.

1.3.2 Basic products module

This module controls the production of basic products, i.e. products that can be retrieved from one dataset. The module has function on different levels. On the top level we have one generic function *retrieve_basicproducts,* which is called from the sensor specific functions. These functions can be specified in the production chain setup, and they are:

- MODIS_basicproducts
- AATSR_basicproducts

The basic products to retrieve are specified in the corresponding configuration file. Each basic product can be retrieved from at least sensor by one or more methods. There is one sub-module for each of the basic products:

- Cloud Mask module (both sensors, two methods)
- Snow Covered Area (SCA) module(both sensors, several methods)
- Surface Temperature of Snow (STS) module (optional; Modis only; one method only)
- Snow Grain Size (SGS) module (optional; Modis only; one method only)

The methods for creating a specific product can be called form either the sensor specific top level function, but it is also possible to call most of them from the production chain setup via specific application functions specific to product and sensor.

1.3.3 Geometric correction module

The geometric module undertakes the geometrical projection of swath data into a map grid. The data to be resampled is typically a set of basic products, but may also be original swath data. The module is called by the function projectProducts.

Note that he geometrical correction module is not needed for AATSR data in grid format. These data are projected and mosaiced under the import by calling the external BEAM software.

1.3.4 Derived products module

This module handles products where one needs to consider multiple basic products, e.g. in a time-series, or derived from multiple sensors. A central part of this module is a time-series / multi-product controller, which all main function relies on. These functions are:

- SCA multi-scene / multi-sensor / time-series combination

- Snow surface wetness (SSW)

1.3.5 Export module

The purpose of the export module is to export the product to a format tailored to the end-user of the product. This may involve combination of basic products with masks, inclusion of meta data and colour tables, and saving files in specified formats.

1.4 Outline of the production chains

The production chains are controlled by the framework its setup file. The mains steps correspond to the main functions in the modules.

The product chains for MODIS and AATSR data are somewhat different. For AATSR data only cloud mask generation and SCA is available, but with the choice of different methods. For times series 'derived' products, only multi-scene and multi-sensor SCA are available for AATSR.

1.4.1 Outline of the MODIS production chain

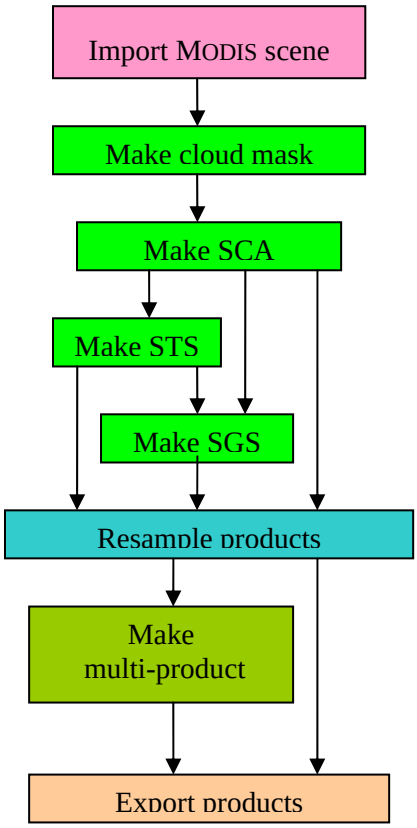


***Figure 4a**. Outline of the MODIS production chain*

The production chain is controlled by the framework. The mains steps mainly correspond to the main functions in the modules.

- Import data
  Import MODIS data
- Basic products generation
  a. Make cloud mask
  b. Make SCA – Snow Covered Area

c. Make STS – Snow Temperature Surface (optional)
d. Make SGS – Snow Grain Size (optional)

- Resample to geometric reference
- Time-series 'derived ' products (optional)
  a. Make SSW (Snow Surface Wetness) : based on STS and recent change in SGS
  b. Make multi-scene SCA: estimate current SCA using a time-series of basic SCA
  c. Make multi-sensor SCA: estimate current SCA using multiple sensors
- Export products

1.4.2 Outline of the AATSR production chain

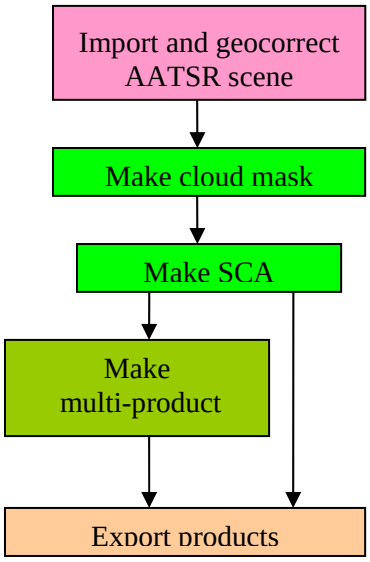


Figure 4b. *Outline of the AATSR production chain*

- Import data
  Import AATSR data into a geocorrected grid, using of BEAM software
- Basic products generation
  a. Make cloud mask
  b. Make SCA – Snow Covered Area
- Time-series 'derived ' products (optional)
  a. Make multi-scene SCA: estimate current SCA using a time-series of basic SCA
  b. Make multi-sensor SCA: estimate current SCA using multiple sensors
- Export products

1.5 Retrieval algorithms for basic snow parameters

The basic snow parameters that can be produced are Cloud Cover mask, Snow Covered Area (SCA), Surface Temperature Snow (STS), Snow Grain Size(SGS), and Snow Surface Wetness (SSW).

1.5.1 Cloud Cover Mask

A particular problem for practical use of the snow algorithms has been cloud detection. NR has experimented with several approaches, and the current best cloud detection algorithm is based on K Nearest Neighbour (kNN) classification of MODIS or AATSR data. In a kNN classifier a

pixel, represented by a vector of band values, is assigned the label, which is most prevalent among the K nearest labelled vectors from a reference set. A kNN classifier is an asymptotically optimum classifier as the size of the reference set increases. This algorithm has been described in a more detail in the Envisnow report D1-WP3, Part 1 (Solberg et al., 2005). Reference sets ('codebooks') have been defined for MODIS as well as for AATSR. For AATSR all 7 nadir bands are used, and for MODIS also 7 bands are used.

For the GlobSnow project, the SYKE cloud algorithm has also been included in the production chain for AATSR data. This algorithm uses three thermal bands and three reflectance bands. The normalized vegetation and snow indices (NDVI and NDSI) are calculated and used with various thresholds to detect clouds (see below).

```
Tdiff = T11 - T3.7
ndvi = (R0.87 – R0.55) / (R0.87 + R0.55)
ndsi = (R0.55 – R1.6 ) / (R0.55 + R1.6)
thres1 = -9K
thres2 = -7K

Cloud IF
Tdiff<Thres1 & R0.55>10% & ndvi<0.5
OR
Tdiff<Thres2 & (ndsi*100%/R0.55)< 1.115 & ndvi<0.5 & 0.0<ndsi< 0.8 & T12<275K
```

1.5.2 Snow Covered Area (SCA)

For MODIS data the NRL algorithm is used. For AATSR data NRL, Shi and SYKE algorithms are available.

1.5.2.1 NLR

The NLR algorithm for the retrieval of SCA stands for the Norwegian Linear Reflectance-to-Snow-Cover algorithm. The algorithm is based on an empirical reflectance-to-snow-cover model originally proposed for NOAA AVHRR by Andersen (1982) and later refined by Solberg and Andersen (1994). The algorithm was later tailored to MODIS data by NR.

The NLR algorithm retrieves the snow-cover fraction for each pixel. The model is calibrated by providing two points of a linear function relating observed radiance to fractional snow-cover

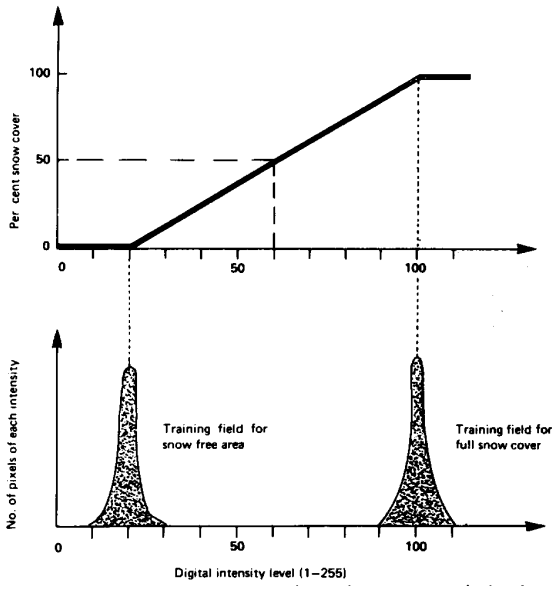


Figure 5 *The Norwegian Linear Reflectance-to-Snow-Cover (NLR) algorithm illustrated. A pixel value is linearly transformed to a snow cover percentage for that pixel. The algorithms are based on the assumption that the bare-ground reflectance is constant. (Andersen 1982)*

area (see Figure 5). The calibration is usually done automatically by means of calibration areas. Statistics from the calibration areas is then used to compute calibration points for the linear relationship. The statistics need to pass a series of tests, where the thresholds have been learned empirically. This algorithm and its validation has been reported in more detail from the project Envisnow, see D1-WP3, Part 1 (Solberg et al. 2005).

The algorithm was recently adapted to AATSR data by NR. Note that the various empiric parameter settings refer to radiance values, and therefore the AATSR radiance had to be converted to radiance values to pass the tests.

As an option the radiance values may be topographically corrected. The effects of illumination variations caused by the topography are modelled and then used to correct the data. The effect of the solar elevation is also taken into account in this correction. In order to preserve the validity of the statistical tests, the topographical corrected data should be in the same range as a typical radiance image for Norway, i.e. it is required to specify a moderate standard solar elevation in the algorithm, instead of a zenith position. In order to take the diffuse illumination into account, and avoid over-correction of dark slopes, a C-correction is applied with a default C=0.1.

When cloud-free calibration areas are totally missing, the calibration value from the preceding days will be used. Alternatively, each missing area will be simulated by looking at the history for that calibration area and derive the value from the last 5 observations.

1.5.2.2 Modified Shi

The Shi algorithm (Shi 2000) has been modified into a simpler version by ENVEO IT GmbH in Innsbruck, Austria. It has been included in the SnowLab production chain for AATSR data during the GlobSnow project. The method is based on the same principals as NRL, but the calibration is done in a different manner. Certain thresholds for NDSI and the reflectance in the 0.67 µm band are used to classify 100% snow covered pixels, mixed snow pixels and 100% snow free pixels. For each mixed pixel the snow fraction (SCA) is estimated based on linear spectral unmixing between the pure classes 100% snow covered and 100% snow free. The unmixing process estimates the end-menber values from the 5 spatially closest pixels for each of the pure classes. For areas within the forest, a binary snow or no snow detection is done. Since the algorithm needs to identify the 5 closest neighbours the processing time will vary from image to image. Topographic correction is optional.

1.5.2.3 SCAmod

The SCAmod (SYKE) algorithm is developed at the Finnish Environment Institute (Metsämäki et al 2005). This algorithm use constant reflectance values for snow free ground, forest canopy and wet snow (e.g. 7, 6, and 66). In addition a transmissivity map is required. This map specifies the fraction of the reflected radiation that will transmit the vegetation layer and reach the satellite sensor observing forest areas. This means that the transmissivity is an expression of the effect of forest on local reflectance observations. SCA can then be derived from observed reflectance based on the reflectance constants and the transmissivity values. Topographic correction is optional, but the effect of the solar elevation must be corrected. The coefficients are given under the assumption that the sun is in zenith.

1.5.3 Surface Temperature Snow (STS)

The retrieval of STS is based on Key's algorithm (Key 1997), which has been calibrated for various AVHRR sensors as well as for Modis, see http://stratus.ssec.wisc.edu/products/surftemp/. The application of Key's algorithm for the retrieval of snow surface temperatures in Norwegian mountains was recommended by Amlien and Solberg (2004). The standard method for

correcting the atmospheric attenuation is the split-window technique. In Key's algorithm the atmospheric path-length is also taken into account by introducing the view angle as a parameter.

1.5.4 Snow Grain Size (SGS)

The idea behind the SGS parameters is that the spectral signature of snow depends on the grain size. The spectral signature curves show a clear drop in reflectance when moving towards the longer wavelengths. The effect is more prominent for larger snow grains. The method has been described and evaluated by Koren et al. (2004). The algorithm compares the reflectance in MODIS bands 2 and 7 and calculates a grain size index, defined as SGS = (M2-M7)/(M2+M7).

1.6 Derived product algorithms

The derived snow parameters that are produced are SnowCoveredArea multi-product (multi-SCA), and SnowSurfaceWetness (SSW). Common for the derived products are that they are derived from a set of basic products.

1.6.1 Multi-SCA

Due to cloud cover problems there are large difficulties in obtaining an updated product every day. In order to overcome these problems, the SCA value must be predicted by means of recent results from the same sensor (multi-temporal), or from a different one (multi-sensor).

Each observation will be given a confidence value, which is declining with the time. As long as the confidence value is above a threshold, the observation is considered as useful. When new data arrives, the SCA value will be updated where the confidence for the new observations are better than the old ones. The confidence depends on the cloud cover and the view angle, in addition to the time since the acquisition.

The multi-SCA algorithm is described by Solberg et al. (2004a, 2004b, 2005b), and Malnes et al. (2005). It has also been described in a more detail in the Envisnow report D1-WP3, Part 1 (Solberg et al. 2005).

1.6.2 Snow Surface Wetness (SSW)

The main idea is to compare the temporal development in the SGS with the current value of STS. Where the temperature is close to 0 °C and the grain size is increasing, it is likely that the snow is becoming or has become wet.

The temperature observations give a good indication of where wet snow may be present, but are in themselves not accurate enough to provide very strong evidence of wet snow. However, a strong indication of a wet snow surface is a rapid increase of the effective grain size observed together with a snow surface temperature near 0°C. The algorithm can be expressed in a simplified version as

$$
\begin{aligned}
&\text{if } (SGS_{\text{today}} - SGS_{\text{recently}} > \text{SGS}_{\text{snowmelt-tresh}}) \\
&\qquad \text{and } (STS_{\text{low}} < STS_{\text{today}} < STS_{\text{high}}) \text{ then} \\
&\quad SSW = \text{WET-SNOW} \\
&\text{else} \\
&\text{if } SGS_{\text{today}} < \text{SGS}_{\text{bare-ground-tresh}} \text{ then} \\
&\quad SSW = \text{SNOW-FREE} \\
&\text{else} \\
&\text{if } STS_{\text{today}} > STS_{\text{high}} \text{ then}
\end{aligned}
$$

SSW = SNOW-FREE

else

SSW = DRY-SNOW

Note that more temperature classes are used in the implemented algorithm. Also a threshold of the SCA product is applied in order to mask out snow-free areas. The algorithm will also identify partly bare ground from temperature observations above 0°C and a rapid developing negative gradient for SGS (both due to appearance of snow-free ground patches at the sub-pixel level).

1.6.3 Other

Other derived snow parameters can also be estimated. The snow distribution pattern of a local area, like a drainage area, can be retrieved from the SCA product and a snow distribution model for that local area. The snow distribution model is an empirical one, based on classifications of the snow cover in a series of high-resolution images, like Landsat. The series must be representative for the development of the snow cover during a typical melting season. The model is represented by a likelihood function that gives the sequence the pixels will become snow-free.

The algorithm for the snow distribution pattern goes like this: Retrieve the SCA for the local area from the SCA product. Then produce a corresponding SCA from the empirical model by applying a threshold of the likelihood function. The value of the threshold should be the one that generates a snow mask that corresponds to the retrieved SCA value. This mask is the estimated snow distribution pattern.

2 System Operator's Manual

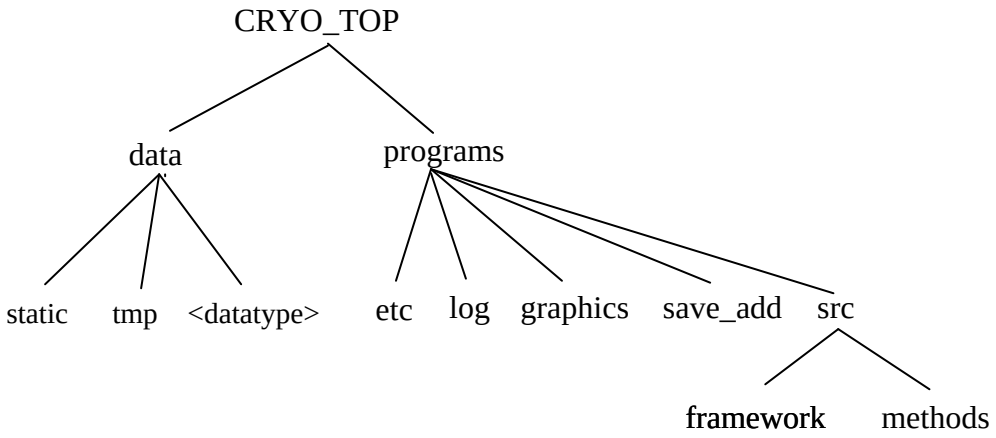
2.1 System Installation Guide

2.1.1 Unpacking the software

The software of the framework is contained in the tar-file cryo.tar. To install the framework, do the following:

- Copy the tar-file to the directory where you want to put the software.
- Unpack the tar-file:
  `> tar –xvf cryo.tar`
- Go to the sub directory cryo/programs:
  `> cd cryo/programs`
- Run the setup script cryo_setup from this directory:
  `> source cryo_setup`
  The environment variable CRYO_TOP will be set to the top directory where the software was unpacked, and IDL/ENVI variables and configurations will be set up.

2.1.2 Directory structure



The contents of these directories will be as follows:

data	default directory for list files containing specifications of local datasets. Root of data directory tree.
static	configuration files, calibration data, codebook for cloud classification etc.
tmp	temporary data.
<datatype>	intermediate and final results from the processing of a given data type. The directory has the same name as the processed data type. It contains a number of sub-directories, out*, each of them with a subdirectory for each processed dataset.
programs	setup and the main configuration file. Root for software directory tree.
etc	ftp-configuration files for data providers.

log	logfiles.
graphics	logos etc.
save_add	*.sav files (compiled IDL code).
src	root directory for source code
framework	source code for the framework, if provided
methods	root directory for application source code, if provided

The data directory will initially be empty, except that *static/* will contain various files needed for the application.

As the datasets are processed, sub-directories holding the intermediate results from each step of the processing chain will be created under the data directory, and given names based on the data type, the current step in the process and the original name of the dataset:

```
<datatype>/out*/<dataset>
```

In a process with n steps, the final results will be stored in out<n>, while the intermediate results are stored in the subdirectories out1 to out<n-1>.

2.1.3 Setup of the production chain

Although the production chain is started from a GUI available in the ENVI menu, its behaviour is controlled by predefined configuration files. When running the system the user has no possibilities to control the behaviour of the system.

The behaviour of the system can be changed by manipulation of these configuration files. This section presents the main principles for the configuration files and gives an overview of how they are used in the current system.

NOTE that the system operator should avoid changing the configuration files.

2.1.3.1 Configuration file format

The general format of the configuration file is simple. The configuration file is a text file that contains keyword–value pairs, where the keyword is separated from the value with an '=' sign: <keyword> = <value>. The value may be one single item or a list of items. When the value consist of a list of items, the list is enclosed in {} and each item separated by comma, like this: <keyword> = { <item1>, <item2>,…, <itemN> }

2.1.3.2 Main configuration file

The main configuration file is usually named `current.cfg` and always located under the `programs/` directory (see above description of the directory structure). This file will be read once as the program is started.

The purpose of the main configuration file is to define:

- the parameters needed by the automatic downloading routine of the framework
- one or more production chains
- a few optional parameters

Configuration of automatic download

The automatic download part is controlled by parameters defining one or more data providers, as well as a local mail user receiving messages about data ready to be downloaded. Below the addresses of NR has been used as an example. They should be changed to the correct addresses for mail host and ftp host.

The specification of a local mail user is required for receiving messages that will trigger the automatic remote download of data:

mailusr	user name
mailpwd	password
mailhost	mail host, e.g. mail.nr.no
mailport	local port e.g. 110

The various data providers are identified by:

nof providers	number of data providers
providers	list of data providers

<provider> *sender*: email-address that each of the providers will use to send the message

For each provider there is also a separate configuration file, <provider>.cfg, located in `programs/etc/`. The ftp-download from that provider is specified by these parameters:

host	name of ftp host, e.g. ftp.nr.no
user	user name, e.g. anonymous
pass	password, e.g. anonymous

Configuration of the production chains

There should be specified one chain for each data type.

The parameters controlling the production chains are:

nof datatypes:	number of data types
datatypes:	list of data types

For each data type there should be defined a production chain

<datatype> *nof steps*	number of steps for this specific data type
<datatype> *steps*	list of the steps in the production chain for specific data type

Each step is specified by a describing name, the function to call, an input dataset, an output dataset, and a configuration file

Optional main configuration parameters

The other main parameters are optional:

project	default is ‘cryo’
maxLog	maximum size of logfile in Bytes, default is 100KB
image catalog	full path to an existing directory, default is $CRYO_TOP/data

2.1.3.3 Setup of the production steps

In the current system, there is defined a production chain for MODIS data. This is done through the definition of the parameter *modis steps* in the main configuration file. This parameter comprises several items that represent the steps in the production chain. The steps are performed in the sequence that they are listed in the *modis steps* parameter.

Each step is defined by a corresponding item in the parameter *modis steps*. Each item will typically fill one line in the main configuration file, consisting of a number of strings that represent:

- a reference name (one or more strings) to be shown in the GUI
- name of function to call (one string)
- input directory (one integer)
- output directory (one integer)
- the name of a configuration file (one string)

The following example shows how the production chain could be defined in the main configuration file. Note that the parameter *modis nof steps* will determine how many steps in

the chain that actually will be performed. Also note that each step, except the last one, should be followed by a comma.

```
modis nof steps = 4          # Number of steps in the chain
modis steps = {
import               import_modisdata        1  2  importModis.cfg,
make products        basic_products          2  3  makeProd.cfg,
resample products    projectproducts         3  4  resampl.cfg,
export products      export_modisproducts    4  5  export.cfg   }
```

The system programmer may control what will happen in each step by editing the corresponding configuration files, according to section 2.3. As a general rule the operator should not change these files.

- The import step will consider a list of MODIS swath files and import specified subsets of these files.
- The basic_products step will consider a list of basic products and make these products. Although the list of products to make is defined in the corresponding configuration file, *makeProd.cfg*, the sequence of their processing is controlled by the basic_products step. The geometrical reference grid will also be produced in this step.
- The resample step will consider a list of basic products and resample them into a specified map projection. The geometrical reference grid established in the previous step will be utilized.
- The export step will consider a list of products and export them to final products intended for some users.

2.2 System Operator's Guide

The CRYO production chain for snow products are implemented as a plug-in to the ENVI software. It is started from a simple GUI, but the interactions during the processing steps are kept to a minimum.

From the GUI, the user may choice to
- perform ENVI and IDL commands
- run the production chain in remote modus
- run the production chain in local modus
- stop the processing

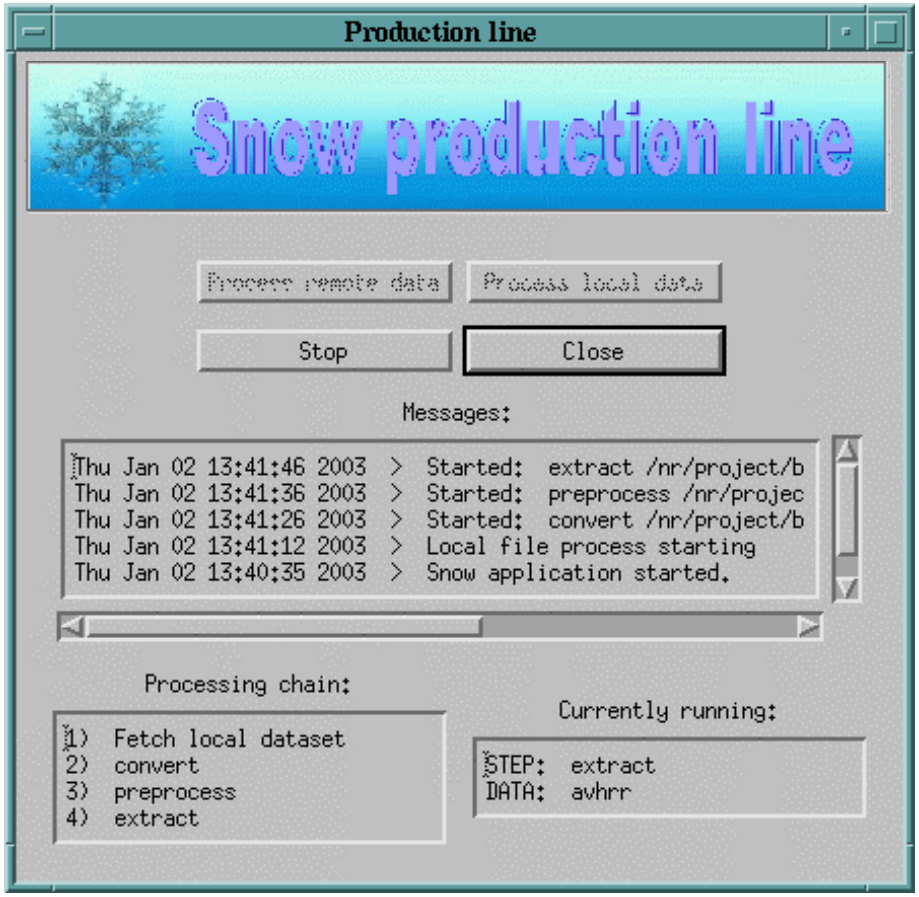
2.2.1 Starting the software

The production chain should be run on a *linux* platform. In order to start the production chain, the user needs to run a setup program, and then start ENVI:

```
> setenv IDL_PATH '<IDL_DEFAULT>'
> source cryo_setup
> envi
```

Now the user will see the ENVI prompt and may enter ENVI and IDL commands from that prompt. In addition, a simple menu will be available:

The lower 'ENVI' button gives access to the complete ENVI menu. By clicking the button labelled 'SNOW', a pulldown menu appears and the user will start the automatic production chain by clicking 'Snow process'. Depending on the implementation, the label on the 'SNOW' button may differ, e.g. 'CRYO'. After calling the automatic production chain, the GUI on the next page will become available, and the prompt will be locked for input.



The user may choose to start the production chain in local modus ('process local data') or in remote modus (process remote data'). In both modi the central field of the GUI will report back the progress through the various steps. Only the remote modus utilize the automatic down-loading facilities in the framework.

The user may stop the processing by clicking the 'Stop' button. By clicking the 'Close' button, the control is returned to the ENVI prompt and the initial menu will be made available.

The operator may monitor the progress by means of the GUI. The lower left pane gives an overview of the steps in the chain. The lower right pane identified the step that is currently being processed. The central pane yields a more detailed log of the steps in the chain, including if some steps return with an error.

2.2.2 Remote modus

When run in remote modus, the system will wait for e-mails to arrive. Each e-mail will be parsed, and the specified files will be downloaded and put through the production chain according to the data type specified the e-mail. When a dataset has been processed, the system will wait for the next e-mail to arrive. The control may be returned to the user by pushing the 'stop' button.

2.2.3 Local modus

The local modus of the production chain does not utilize the automatic downloading facilities in the framework, but requires that the datasets exist somewhere in the local file system.

By selecting the local modus, the user will get access to ENVI's file selector in order to select a text file containing a list of local datasets. A dataset is a set of files, representing one single scene, residing in a separate directory. The dataset list file will contain one line for each dataset. This line consists of the full path to the directory, followed by a specification of the image type in order to identify the appropriate production chain.

When run in local modus, the system goes through the list in sequence and processes one dataset at the time undertaking all steps required for the given data type. When all datasets are processed, the control is returned to the production line menu.

A more detailed set of instructions for running the chain in local modus is given in section 2.3.

2.2.4 Process control

The user may stop the processing by pushing the 'stop' button. In some cases it may take some time for the system to react to the stop request.

Temp files

Note that the directory *tmpdir* in the current system will not be strictly temporary. It may also contain files needed by the software for the management of multi-temporal data series. As a general rule assume that temporary files will be removed by the software itself.

NOTE that when starting a new series of data processing, the old calibration values should be removed by cleaning up the tmp directory.

Logfiles

Log messages from the system will be written to the command window, as to the logfile, which resides in the *programs/log* directory. When the logfile *cryo.log* reaches its maximum size it will be backed up as *cryo.log.prev.*. If it is required to save more of the logging info, the user should make copies of the logfile regularly or change the maximum allowed filesize.

Error handling

If the system aborts, the software should be terminated and then restarted. However, a lot of error situations are managed by the software without aborting the processing. In these cases, the processing of the current dataset will be terminated and the processing of the next dataset will be started.

In the central field of the GUI the main steps in the process are being logged, with a simple error message if something has gone wrong. If an error has occured, the program stops the process for the given dataset and starts processing the next dataset. No detailed error messages are given here, but you can see in which step in the process the error has occurred. In the logfile more detailed error messages are given.

The most common errors happen during import of data. Typical errors are:

1. The directory containing the input-files does not exist. Error in the list file (see 0).
2. One or more of the input files are missing. For calculating the SCA, files with names MOD021KM* and MOD02QKM* must be present.
3. One or more files are incomplete.

From the message in the logfile you should be able to find the type of error.

Warnings

In the log file or the command window warnings may appear. Most of these warnings may be ignored by the operator. When old calibration data have been used, you will get a message, and you have to notice that the confidence of the result is being reduced.

If the resulting SCA image is showing clouds or no data for the complete area, there are two possible reasons:

1. The complete area is covered by clouds. This will rarely happen.
2. Too many of the calibration areas are covered with clouds. A number of chosen areas completely covered with snow and some areas completely without snow are used as calibration for the SCA algorithm. If too few of these areas are seen from the satellite, the value of SCA can not be calculated, unless calibration values can be retrieved by other means.

In such cases the last calculated calibration values are being used. These values could originate from an earlier pass of the same day or from an earlier day. As the signals received by the satellite from one location may change during the day, and from day to day, the use of old calibration data is not optimal. The calculated SCA values should be considered less confident than values calculated with calibration data from the same scene. In the log file you will get warnings if old calibration data has been used. If no old calibration data exist, there will be no SCA result.

When old calibration data have been used, you will get a message, and you have to notice that the confidence of the result is being reduced.

2.3 Instructions for operating in local modus

The steps to be handed by the operator when operating in local modus are:

- Download data
- Edit .lst file
- Start processing chain
- Upload products

2.3.1 MODIS data

Data download

Create a directory for the new dataset

```
>     mkdir <path_to_modis_data>/yyyy.mm.dd_hhmm
```

Download data from KSAT or NASA by means of ftp. The dataset are represented by two products of level1b (MOD021KM and MOD02QKM). Download both of them.
KSAT data should be found in the MODIS-testdata directory on the ftp-server ftp3.tss.no.

```
>     cd <path_to_modis_data>/yyyy.mm.dd_hhmm
>     lftp -u norskr,<passwd> ftp3.tss.no
lftp  cd MODIS-testdata
lftp mget MOD021KM* MOD02QKM*
```

Edit list file

Edit the file *$SNOW_TOP/data/modis_sca.lst* (or make a new file *modis_yyyymmdd.lst*)

For each scene to be processed, the file should contain one line like:

```
      <path_to_modis_data>/yyyy.mm.dd_hhmm   modis
```

Start the processing chain

```
>     setenv SNOW_TOP <path_to_processing_chain>
>     cd $SNOW_TOP/programs
>     setenv IDL_PATH '<IDL_DEFAULT>'
>     source cryo_setup
>     envi
```

From the GUI:

- Select 'SNOW' and then 'Snow process'
- Select 'Process local data'
- Pick the appropriate .lst file (the one you just edited) *$SNOW_TOP/data/modis_sca.lst*

Upload the product

The end product from the processing chain is located in the directory: *$SNOW_TOP/data/modis/out<n>/yyyy.mm.dd_hhmm*

The .png file in this directory can easily be inspected visually. It should show ocean water, clouds, and SCA values.

Copy the product directory to the outgoing FTP:

```
>     cp $SNOW_TOP/data/modis/out4/yyyy.mm.dd_hhmm <outgoing_ftp>
```

2.3.2 AATSR data

The processing of AATSR data is always undertaken in local mode. There are some modifications compared to MODIS.

Data download

The AATSR data is delivered from ESA on high-capacity tapes. The data should be copied to disk by the linux tar command. There is one directory for each day, containing orbits that day. The directory structure of the tar-set is yyyy/mm/dd/ATS_TOA_1P

Edit list file

The production chain needs to refer to unique datasets. Therefore the list file should therefore refer to datasets named as ATS_TOA_1PRUPAyyyymmdd/.

Start the processing chain

This is similar to MODIS.

Upload the product

This is similar to MODIS.

3 System Developer's Manual

3.1 Introduction

The application software is programmed in IDL, with calls to ENVI. Some functions are programmed separately and are available as system calls to executable binaries.

The application software is organized into software modules, providing one or more functions that can be called by the framework as a step in the production chain.

3.2 Interaction between the framework and the application software

Since the framework does not know the application software, the application functions must fulfil some requirements in order to interact with the framework (see sect. 1.3). This section describes these requirements.

The application program interface (API) in the framework will call the application functions through this specified function call:

```
status = funcName(  dataset_in, dataset_out, $
                    staticdir, tmpdir, logfile, configFileName )
```

All arguments are input arguments that specify directories and files that are made available to the application function. The return value is the error status, which may force the production chain to terminate. The relationship between the function call and the corresponding line in the main configuration file is illustrated by an example: Consider the production chain specification in section 2.1.3.3. The item

```
"make products        basic_products          2  3  makeProd.cfg"
```

will define the following function call:

```
status = basic_products( CRYO_TOP/data/modis/out1/<dataset_name>, $
                         CRYO_TOP/data/modis/out2/<dataset_name>, $
                         CRYO_TOP/data/static,                    $
                         CRYO_TOP/data/tmp,                       $
                         CRYO_TOP/programs/log/cryo.log,          $
                         makeProd.cfg)
```

Note that the numbers 2 and 3 correspond to out1 and out2 respectively.

3.2.1 Main principles

The dataflow through the chain is controlled by the two first arguments *dataset_in* and *dataset_out*, which refer to two different directories that are unique to each specific scene. Thus they are dynamic arguments that will depend on the actual scene being processed.

The argument *dataset_out* in one function will typically be identical to the argument *dataset_in* in a function called in a later step in the production chain. This directory thus serves as the connection between the two steps.

The other arguments always refer to the same directories or files, independent of what scene that is being processed. These directories and files are thus common data.

Since the framework lacks the possibility to provide the functions with application specific arguments, all such arguments are transferred by means of configuration files. The system operator controls the production chain by managing the configuration files.

- Input directory – *dataset_in:*
  The input directory is dynamically set by the framework. It will typically be the output of a preceding step in the production chain.
- Output directory – *dataset_out:*
  The output directory is dynamically set by the framework. It will typically become the input to a succeeding step in the production chain.
- Static directory – *staticdir:*
  The static directory ($SNOW_TOP/data/static) is a fixed directory where static data should be found,. Such data may be configuration files, water masks, training areas, class definitions, colour tables, etc. The static data may be organized in sub-directories.
- Temporary directory – *tmpdir:*
  The temporary directory ($SNOW_TOP/data/tmp) is a fixed directory where temporary file could be put by the application software. The files in this directory could be deleted at any time when they are not being actively used. However, note that the *tmpdir* in this application is used for storing the state of multi-scene processing.
- *Logfile:*
  This argument identifies a fixed text file ($SNOW_TOP/programs/log/snow.log) intended for appending log-messages.
- Configuration file – *configFileName:*
  This argument identifies the name of a text file that contains the input parameters required by the application software. The application software should know where this file is expected to be found, but it is common practice to use the static directory for these files. The format of the configuration files is described in Appendix.
- Return value – *status:*
  A successful completion should return 1.

3.2.2 Modifications and specifications

This section describes the practices that have been followed concerning the files and directories referred in the API. Note that the practice concerning temporary files may be considered as a violation or modification of the main principles that were initially defined for the framework.

Temp files

In order to manage multi-temporal datasets, some commonly available data directory had to be made available for temporary versions of a multi-temporal product. The problem is that the framework does not provide a directory for this purpose. The chosen strategy was to store such temporary multi-temporal products in the scene-specific directories and their references in the *tmpdir* directory. This directory should therefore not be deleted.

Since the role of the *tmpdir* directory was changed to managing multi-temporal scenes, all temporary files relating to specific scenes was put into the scene-specific directories *dataset_in* and *dataset_out*. Typically, temporary subdirectories are being used for this purpose in the current application software.

Input/output directories

These directories may contain temporary subdirectories, as described above.

The output directory may already contain data produced in a preceding step or sub-step, typically when more than one product are being produced. In some cases the application functions may need to use these files as input files.

The application software should never use the input directory for output, but may optionally delete files no longer needed.

Configuration files

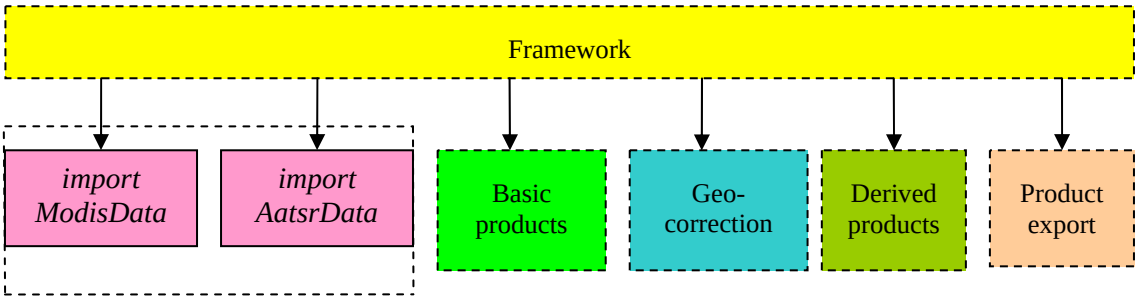
The *configFileName* parameter may contain any string variable. It is assumed that this string contains the name of a file residing in the *staticdir* directory, and that this file is a configuration file, i.e. a text file that follows the configuration file format defined in the framework.

The current software does not expect to find all configuration parameters directly in the file referred to by the *configFileName* parameter. Instead it is expected that this file refers to other configuration files, which may contain configuration parameters common to more than one function. The intension of this strategy is to reduce the risk of inconsistent configuration data.

3.3 Module descriptions

In this chapter each module is described in more detail. For each module its main functions are described. All the functions are named like *functionName*, while the modules will be referred to as 'module name' in this chapter.

3.3.1 Module data import



The data import module consists of one main function and an optional one:

- *importModisData* is the main function and reads MODIS images and stores specified image layers (spectral subsets) of them as ENVI files
- *importAatsrData* is the main function that use BEAM software to rectify and mosaic AATSR data and stores specified image layers of them as ENVI files

3.3.1.1 Function importModisData

The purpose of the function *importModisData* is to import subsets of MODIS datafiles, including radiometric data, geo-location data and view angle data.

Input data

MODIS calibrated data (MOD02) stored as *.hdf files. These files also contain meta-data.

The input directory may contain MODIS files of these types:

- MOD021KM: Image data, 1 km resolution
- MOD02HKM: Image data, 500 m resolution
- MOD02QKM: Image data, 250 m resolution
- Metadata, including acquisition date and time

The scenes are in the original acquisition geometry (swath geometry).

The hdf-files contain a lot of data, including:

- Calibrated image data, represented as integers, and supplied with calibration coefficients for converting the integers to real radiance values. When applicable, there are also calibration coefficients for reflectance values.
- Geo-location info, for points regularly distributed over the image grid
- View angle info, for points regularly distributed over the image grid. Note that this data field is contained in MOD021KM only.

Output data

For each specified input image, there may be generated three types of ENVI-files:

data: containing the specified spectral bands from the hdf-file, stored as integers

latlon: containing the latitude and longitude for the geo-location points, stored as a two-layer float image with one cell for each geo-location point

angle; containing the view angle from the hdf-file, stored in a one-layer integer image. The angle image will be expanded to the same dimensions as the data image. Note that angle data will be produced for MOD021KM only.

Configuration data

The configuration file may contain these parameters:

imgTypes:	*list of image types, allowed values are '1KM', '500', '250'*
250_import:	*name of cfgFile, triggered if imgTypes contains '250'*
500_import:	*name of cfgFile, triggered if imgTypes contains '500'*
1KM_import:	*name of cfgFile, triggered if imgTypes contains '1KM'*

Each of these files provides these configuration arguments for their respective imgType:

imgType:	*for identification only*
bandnames:	*list of band names identifying the spectral bands to retrieve*
angleFields:	*list of angle 'bands' to read, should specify SensorZenith, SolarZenith, SolarAzimuth*
latLon:	*flag (0 or 1) whether latlon data should be read for this imgType*

Interactions

Called from framework

Calling: *getSpatialSubset* if requested

Optional function getSpatialSubset

The purpose of the *getSpatialSubset* function is to compile a contiguous swath-scene that covers a given region of interest (ROI) from north to south. This process may include the merging of adjacent scenes and the removal of scans that are outside the region.

The function is called from the *importModisData* function as an option. It will be trigged if *importModisData* identifies the configuration parameter *clip* or *merge* as set.

This function substitutes the current imported scene with a compilation of the existing imported data that fulfil the requirements. These requirements are specified in the file given by the configuration parameter ROI, e.g. ROI=clip.cfg

If *merge* is set, the configuration file should also contain the parameter *sceneList*, which identifies a file in *tmpdir* that will refer to the preceding scene processed

The configuration parameters for this function are omitted in this version of the document.

Configuration data

The configuration file provides these parameters for the *getSpatialSubset* function.

clip	*flag if scenes are to clipped, i.e. spatial subset extracted*
ROI	*triggered by clip; defines the area to be covered by the clipped scene*
merge	*flag if adjacent scenes are to merged*
sceneList	*keep track of previous scenes*

3.3.1.2 Function importAaatsrData

The purpose of the function import_AatsrData is to act as a wrapper to the two functions undertaking the geometrical and radiometrrical corrections. These two functions does not follow the standard parameter interface.

The input and output to the wrapper function is given by input to the geometrical correction and the output of the radiometrical correction.

Input data

AATSR data in Envisat format (N1-files) residing in dataset_in. All files to be merged should reside in one directory. There will typically be one file per orbit and one directory for each day.

These data are typically received on tape or portable hard disks. In that case they may reside in a separate directory given by the parameter 'remoteDir'. The purpose of dataset_in will then only be to specify the name of the dataset, following the convention above.

Output data

The default output of the geo-corrected data is the DIMAP-format. This format is a hierarchical format. The DIMAP file itself (<dataset>.dim) contains metadata only. The image data resides as Envi-files in a directory with the same name (<dataset>.data) as the dataset. There will be one output file for each band. In the lab prototype the dim-file is ignored, and the Envi-headers used instead.

This convention is followed for all data in DIMAP format: The data are found in a directory found by adding '.data' to the datasetname. This is taken care of by the reading tools.

If radiometrical correction is requested, the output will be in Envi-format, in the <dataset> directory.

Configuration data

The parameters to the import function are simply the configuration files for two functions that are being called:

beamMosaicCfg = dimapMosaic_Europa.cfg

topoCfg = topoPanEuropa.cfg

If one of these parameters is unspecified, the corresponding function will not be called.

3.3.1.3 Function beam_mosaic

This function will import AATSR data, including radiometric data and view angle data into a predefined grid. The function is a wrapper around the BEAM program mosaic, which projects and merges a set of Envisat scenes.

Input data

The input data are as specified for the wrapper function.

Output data

The default output is the DIMAP-format as specified for the wrapper function.

Configuration data

bandNames: list of band names identifying the spectral bands to retrieve
angleNames: list of angle 'bands' to read, e.g. 'SensorZenith'
projection : e.g. geographic
mapExtent : UL_E UL_N LR_E LR_N (in degrees)
cellSize : (in degrees, eg. 0.01)
resampling : method for resampling, eg. BL
background : value for missing data, e.g. -1
orthoDem : e.g. GETASSE30
remoteDir : path to where N1-data has been downloaded
firstHour : ignore orbits earlier than this hour
lastHour : ignore orbits later than this hour
prep : set this flag to prepare for BEAM only
post : set this flag to postprocess BEAM result only
call : set this flag to a full import, incl. a call to BEAM

Comment

The BEAM software is controlled by a request file. The beam_mosaic function can be considered as composed of three parts 1) making the request file and copying the N1- data from the repository ('remotedir'), 2) running Beam, 3) checking and correcting the result.

The import function can be configured to exit after the request files have been generated (prep=1: step 1 only) in order to run the BEAM as a standalone program. In that case it is necessary to re-run the import function in order to fix some errors produced by the BEAM software (post=1; step 3 only). Setting call=1 will run all 3 steps.

3.3.1.4 Function getilluminationmodel

Input data

The input data is the output of the beam_mosaic function, found in dataset_out. The image bands used are the solar elevation and azimuth angles. In addition a terrain gradient file is used. This location of the gradient file is given on the configuration file. It contains gradients derived from the DEM, and are I the same grid system as the image.

Output data

The output data is an Envi file containing the direct illumination of each pixel relative to the illumination from zenith on a horizontal surface.

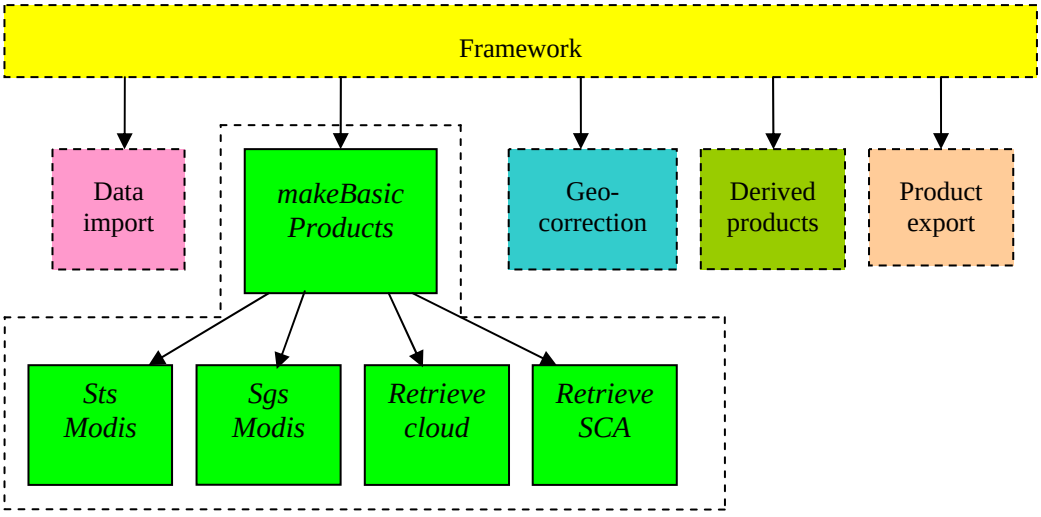
Configuration data

gradientFile = terrain/gradients_panEuropa_cdeg

The terrain directory is a subdirectory of staticDir.

3.3.2 Module basic products

This module undertakes the production of basic products. By basic products are meant products that can be retrieved from one single dataset. This input scene will typically be in image (swath) geometry in the Modis production chain and in a map grid for the AATSR production chain. For swath data the *latlon* file will be copied forward in the processing chain for later resampling.



The basic products that can be produced by this module are: Cloud Cover, and Snow covered Area (SCA). In addition Surface Temperature Snow (STS) and Snow Grain Size (SGS).

3.3.2.1 Function makeBasicProducts

This function serves as an organizer of the basic product generation functions. Essentially it produces all requested products from the data stage to the basic product stage.

One important issue for the function *makeBasicProducts* is to process the cloud mask before the other products. Therefore, independent of the sorting of the products in the configuration file, the *makeBasicProducts* will ensure that the cloud cover will be produced before the snow products. For swath data it will ensure that required geolocation information will follow the basic products.

Input data

The input directory contains ENVI files produced by the data import module.

data: files of the required types

latlon: files (will be copied forward, unless data files are already resampled)

angle: data for 1KM data only, to be used in cloudCover and in STS

Configuration data

The configuration data simply defines what to produce. There is one list for the basic products and one list for the map reference grids (geo-index maps). For each of the listed items, there must be specified a configuration file.

The list of products is defined by specifying the cfgFiles that should be used in the functions that produces the requested products, e.g.

products: *list of basic products to make, i.e. among {cloud, SCA, STS, SGS}*
cloud.cfg: *cfgFile for cloudModis*
sca.cfg: *cfgFile for scaModis*
sts.cfg: *cfgFile for stsModis*
sgs.cfg: *cfgFile for sgsModis*

Output data

basic-conf file containing the basic confidence, used for retrieving the product confidences

<prod>: product files

<prod>-conf product specific confidence files

latlon geolocation files, copied forward if present in input

Interactions

Called from framework

Calling *makeGeoIndex* if requested
cloudModis
scaModis
stsModis
sgsModis

3.3.2.2 Function *cloudModis*

The purpose of the *cloudModis* function is to make a cloud mask from the image data.

The cloud mask is the result of a kNN classification of the MODIS data. It requires 1KM data, but can be resampled to a different cell size.

Input data

The input directory contains ENVI files produced by the data import module.

data file should represent a MODIS scene of type 1KM that contains the MODIS bands 1, 4, 6, 19, 20, 26, and 31 as its first 7 layers.

Configuration data

maskCodeFile: *specifies the classCodes to be used in the output product*
ROI *defines the geographical area where cloud classification is required*
knn.cfg: *defines a configuration file for the details of the kNN classification. The knn.cfg file should never be modified.*

Output data

cloud product resolution as specified; classCodes as specified

Interactions

Called from *makeBasicProducts* function

3.3.2.3 Function *cloudAatsr*

The purpose of the *cloudAatsr* function is to make a cloud mask from the image data.

The cloud mask is the result of a kNN classification of the AATSR data or by using SYKE's SCDA algorithm for cloud masking.

Input data

The input directory contains files produced by the data import module.

dataset_in: AATSR data containing all nadir bands
AATSR data containing solar elevation

static_dir: watermask

Parameters

method = syke *specifies the method for the cloud classification*
syke.cfg: *defines a configuration file for the details of the SYKE classification.*
maxSolarZenith = 70.0 *specifies a value for the maximum solar zenith angle allowed*

maskName = masks/PanEur_watermask_wgs84_latlon01deg_envi

maskCodeFile: *specifies a file containing the classCodes to be used in the output product*

Output data

dataset:out/cloud the mask contains codes for clouds and other features

Interactions

Called from the framework and not from *makeBasicProducts* function

3.3.2.4 Function *retrieveSCA*

The *retrieveSCA* function is a wrapping function that retrieves the snow covered area (SCA) from the image data using NLR or another available method.

Configuration data

scaResol:	*defines the resolution for the SCA product*
method	*defines if the NLR, SCAmod or Shi/Enveo algorithm should be used.*
SCAmod.cfg	*identifies a configuration file that defines parameters used in the algorithm*
ShiEnveo.cfg	*identifies a configuration file that defines parameters used in the algorithm.*

Note

If the method is nlr, the same config file is used as for the wrapper

3.3.2.5 Function *sca_aatsr*

The *sca_aatsr* function is a function that retrieves the Fractional Snow Cover (FSC) from the AATSR image data using a specified method. It is assumed that a cloud mask resides in the dataset_out directory.

Configuration data

method	*defines if the NLR,SCAmod or Shi/Enveo algorithm should be used.*
calibMask	*mask for the NLR algorithm*
nlr_threshold_file.cfg	*configuration for nlr*
topo.cfg	*config file for toographical correction*
SCAmod.cfg	*identifies a configuration file that defines parameters used in the algorithm.*
combMask	*defines where to use SCAmod and where to use nlr*

3.3.2.6 Function *nlr_sca_compute*

This *nlr_sca_compute* function retrieves the snow covered area (SCA) from the image data using the NLR algorithm. Some of the parameters are valid for Modis only.

Input data

The input directory contains ENVI files produced by the data import module.

data file	should represent a MODIS scene of any type should contain MODIS band 1 as its first band
cloud file	residing in the output directory, it should have the same geometry as the data file
calib file:	residing in the static directory, defining training areas in a map geometry

latlon file:	geolocation file for MODIS swath data, to be used for transfer the training areas from the map geometry to the input image geometry
basic-conf file,	residing in the output directory, for producing the SCA-confidence

Configuration data

scaResol:	*defines the resolution for the SCA product from MODIS*
nlr_threshold_file:	*identifies a configuration file specific for the NLR method. Details are not shown here. This file should never be modified*
calibMask_<scaResol>	*calibration mask to be applied by the NLR method for given scaResol*

Output data

SCA product file:	basic product with SCA values for every cell without respect to confidence, cloud cover or land mask
SCA-conf file:	confidence for basic SCA product

Interactions

Called from *retrieveSCA* function

3.3.2.7 Function *scamod_fsc_compute*

This *scamod_fsc_compute* function retrieves the snow covered area (SCA) from the image data using the SYKE (SCAmod) algorithm.

Input data

The input directory contains ENVI files produced by the data import module.

data file	should represent a MODIS scene of any type should contain MODIS band 1 as its first band
cloud file	residing in the output directory, it should have the same geometry as the data file

Configuration data

transFile :	*path to the file containing the trasmittivity map*
forest_refl	*e.g 6 (6%)*
ground_refl	*e.g.7 (7%)*
wetsnow_refl	*e.g.66 (66%)*

Output data

SCA product file:	basic product with SCA values for every cell without respect to confidence, cloud cover or land mask
SCA-conf file:	confidence for basic SCA product

Interactions

Called from *retrieveSCA* function

3.3.2.8 Function *enveo_sca_compute*

This *enveo_sca_compute* function retrieves the snow covered area (SCA) from the image data using the Shi (enveo) algorithm.

Input data

The input directory contains ENVI files produced by the data import module.

data file	should represent a MODIS scene of any type should contain MODIS band 1 as its first band
cloud file	residing in the output directory, it should have the same geometry as the data file

Configuration data

forestFile path to forestmask
waterFile path to watermask

 # Codes: waterCode = 40
forestThres = 30
thres_ndsi = 0.7
thres_ndsi_mixed = 0.12
thres_refl06 = 15
thres_refl06_mixed = 7.0
snowOpenCode = 1
snowMixedOpenCode = 2
snowForestCode = 3
snowFreeOpenCode = 4
snowFreeForestCode = 5

 # Minimum number of endmembers needed to make fsc-estimate
noEndMemb = 3
searchRadius = 30

nofbands = 4

#Code used for mixedpixel where number of endmembers located are below noEndMemb
nonValidCode = 200

Output data

SCA product file:	basic product with SCA values for every cell without respect to confidence, cloud cover or land mask
SCA-conf file:	confidence for basic SCA product

Interactions

Called from *retrieveSCA* function

3.3.2.9 Function *stsModis*

This function retrieves the surface temperature (STS) of snow from calibrated thermal MODIS data. The STS product is produced by means of Key's algorithm.

Input data

The input directory contains ENVI files produced by the data import module.

data file :	should represent a MODIS scene of type 1KM. should contain the MODIS bands 31 and 32 among its layers.
angle file:	should represent the view angle. It is used in Key's algorithm
latlon file:	should follow the processing chain if data file is not resampled

Configuration data

stsMethod: *identifying what method to use, default is 'Key', no other options yet*
keyFileName: *identifies a file with the complete set of coefficients for Key's algorithm*

The rest of the parameters aim at identifying what subset of Key's coefficients to use and should not be changed

Output data

STS product file — product with STS values for every cell without respect to confidence, cloud cover, snow cover or land mask

latlon file — forwarded from input directory if required

Interactions

Called from *makeBasicProducts*.

3.3.2.10 Function *sgsModis*

This function retrieves a snow grain size index (SGS) from the image data

Input data

The input directory contains ENVI files produced by the data import module.

data file: should represent a MODIS scene of type 1KM.
should contain the MODIS bands 1 and 5 among its layers.

latlon file: forwarded from input directory if required

Configuration data

This function has currently no configuration data.

Output data

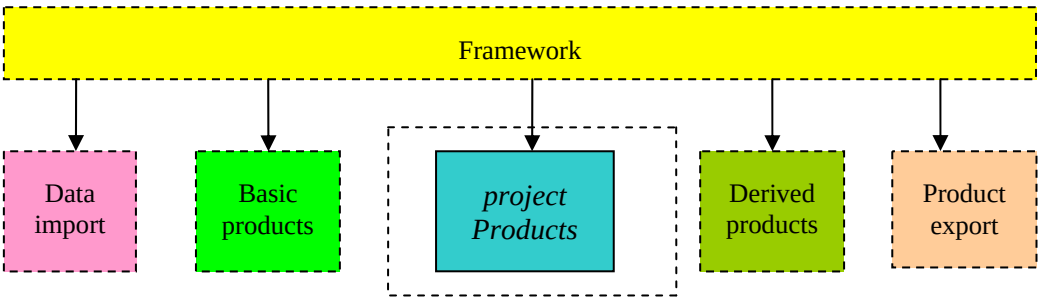
SGS product file — product with SGS values for every cell without respect to confidence, cloud cover, snow cover or land mask

latlon file: should follow the processing chain if data file is not resampled

Interactions

Called from *makeBasicProducts*

3.3.3 Module projectProducts



This module provides one function, which is callable from the framework through the API:

- *projectProducts* is a function that projects the specific image data or products into the given geometrical reference system.

3.3.3.1 Function *projectProducts*

The purpose of the *projectProducts* is to take a swath image and/or a set of basic products derived form such an image, and resample it into a map grid. The function will first make a geo-index map that will map the swath image into the grid, and then use that index in the resampling of the swath data.

Input data

The input data is a list of basic products from the *makeBasicProducts* module (or alternatively image data from the *dataImport* module), together with their corresponding *latLon* file.

Configuration data

The resampling of the products is specified like this:

products:	*list of products to resample*
<product>_resamplMet	*the resampling method to use for a given product*
<product>_conf	*flag if also the product configuration file is to be resampled*
cell size	*cell size of output products*
projCfg:	*name of cfgFile for the projection definition, as above*

This configFile specifies the map projection and area by means of these parameters:

Projection:	*projection type, default 'UTM'*
UtmZone:	*triggered if projection is UTM*
Datum:	*datum name, default is 'WGS-84'*
UpperLeft:	*map projection coordinates of upper left corner*
LowerRight:	*map projection coordinates of lower right corner*

projectionName: to be used for identification / reference

Output data

For each of the specified products, one product file and optionally one confidence file resampled to the specified map projection.

Interactions

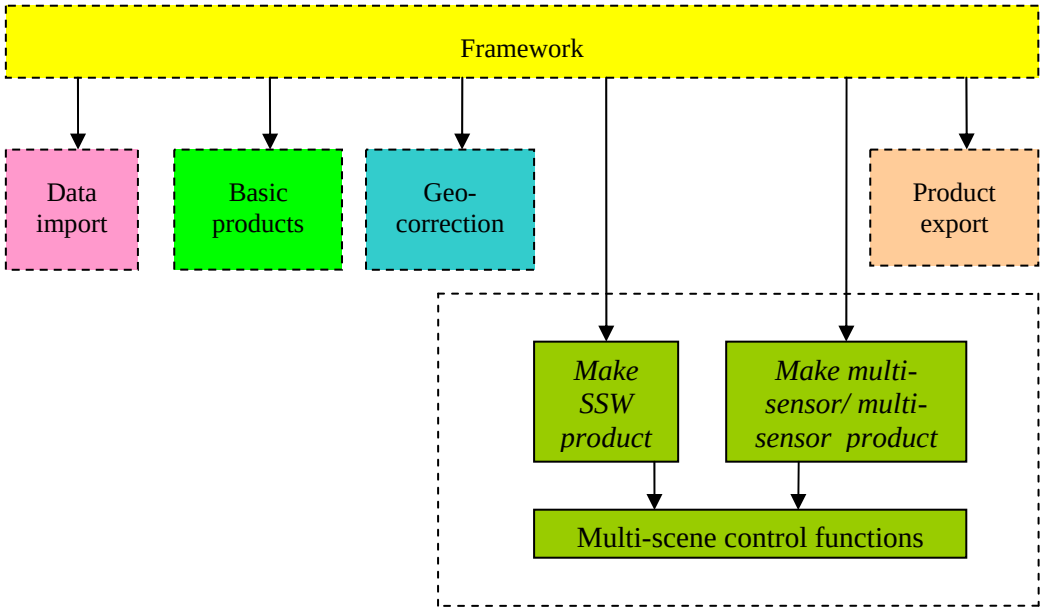
Called from framework

Calling *makeGeoIndex*

Note: The resampling method for the cloudMask should be specified as BL on order to identify all map cells that are influenced by clouds.

Note: Former versions of the system required *makeGeoIndex* as a separate step in the production chain or as a part of the basicProducts step. Now all this is handled directly by the *projectProducts* step. Also note that the former calls to the compiled C-programs 'spawn_corr' are obsolete in the current version.

3.3.4 Module derived products



This module handles products where one needs to consider a time-series of one or more basic products. All functions in this module require that all input products are given in a common geometric reference. The functions also require a cloud mask or a confidence map for each input product.

The functions need to consider the time sequence of various basic products through the production chain. The framework itself only controls the various steps for one particular scene at the time, and does not know anything about the other scenes in a time-series. Therefore the multi-functions need a common toolbox, here referred to as a multi-scene controller or a time-series controller. This controller keeps track of the current time-series, and gives access to scenes that belong to the current time series.

3.3.4.1 Function makeSSW

This function estimates an index for snow surface wetness (SSW) based on the current change in SGS in addition to the current value of STS

Input data

The input consists of geo-corrected ENVI product files

- STS product file
- SGS product files in a short time-series
- SCA product file
- Cloud cover product file

Configuration data

 # Codes for SSW
startday = 1
maxdays = 5

 # Temperature limits

sts_snow_tresh = 3.0
sts_wet = 0.5
sts_moist = -0.5
sts_dry = -2.0

 # Grain size limits
sgs_snow_tresh = 70
sgs_diff_min= -0.9
sgs_diff_max= 0.9
code_list = ssw_codelist.cfg

 # Parameters internal to the SSW algorithm, to be applied on SGS product
sgsOffset = 100 # to be added to SGS prodcut values
scaLimit = 80 # SCA limit between partly and full snow coverage
bareCode = 55 # code to be applied where SCA is below scaLimit (partly snow cover)
simpleOutput = 1

Output data

SSW product for the current day

Interactions

Called from framework

3.3.4.2 Function makeMultiSceneSCA

This module will consider current SCA results within some running time frame and identify the best observation within that period. The required inputs are SCA products their corresponding confidence products.

Input data

The input consists of geo-corrected ENVI product files

- sca product files in a time series
- sca confidence files that corresponds

Configuration data

listFile = images10d.lst
omask   = observMask.cfg
product = sca
accType = bestobs
period   = yearly
dayFactor = 10

Output data

SCA product file for the current day

Interactions

Called from framework

3.3.4.3 Function aatsr_aggregate_snowextent

This module considers current FSC results within some running time frame and identify the best observation within that period. The required inputs are FSC products and their corresponding confidence products.

Input data

The input consists of geo-corrected product files

- Previous aggregated product, included mask and confidence
- Current product, included mask and confidence

Configuration data

listFile = images10d.lst
maskcodefile = mask_codes.cfg defining codes for the mask
product = nlr-SCAmod for identification of correct input files
dayFactor = 10 time span, count down from 100 until 0

Output data

FSC product file for the current day, with mask and confidence. The mask contains mask codes for the last unconfident observation and cannot be used directly.

3.3.4.4 Function makeMultiSensorSCA

Input data

The input consists of geo-corrected ENVI product files

- sca product files from various sensors in a time series
- sca-confidence files that corresponds

Configuration data

As for multi-scene

In addition a product specific confidence factor to adjust or weight the confidences from the various sensors.

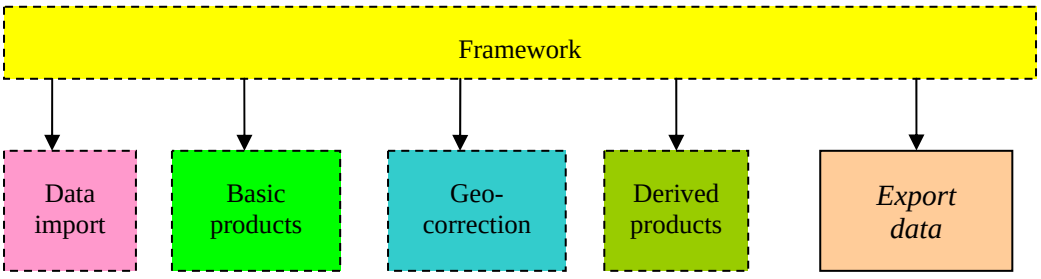
Output data

sca product file for the current day

Interactions

Called from framework

3.3.5 Module data export



The module data export contains the function *exportData*, which combines basic products or derived products with various masks in order to make a presentable final product. It also

converts ENVI files to other file formats, if requested. It contains one function that can be called from the framework:

3.3.5.1 Function *exportData*

The current version of *exportData* considers one product and exports it according to specifications in the configuration data.

Input data

The input directory must contain a file with the specified basic or derived product.

Configuration data

The configuration files should specify these parameters:

product :	*what product to export; default is 'sca'*
dataOffset:	*offset value to add to value of geophysical parameter; default = 0*
landMask_250 :	*landmask to be included if imgType is 250*
landMask_1KM :	*landmask to be included if imgType is 1KM*
coltab	*text file with colour table to be applied in the exported product*

In addition the class (mask) codes in the input and output files should be specified.

Output data

The raw products are combined with the corresponding cloud mask and the static land/water mask into a presentable result. The result may be gives as ENVI-files and/or tiff-files in gray tones and/or colours, as well as a jpg-file or png-file in colours.

3.3.5.2 Function export_globsnow

The current version of `export_globsnow` combines basic products or derived products with the corresponding mask and confidence map in order to make a presentable final product. The product is stored in HDF-format.

Input data

The input directory must contain a file with the specified product, a mask and a confidence map.

Configuration data

product	*what SE product to export, e.g. SCAmod or nlr-SCAmod*
confProduct	*sca confidence product produced, e.g. sca-conf*
mask:	*maks product to export, e.g cloud*
level	*L3A(single products) or L3B (multi products)*
classCodings	*e.g. fsc and/ or 4cl*
coltab	*text file with colour table to be applied in the exported product*
region	*panEurope ; to be put in meta-data*

Output data

One HDF-file for each of the specified classCodings. In addition a color png-file is produced according to the fsc coding.

4 Appendix

4.1 Configuration files

The configuration file should contain information needed by the controller to perform the right operations on the different datasets. The general format of the configuration file is as follows. It should consist of keyword-value pairs: <keyword> = <value>. (The keywords currently defined are marked with bold font in the example below). The keyword should consist of one or more strings and be separated from the value with the '=' sign. The value may consist of either a single value which can be one or more strings ended by EOL, or it can be a list. The start and end of the list should be marked by parentheses, and each item should be separated by a comma: {<item1>, <item2>, <item3>}. Each item may consist of one or more strings, and the list may run over several lines. Comments should start with an '#' and end at EOL.

4.1.1 The main configuration file

In the following example, the format of the configuration file is described in more detail. The example shows the production chain for two similar processes from two different types of data. These are MODIS and AATSR.

Note that local mode only is available for AATSR data. However, the main configuration file still needs to contain the lines referring to remote processing.

```
# ----------------------------------------------------------------------
#                  CONFIGURATION FILE
#          Production line for cryospheric variables
# ----------------------------------------------------------------------
project = cryo               # Name of project. Default value: 'cryo'
maxLog  = 100000             # Maximum size of logfile in bytes
                             # Default value: 100KB

image catalog = /nr/project/bild/images
                             # Must be an existing directory!
                             # Default value: $CRYO_TOP/data

# The following parameters must be set - there are no default values

# Specify the number of (external) providers and their ID.
nof providers = 2             # Number of data providers
providers = {ksat, nasa}      # Names of providers.
                              # For download of data over ftp, there
                              # will also need to be one ftp-configuration
                              # file per provider. The format of this will
                              # be explained below.

# For each provider, specify the mail address of the sender
# (There should only be one sender per provider)
ksat sender = mailer@ksat.no
nasa sender = mailer@nasa.no


#Specify the number of datatypes and their ID.
nof datatypes = 2                    # Number of datatypes
datatypes = { modis, aatsr}          # Names of datatypes
```

```
# Specify the number of steps in the processing chain for the datatypes.
# Syntax:    <datatype name> nof steps = <nof steps>
# There should be one processing chain for each data type.
# If there are defined more steps than the number specified,
# the last ones will be ignored
#
modis nof steps = 4

# Specification of steps in the processing chain.
# Syntax:  <datatype name> steps = { … }
# There should be one line for each step, where each step
# will correspond to a function in the “method API”
# Each line should be comma separated. The syntax of each line is:
#  <name of operation> <name of function> <dataset in> <dataset out> <cfg >
# Where:
# <name of operation> - the name which will appear in the GUI
# <name of function>  - the corresponding name of the function to be called
# <dataset in>        - a number specifying the input dataset
# <dataset out>       - a number specifying the output dataset
# <cfg>               - name of a configuration file for the function.
#                       If none is needed a dummy name should be given.
#                       (The file could be located under ‘static’ directory)
modis steps = {
import            import_modisdata        1    2   ksat_import.cfg,
basic             basic_products          2    3   basic_products.cfg,
project           projectproducts         3    4   project_products.cfg,
export            export_modisproducts    4    5   sca_export_ksat.cfg }

# Here the datasets which are input and output should be given a number as
# an ID, with the numbering starting from 1. This corresponds to dataset
# number 1, which will always be the original raw data fetched either
# remotely or locally. Hence, in the example above, the original dataset
# will be input to import, and the import function will put the output in
# the directory for dataset number 2 (out1). Dataset number 2 will then
# be input to basic_products, which puts the results in dataset 3 (out2).
# The step projectproduc reads from dataset 3 and puts the result in
# dataset 4 (out3). Finally, an export function is used to show the results
# in appropriate formats and the output dataset is found in dataset 5 (out4)

# Specification of the steps in the Aatsr processing chain.
#
aatsr nof steps = 6
aatsr steps = {
import aatsr      import_aatsrdata        2  2   preprocessing.cfg,
cloud             cloud_aatsr             2  3   cloud_aatsr.cfg,
sca               sca_aatsr               2  3   sca_aatsr.cfg,
multi             aggregate_snowextent    3  5   aggregate_10d.cfg,
export single     export_globsnow         3  4   export_L3A.cfg,
export multi      export_globsnow         5  6   export_L3B.cfg}

# Specification of local mail. (Needed for automatic remote download.)
mailusr  = cryo              # local mail user
mailpwd  = cryo              # local mail password
mailhost = mail.nr.no        # local mail host
mailport = 100               # local mail port
```

4.1.2 The relationship between the configurations files

The main configuration file makes references to each function to run, and to a corresponding configuration file. Each of these configuration files will specify a set of arguments. These arguments may be

- Ordinary arguments
- References to static or dynamic files
- References to other configuration files

The references to other configuration files build up a hierarchy or network of configuration files. In particular this is the case for AATSR data where the range for choosing methods is wide. This purpose of this section is to give an overview of this structure.

- aatsr_prepocessing.cfg
    - dimapMosaic.cfg [bands to read; map coordinate system and extent]
    - topoCorrection.cfg [how to calculate the illumination]
- aatsr_basic_products.cfg
    - cloud_aatsr.cfg
        - method
        - cloud_syke.cfg
        - cloud_knn.cfg
        - mask_codes.cfg
        - mask_name [watermask to be used
    - sca_aatsr.cfg
        - method
        - calibMask [to be used by the NLR method]
        - nlr_thresholds.cfg
        - SCAmod.cfg
        - ShiEnveo.cfg
        - combMask [for combining nlr and scamod]
- aggregate.cfg [for making multi-products]
    - mask_codes.cfg [same as for cloud masking]
    - list_file [to keep record of former products]
- export.cfg
    - colTab [file with color table]

5 References

Andersen, T. 1982, "Operational snow mapping by satellites," Hydrological aspects of alpine and high mountain areas, Proceedings of the Exeter symposium, July 1982, IAHS publ. no. 138, pp. 149-154.

Amlien, J and Solberg, R, 2004. “Evaluation of algorithms for the retrieval of snow surface temperature from medium resolution satellite data”. The 8th Circumpolar Symposium on Remote Sensing of Polar Environments, Chamonix, France, 08-12 June, 2004.

Key, J.R., J. B. Collins, C. Fowler, and R. S. Stone, 1997. “High-latitude surface temperature estimates from thermal satellite data”, *Remote Sensing of Environment*, 1997. 61(2), pp. 302-309.

Koren, H, Solberg, R and Amlien, J. 2004. “Evaluation of algorithms for the retrieval of snow grain size from optical satellite data”. The 8th Circumpolar Symposium on Remote Sensing of Polar Environments, Chamonix, France, 08-12 June, 2004.

Malnes, E, Storvold, R, Lauknes, I; Solberg, R; Amlien, J and Koren, H, 2005 “Multi-sensor monitoring of snow parameters in Nordic mountainous areas” IEEE International Geoscience and Remote Sensing Symposium (IGARSS 2005), Seoul, Korea, 25-29 July 2005

Metsämäki, S.J., Anttila, S.T, Huttunen, J.M and Vepsäläinen, M, 2005. A feasible method for fractional snow cover mapping in boreal zone based on a reflectance model. Remote sensing of Environment 2005, 95, pp 77-95.

Shi, J, 1999 “Estimation of snow fraction using AVIRIS simulated ASTER image data” Proc. of Eighth Airborne Geos. AVIRIS Workshop JPL, Calif. Inst. of Technology, Pasadena, CA, February 10-11, 1999

Solberg, R. and T. Andersen, 1994. “An automatic system for operational snow-cover monitoring in the Norwegian mountain regions,” Geoscience and Remote Sensing Symposium (IGARSS), Pasadena, California, USA, 1994.

Solberg, R, Amlien, J, Koren, H, Eikvil, L, Malnes, E, and Storvoll, R. 2004a. Multi-sensor and time-series approaches for monitoring of snow parameters. IEEE International Geoscience and Remote Sensing 2004

Solberg, R, Amlien, J, Koren, H, Eikvil, L, Malnes, E and Storvold, R, 2004b. “Multi -sensor/multi-temporal analysis of ENVISAT data for snow monitoring” ESA ENVISAT & ERS Symposium, Salzburg, Austria, September 06-10, 2004.

Solberg, R, J Amlien, H Koren, E Malnes and R Storvold 2005, “Multi-sensor multi-temporal snow cover area algorithms. Part 1: Mountain regions ” Envisnow EVG1-CT-2001-00052. Norut, Feb. 2005.

Solberg, R, Amlien, J, Koren, H, Eikvil, L, Malnes, E and Storvold, R 2005b “Multi-sensor/multi-temporal approaches for snow cover area monitoring” EARSeL LIS-SIG Workshop, Berne, February 21-23, 2005.