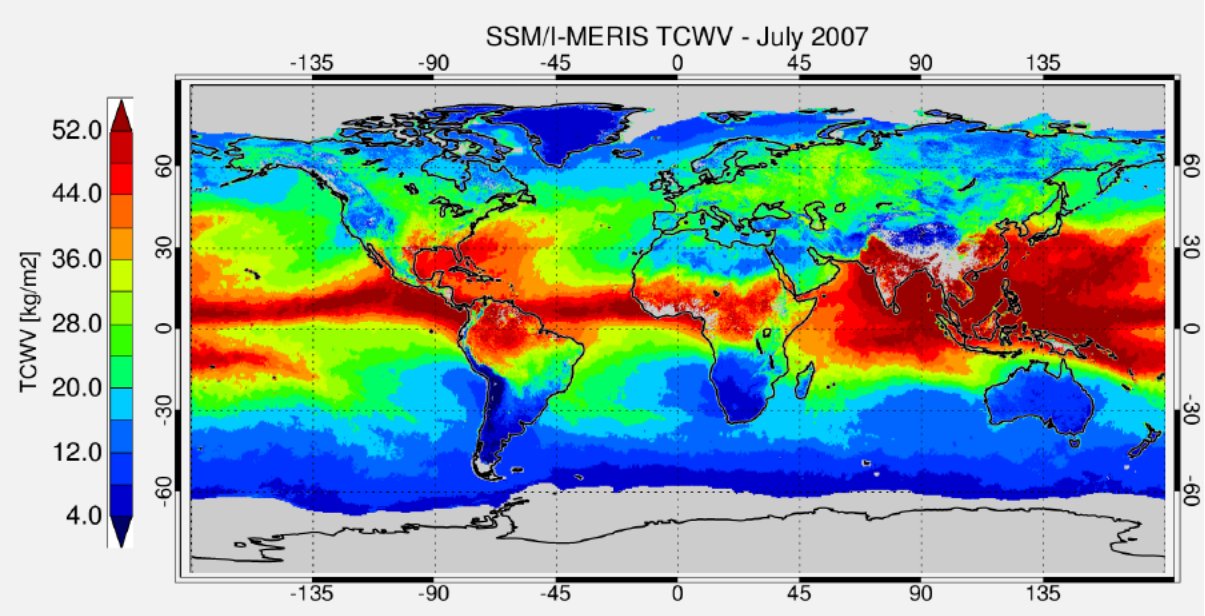

CAWA D-9: Software User Manual

Release 1.0

Brockmann Consult GmbH

Aug 07, 2017

Contents

Introduction

Project background

The SEOM S3 'advanced Clouds, Aerosols and WAter vapour products for Sentinel-3/OLCI' CAWA project aims to the development and improvement of advanced atmospheric retrieval algorithms for the Envisat/MERIS and Sentinel-3/OLCI mission. A sensor comprehensive and consistent 1D-Var water vapour algorithm has been developed and applied to the MERIS, MODIS and first available OLCI measurements. An innovative and consistent cloud top pressure 1D-Var procedure was defined for MERIS and all three OLCI O2 A-band channels, which will significantly improve the retrieval accuracy. The challenging and innovative GRASP algorithm for the retrieval of aerosols and surface properties has already shown its advantage in comparison to conventional aerosol retrieval methods. All three algorithms will be further improved, applied to the complete MERIS dataset, to a four months MODIS global time series and six months of OLCI data. We expect to create improved consistent datasets of water vapour, cloud properties, namely cloud top pressure, and aerosol and surface pressure. The intention of the CAWA team is to establish new and improved procedures to estimate atmospheric properties, which also improve the retrieval of land and ocean properties.

Purpose and Scope

This document is the User Manual for the SNAP TCWV and CTP processors written in Python and Java which have been developed in the frame of the CAWA project. Its purpose is to describe in detail how to obtain, install and operate these processors. Also, a comprehensive overview of all related data products (input as well as intermediate and final products) is provided.

The explicit structure of the document is as follows:

- Chapter 1 is this introduction.
- Chapter 2 gives an overview of the SNAP CAWA TCWV and CTP processing system.
- Chapter 3 describes all relevant SNAP CAWA products.
- Chapter 4 explains how to get and install the processing software.
- Chapter 5 explains how to run the processing software.
- The Annex contains various annexes.

References

1. ADVANCED CLOUDS, AEROSOLS AND WATER VAPOUR PRODUCTS FOR SENTINEL-3/OLCI: Technical, Management and Financial Proposal. Issue 1.0, 28.03.2014.
2. Retrieval for Total Coulumn Water Vapor from MERIS/OLCI and MODIS for Land- and Ocean Surfaces. CAWA TCWV ATBD, available at: https://earth.esa.int/web/sppa/activities/cawa/projects-documents
3. Retrieval of Cloud Top Pressure from MERIS and OLCI O2 A-Band Measurements. CAWA CTP ATBD, available at: https://earth.esa.int/web/sppa/activities/cawa/projects-documents
4. The Sentinel Application Platform (SNAP) Web Site, available at: http://step.esa.int/main/toolboxes/snap/

5. Configure Python to use the SNAP-Python (snappy) interface, available at: https://senbox.atlassian.net/wiki/display/SNAP/Configure+Python+to+use+the+SNAP-Python+%28snappy%29+interface
6. CoastColour Project Web Site, available at: http://www.coastcolour.org
7. OceanColour Project Web Site, available at: http://www.esa-oceancolour-cci.org
8. Bourg, L. (2009): MERIS Level 2 Detailed Processing Model. ACRI-ST, Document No. PO-TN-MEL-GS-0006, 15 July 2009.
9. GlobAlbedo Project Web Site, available at: http://globalbedo.org
10. LandCover Project Web Site, available at: http://www.esa-landcover-cci.org
11. GlobAlbedo ATBD 'Pixel Classification'. Version 4.1, 26 June 2012.
12. ERA-Interim global atmospheric reanalysis dataset, available at: http://www.ecmwf.int/en/research/climate-reanalysis/era-interim
13. European Space Agency: Meris Product Handbook, Issue 3.0, 1 August 2011.
14. MODIS Level 1B Product User's Guide. For Level 1B Version 6.1.0 (Terra) and Version 6.1.1 (Aqua). MODIS Characterization Support Team, Document PUB-01-U-0202- REV C, February 27, 2009.
15. Climate Data Operators (CDO) Web Site, available at: https://code.zmaw.de/projects/cdo
16. The Python Download Web Site, available at: https://www.python.org/downloads/
17. SNAP Wiki: Configure Python to use the SNAP-Python (snappy) interface, available at: https://senbox.atlassian.net/wiki/display/SNAP/Configure+Python+to+use+the+SNAP-Python+%28snappy%29+interface

Acronyms and Abbreviations

Acronym	Definition
ATBD	Algorithm Theoretical Basis Document
BC	Brockmann Consult
BEAM	Basic ERS & Envisat (A)ATSR and Meris Toolbox
Calvalus	CAL/VAL and User Services
CAWA	Advanced Clouds, Aerosols and WAter vapour products
CCI	Climate Change Initiative
CTP	Cloud Top Pressure
DFS	Distributed File System
DUE	Data User Element
ESA	European Space Agency
GPF	Graph Processing Framework
GRASP	Generalized Retrieval of Aerosol and Surface Properties
JDK	Java Development Kit
MERIS	Medium Resolution Imaging Spectrometer
MODIS	Moderate Resolution Imaging Spectroradiometer
MR	Map-Reduce
NetCDF	Network Common Data Form
OLCI	Ocean and Land Colour Instrument
RAM	Random Access Memory
SEOM	Scientific Exploitation of Operational Missions
SNAP	Sentinel Application Platform
SNAPPY	SNAP Python
TCWV	Total Column of Water Vapour
TOA	Top Of Atmosphere
UCAR	University Corporation for Atmospheric Research

The SNAP Cawa TCWV and CTP Processing System

Overview

The key goal of the CAWA project regarding software development, production and dissemination was to implement the proposed algorithms for TCWV and CTP in free and easily accessible open source toolboxes, notably and foremost ESA's SNAP toolbox. After successful implementation, TCWV and CTP datasets from the full MERIS archive were generated with BC's 'Calvalus' Linux cluster following the project targets. In addition, TCWV from several months of 'OLCI-like' input datasets (i.e. MODIS Aqua/Terra products MOD021 and MYD021) were generated. However, the SNAP TCWV and CTP processors are in principle fully portable and can be run on any Linux platform. The procedure for installation and operation is described in this chapter.

Theoretical Background

The motivation and theoretical background for the TCWV and CTP retrieval is summarized in the CAWA project proposal [1]. The underlying algorithms are described in detail in the corresponding ATBDs for TCWV [2] and CTP [3], respectively.

Processing Environment

Most of the TCWV and CTP processing in the frame of the CAWA project has been carried out on BC's Linux-based processing system ('*Calvalus*' = CAL/VAL and User Services), based on the so-called MapReduce (MR) programming model, combined with a distributed file system (DFS). Calvalus uses *Apache Hadoop*, which is a Java open source implementation of MR and DSF. It gains its performance from massive parallelization of tasks and the data-local execution of code, which avoids expensive network traffic. Actually the Calvalus system has ~90 cores, ~ 1 PetaByte data storage volume. It is extensively used within various projects.

However, as said, the SNAP TCWV and CTP processors can in principle be set up and run on every Linux based computing systems. This is described in more detail in section *Processing Software Installation*.

Processor Components

The SNAP TCWV and CTP processing system consists of the following SNAP software components and auxiliary datasets:

- *snap-core* module
- *snap-gpf* module
- *snap-python* module
- *snap-netcdf* module
- *s3tbx-idepix* module
- *snap-cawa* plug-in
- *snap-cawa-io* plug-in
- FORTRAN shared libraries providing high-performance utility functions used in *snap-cawa*

- lookup tables for TCWV and CTP retrieval

These components are described in more detail in the following subsections.

The Sentinel Application Platform (SNAP)

A common architecture for all Sentinel Toolboxes is being jointly developed by Brockmann Consult, Array Systems Computing and C-S called the Sentinel Application Platform (SNAP).

The SNAP architecture is ideal for Earth Observation processing and analysis due to various technological innovations as well as approved concepts from the BEAM toolbox. Most of the software components listed above make use of various SNAP core capabilities.

A good starting point for much more detailed information is the SNAP homepage [4], and also the comprehensive help documentation integrated in the SNAP desktop application.

The SNAP Graph Processing Framework

One of the key components in SNAP is the Graph Processing Framework (GPF) for creating user-defined processing chains. Both CAWA TCWV and CTP processors make use of this framework.

Within SNAP, the term data processor refers to a software module which creates an output product from one or more input products configured by a set of processing parameters. The GPF framework was originally developed for BEAM. Since the early days of BEAM, a number of data processors have been developed; some of them are standard modules while others are contributed by 3rd parties. All of these data processors have been developed using a dedicated processing framework which was already part of the first version of BEAM.

Based on the experience collected within a number of projects, the SNAP authors have developed what is now the SNAP Graph Processing Framework. The GPF provides all the features inherited from BEAM, but adds a number of new ones for developers and reduces the amount of source code to write while drastically improving its readability and maintainability.

Much more detailed information on the SNAP GPF is provided by the specific GPF help documentation integrated in the SNAP desktop application.

The SNAP-Python Interface (SNAPPY)

A new concept provided in SNAP is the possibility to develop preocessing scripts using Python. This is realized by a new SNAP-Python extension (SNAPPY). This component basically provides a bi-directional communication between Python and Java since the Python extension code must be able to call back into the Java APIs. This communication is realized by the bi-directional Python-Java bridge ‘jpy’, which comes with a number of outstanding features, such as

- Fully translates Java class hierarchies to Python
- Support of Java multi-threading
- Fast and memory-efficient support of primitive Java array parameters (e.g. NumPy arrays)

The jpy Python module is entirely written in the C programming language. The same resulting shared library is used as a Python jpy module and also as native library for the Java library (*jpy.jar*). This means that

- Python programs that import the '*jpy*' module can load Java classes, access Java class fields, and call class constructors and methods.
- Java programs with *jpy.jar* on the classpath can import Python modules, access module attributes such as class types and variables, and call any callable objects such as module-level functions, class constructors, as well as static and instance class methods.

SNAPPY can also be used from the Graph Processing Framework so that in SNAP scientific GPF operators can be developed not only in Java, but now also in Python. In CAWA, both TCWV and CTP processors are making use of this and were written in Python, whereas the pre-processing (i.e. the IdePix pixel classification) uses a GPF processor which was written in Java.

More detailed information on SNAPPY can be found in [5].

The SNAP-NetCDF Module

The SNAP NetCDF module provides comprehensive capabilities for NetCDF file I/O within SNAP, based on the set of NetCDF software packages provided by UCAR Unidata. In return the SNAP NetCDF module is used by the *snap-cawa-io* module which ensures a project-related generation of TCWV and CTP products in CF-compliant NetCDF format. See section *The SNAP CAWA Products* for more detailed description of the CAWA TCWV and CTP products.

The IdePix Pixel Classification Module

IdePix (Identification of Pixels) is a pixel classification tool which has been developed by BC as BEAM plugin and has been used for a variety of projects. The tool works over both land and water and supports a variety of sensors. Among these are MERIS and MODIS, which made IdePix the most appropriate candidate for cloud and snow identification in the CAWA project.

The IdePix tool for water pixel classification was developed in the frame of the ESA DUE project 'CoastColour' [6], and the ESA OceanColour CCI project [7]. The classification is mainly based on the algorithms described in [8], chapter 5.

The IdePix tool for land pixel classification was developed in the frame of the ESA DUE project 'GlobAlbedo' [9], and the ESA LandCover CCI project [10]. The classification is mainly based on the algorithm used for GlobAlbedo as described in [11].

Although Idepix has been tested and successively improved within GlobAlbedo using a wide selection of regions, also taking into account seasonal variations, some limitations and weaknesses in cloud detection (most of them well known from other existing cloud masking approaches) could not be solved to 100%. These are i.e.

- distinction of cloud and snow/ice is often difficult
- detection of optically very thin clouds
- possible misclassifications over very bright land areas, e.g. deserts or bright beaches

Therefore, within the frame of various projects, the IdePix algorithms are continuously further developed.

In the meantime IdePix has also been integrated in SNAP as modules for both the Sentinel 2 and the Sentinel 3 toolboxes. The latter module ('s3tbx-idepix') provides the support for MERIS and MODIS which is needed for CAWA. This module in return makes use of the SNAP Graph Processing Framework (GPF) described above.

The pixel classification with IdePix is the first processing step in CAWA, applied on the MERIS/MODIS L1b products as preprocessing towards the generation of both TCWV and CTP (see Figure 1).

The TCWV GPF Processor

The TCWV GPF processor is the key component of the SNAP TCWV processing chain. This processor also makes use of the SNAP GPF framework, and also of the SNAP Python interface (SNAPPY) described above. The processor provides the implementation of the TCWV algorithm described in detail in [2].

CAWA TCWV core is meant to be the core of a L1B –> L2 processor, for the retrieval of total column water vapor. It is sensor independend, curently MERIS and MODIS look up tables are provided. It works only for cloud free pixel

Basically, the processor is sensor-independent. However, specific lookup tables are required which are currently provided for MERIS and MODIS. In summary, the processor needs the following inputs:

- normalized radiance (TOA radiance divided by solar constant) at the window and absorption bands [sr-1]
- geometry
- surface (or 2m) temperature [K]
- surface pressure [hPa]
- aerosol optical thickness at the short wave window band
- prior windspeed (for ocean pixels)
- land sea discrimination (as implementations for land and sea slightly differ)

The output of the processor is TCWV [mm] and a TCWV flag (i.e. valid data mask).

The TCWV processing flow is illustrated in Figure Figure 1.

The CTP GPF Processor

The CTP GPF processor is the key component of the SNAP CTP processing chain. As the TCWV processor, the CTP processor also makes use of the SNAP GPF framework, and also of the SNAP Python interface (SNAPPY) described above. The processor provides the implementation of the CTP algorithm described in detail in [3].

Basically, the processor is also sensor-independent. Again, specific lookup tables are required which are currently provided for MERIS and OLCI. The processor works for all pixel, however only cloudy pixel deliver sensible results. The cloud optical thickness does not account for optical effective radius (missing SWIR Bands), thus it will not be accurate in particular close to cloud/rain bows.

The underlying algorithm has been designed in two versions:

- ‘cloud_core’. A slim and faster version being used for MERIS, only retrieving cloud top pressure and cloud optical thickness.
- ‘cloud_complete_core’, the full version, additionally retrieving cloud profile information. This version had been foreseen for OLCI, but in the end was not realized as GPF processor, as the optional ‘OLCI’ workpackage had been descoped from the CAWA project.

In summary, the processor needs the following inputs:

- normalized radiance (TOA radiance divided by solar constant) [sr-1] at the window and absorption bands (Band 10 and 11 in case of MERIS. The MERIS band 11 is corrected for straylight using coefficients which are provided with the processor module.)
- surface pressure [hPa]
- surface albedo around 750 nm. (An examplarily climatology is provided with the processor module.)
- the precise deviation of the central wavelength from the nominal

The output of the processor is CTP [hPa] and a CTP flag (i.e. valid data mask).

The CTP processing flow is illustrated in Figure Figure 2.

FORTRAN shared libraries

The core algorithms for both TCWV and CTP processors are implemented in Python, which is convenient and popular. However, compared to others, it is usually not the fastest programming languages. Therefore, for the most computation intensive parts of the code as well as for frequently used utility functions, equivalent high-performance FORTRAN modules have been developed. These modules were pre-compiled, and appropriate shared libraries for Linux are provided to the processing software package.

Lookup Tables

Various lookup tables are used for both TCWV and CTP retrieval, as described in more detail in [2] and [3]. These lookup table are also provided to the processing software package.

Processing Flow

Although the TCWV and CTP processors are completely independent of each other, their individual processing flow is very similar as shown and explained below.

TCWV Processor

The overall processing flow of the SNAP TCWV processor is shown in Figure 1.

As mentioned, L1b products from MERIS or MODIS are used as input. These products are pre-processed with the IdePix pixel classification module. Idepix provides a classification flag and the reflectance bands (converted from radiances in case of MERIS) needed for the TCWV retrieval. Further optional input (per pixel) are prior values for temperature, pressure, wind speed, and an initial TCWV guess. Ideally, these priors are taken from an external data source to provide values of good quality. For the CAWA TCWV processing on Calvalus, these data were taken from ERA-Interim [12] products which were interpolated and collocated onto the initial L1b/IdePix product grid. If no priors are provided, the processor will use reasonable constant values, but this is not recommended for good TCWV retrievals.

The IdePix products (optionally including the prior bands) are the input for the TCWV processing step, which provides the final TCWV products (TCWV + flag band).

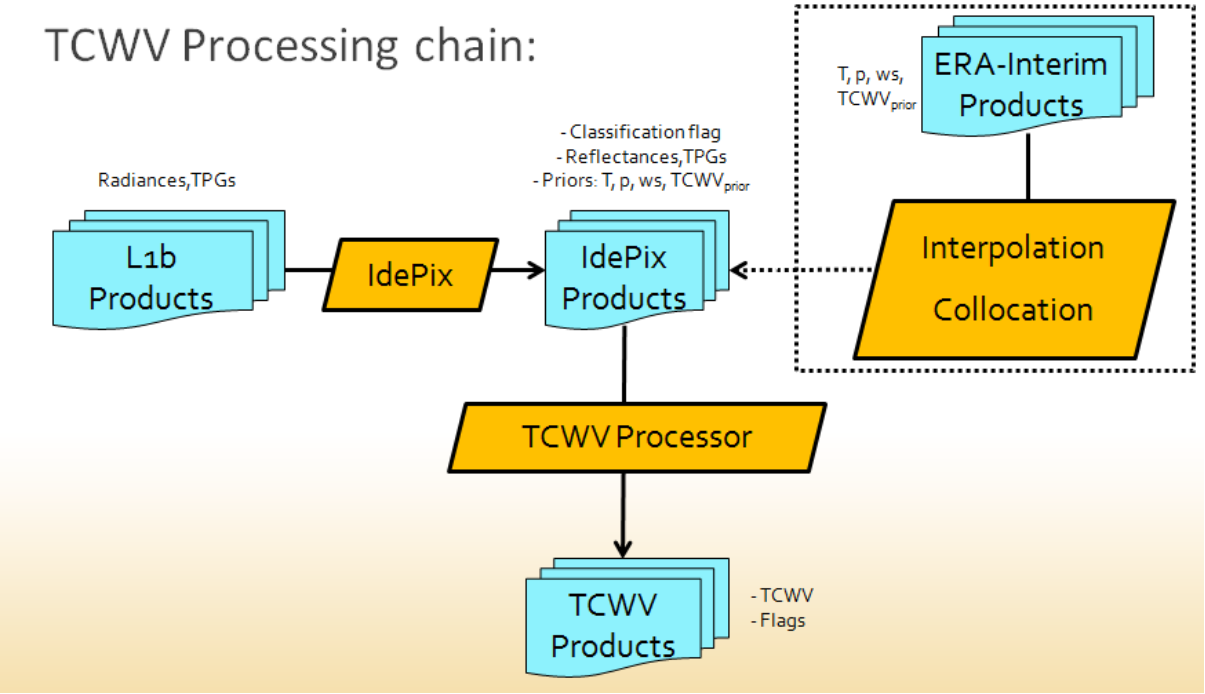

Figure 1: Processing flow of the SNAP TCWV processor.

CTP Processor

The overall processing flow of the SNAP CTP processor is shown in Figure 2.

The setup and structure of the CTP processor is very similar to the TCWV processor. Again, the L1b products are pre-processed with the IdePix pixel classification module. A surface albedo climatology value (white sky albedo) is added to the IdePix products, using an internal climatology product (20-day averages) which is included in the processor module. The IdePix products are the input for the CTP processing step, which provides the final CTP products (CTP + flag band).

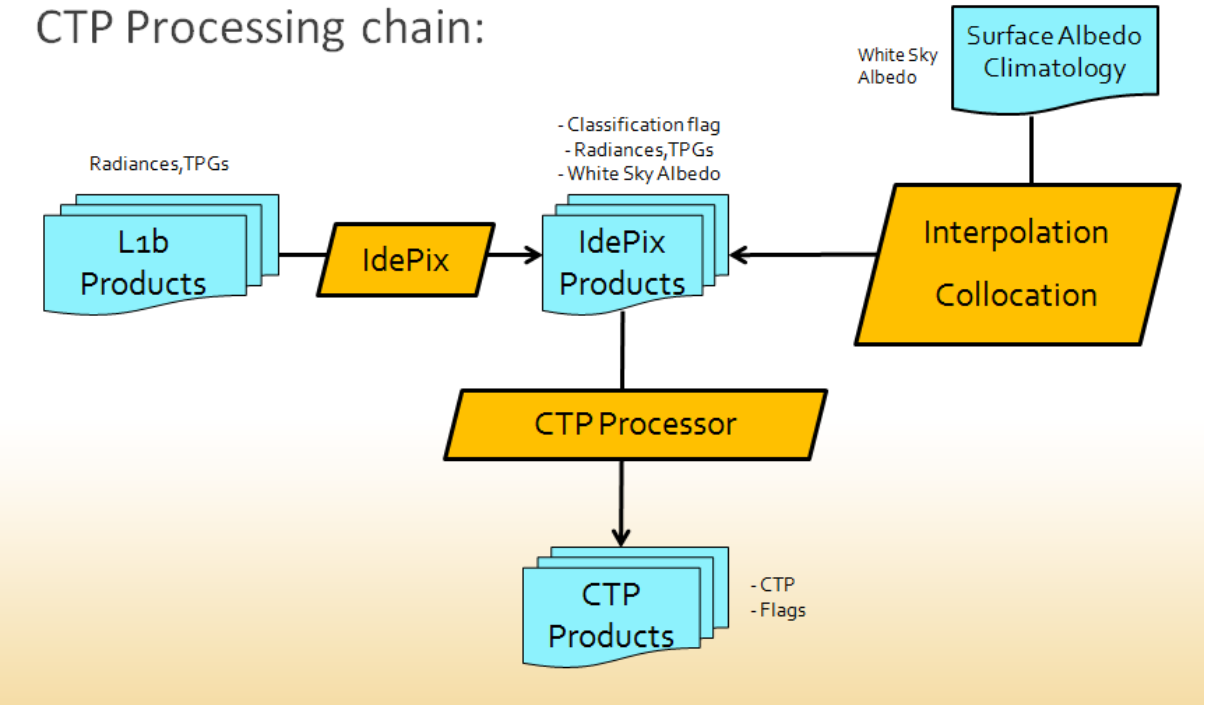

Figure 2: Processing flow of the SNAP CTP processor.

The SNAP CAWA Products

Overview

This section will give an overview of all input, intermediate and final products used and generated by the SNAP GPF TCWV and CTP processors

Input Products

MERIS L1b TOA Radiance Products

From the MERIS full mission (2002-2012), L1b TOA radiance reduced resolution data has been used as input data. Table Table 1 to Table 4 give an overview of MERIS L1b bands, tie point grids and L1b flag coding, respectively. A more detailed description of the MERIS standard L1b product is given in [13].

Table 1: MERIS bands in L1b product

Name in product	Unit	Type	Description
radiance_<n>; n=1,..,15	mW/(m^2*sr*nm)	float32	TOA radiance in band <n>
l1_flags	dl (flag band)	uint8	Level 1b flags
detector_index	dl	int16	Detector index

Table 2: MERIS instrument channels.

Channel	Wavelength	Bandwidth
1	412.5	10
2	442.5	10
3	490	10
4	510	10
5	560	10
6	620	10
7	665	10
8	681	7.5
9	709	10
10	753	7.5
11	761	3.75
12	778	15
13	865	20
14	885	10
15	900	10

Table 3: MERIS tie point grids in L1b product.

Name in product	Unit	Type	Description
latitude	deg	float32	Latitude of the tie points
longitude	deg	float32	Longitude of the tie points
dem_alt	m	float32	Digital elevation model altitude
dem_rough	m	float32	Digital elevation model roughness
lat_corr	deg	float32	Digital elevation model latitude corrections
lon_corr	deg	float32	Digital elevation model longitude corrections
sun_zenith	deg	float32	Sun zenith angle
sun_azimuth	deg	float32	Sun azimuth angle
view_zenith	deg	float32	View zenith angle
view_azimuth	deg	float32	View azimuth angle
zonal_wind	m/s	float32	Zonal wind
merid_wind	m/s	float32	Meridional wind
atm_press	hPa	float32	Mean sea level pressure
ozone	DU	float32	Total ozone
rel_hum	%	float32	Relative humidity

Table 4: MERIS L1b flag coding.

Bit	Flag	Description
0	Cosmetic	Pixel is cosmetic
1	Duplicated	Pixel has been duplicated
2	Glint_Risk	Pixel has glint risk
3	Suspect	Pixel is suspect
4	Land_Ocean	Pixel is over land,
5	Bright	Pixel is bright
6	Coastline	Pixel is part of a coastline

In the CAWA TCWV processing, the following bands and tie point grids from the MERIS L1b products are used:

- radiance_13 (converted to TOA reflectance)
- radiance_14 (converted to TOA reflectance)
- radiance_15 (converted to TOA reflectance)
- sun_zenith
- sun_azimuth
- view_zenith
- view_azimuth

In the CAWA CTP processing, the following bands and tie point grids from the MERIS L1b products are used:

- radiance_10
- radiance_11
- detector_index

- sun_zenith
- sun_azimuth
- view_zenith
- view_azimuth
- dem_alt

MODIS MYD021 TOA Reflectance Products

For the 'OLCI-like' TCWV processing in CAWA, MODIS Aqua L1b data from MYD021KM products were used. These products contain calibrated Earth view TOA reflectance data at 1km resolution, including the 250m and 500m resolution bands aggregated to 1km resolution. The datasets are described in detail in the MODIS Level 1B Product User's Guide [14]. Table 6 gives an overview of the reflective and emissive bands in the MYD021KM product.

Table 5: MODIS Aqua bands in L1b MYD021 product. Taken from [14].

Cryptic name	Resolution	Spectral bands
EV_250_RefSB	250m	1, 2
EV_500_RefSB	500m	3-7
EV_1KM_RefSB	1km	8-19, 26
EV_1KM_Emissive	1km	20-25, 27-36

Here, "RefSB" stands for "Reflective Solar Band" and "Emissive" stands for thermal emissive bands.

Table 6 gives an overview of the tie piont grids available in the MYD021KM product.

Table 6: MODIS tie point grids in L1b MYD021 product.

Name in product	Unit	Type	Description
latitude	deg	float32	Latitude of the tie points (WGS-84), Greenwich origin, positive N
longitude	deg	float32	Longitude of the tie points (WGS-84), Greenwich origin, positive E
Height	m	float32	Height
Range	m	float32	Range
SolarZenith	deg	float32	Sun zenith angle
SolarAzimuth	deg	float32	Sun azimuth angle
SensorZenith	deg	float32	View zenith angle
SensorAzimuth	deg	float32	View azimuth angle

In the CAWA TCWV processing, the following bands and tie point grids from these products are used:

- EV_250_Aggr1km_RefSB_2
- EV_250_Aggr1km_RefSB_5

- EV_1KM_RefSB_17
- EV_1KM_RefSB_18
- EV_1KM_RefSB_19
- SolarZenith
- SolarAzimuth
- SensorZenith
- SensorAzimuth

Intermediate Products

ERA-Interim Products (optional)

The CAWA TCWV algorithm uses the following prior variables:

- temperature at 2m
- mean sea level pressure
- TCWV initial guess
- windspeed at 10m, u-component
- windspeed at 10m, v-component

As said, the way of providing these prior variables to the algorithms is somewhat arbitrary. In the CAWA TCWV processing on the BC Calvalus cluster, the variables were taken from available ERA-Interim reanalysis datasets. The ERA-Interim data extraction and preparation was done with specific scripts which were developed in the frame of other projects and which are running on Calvalus, making use of the collection of Climate Data Operators (CDO) developed at Max-Planck-Institute for Meteorology Hamburg [15]. All these components are not part of the CAWA software package. However, the content of the ERA-Interim products being resampled and collocated with the MERIS/MODIS L1b input products is given in Table Table 7.

Table 7: Bands in ERA-Interim product

Name in product	Unit	Type	Description
t2m	K	float32	temperature at 2m
msl	hPa	float32	mean sea level pressure
tcwv	kgm-2	float32	TCWV initial guess
u10	m/s	float32	windspeed 10m, u-component
v10	m/s	float32	windspeed 10m, v-component
latitude	deg	float32	latitude
longitude	deg	float32	longitude

SNAP IdePix Classification Products

The IdePix classification product is the result of the pixel classification performed on the MERIS or MODIS L1b products for both TCWV and CTP processing. In return, the IdePix product is used as

input for the TCWV and CTP processing. In fact it is an 'extended' classification product containing the following information:

- radiance/reflectance bands needed for TCWV/CTP retrieval
- pixel classification flag band
- prior variables in case of TCWV processing(for CAWA, obtained from collocation with ERA-Interim product described above)
- L1b flags and tie point grids

The IdePix classification flag coding is given in Table Table 8. (Some of the flags may not be computed under certain conditions. E.g., a glint risk is not computed for land pixels.)

Table 8: IdePix classification flag coding.

Bit	Flag	Description
0	INVALID	Pixel is invalid
1	CLOUD	Pixel is either 'cloud sure' or 'cloud ambiguous'
2	CLOUD_AMBIGUOUS	Semi-transparent clouds, or cloud detection is uncertain
3	CLOUD_SURE	Fully opaque clouds with full confidence of their detection
4	CLOUD_BUFFER	A buffer of N pixels (user option) around a cloud
5	CLOUD_SHADOW	Pixel is affected by a cloud shadow
6	SNOW_ICE	Snow or ice pixel
7	GLINTRISK	Pixel has glint risk (over ocean)
8	COASTLINE	Pixel is part of a coastline
9	LAND	Land pixel

The IdePix products are generated in NetCDF4 format. An example of the NetCDF header of an Idepix product is given in the *Annex*.

Final Products

CAWA TCWV Products

The CAWA TCWV final products are generated in CF-compliant NetCDF4 format. They just contain the TCWV, a simple TCWV flag and the pixel classification flag copied from the IdePix product (Table 9). An example of the NetCDF header of a TCWV product is given in the *Annex*.

Table 9: Bands in final CAWA TCWV product

Name in product	Unit	Type	Description
tcwv	mm	float32	Total column of water vapour
tcwv_flags	dl	uint8	TCWV flags
pixel_classif_flags	dl	int16	Pixel classification flags

CAWA CTP Products

The CAWA CTP final products are generated in CF-compliant NetCDF4 format. They just contain the CTP, a simple CTP flag and the pixel classification flag copied from the IdePix product (Table 10). An example of the NetCDF header of a CTP product is given in the *Annex*.

Table 10: Bands in final CAWA CTP product

Name in product	Unit	Type	Description
ctp	mm	float32	Cloud top pressure
ctp _flags	dl	uint8	CTP flags
pixel_classif_flags	dl	int16	Pixel classification flags

Processing Software Installation

Overview

This chapter describes the overall software installation procedure (processing modules and auxiliary data sets) as well as the system requirements (hardware and software).

Usage Requirements

General Requirements

In general, the CAWA processors require:

- a 1.8 version of the Java Development Kit (JDK)
- Python v2.7
- SNAP latest release (currently v5.0.0), including IdePix

Operating System

The software has been developed and tested on Virtual machines based on Linux Ubuntu, which is also used on the Calvalus system. Also, the required FORTRAN shared libraries included in the software bundle were pre-compiled in a Linux environment. Therefore, the CAWA software can be run on Linux systems only.

Hardware Requirements

The CAWA TCWV and CTP Processing System is a complex piece of software which is based on massive numerical operations and lookup table access. Therefore the system requires sufficiently powerful and sufficiently dimensioned hardware for reliable processing. The recommended key parameters for the hardware are:

- Multi-kernel CPU (8 or more), > 3 GHz
- RAM 8GB or more
- sufficient disk space according to sizes of products

Contents of the Processing Software Bundle

The SNAP TCWV and CTP processing software bundle contains the following components in two separate jar files:

- *snap-cawa* plug-in (Python files, lookup tables, FORTRAN shared libraries, GPF configuration files)
- *snap-cawa-io* plug-in

The current processor version is v1.2, therefore we have the two files:

- snap-cawa-1.2.jar

- snap-cawa-io-1.2.jar

How to get the Software

The SNAP TCWV and CTP processing software bundle can be obtained from the CAWA ftp site hosted at BC with the following configuration:

- SFTP, Port 22
- ftp.brockmann-consult.de
- username: cawa
- password: 7t86.8K9i7z
- subdirectory: cawa_processor

Installation Steps

Installation of the SNAP Software

Download SNAP (Unix version) from the SNAP web site [4] and follow the information and instructions for installation given there.

Installation of the Python Software

Download Python v2.7 from the Python web site [16] and follow the information and instructions for installation given there.

Python Configuration

Once downloaded and installed, Python needs to be configured to use the SNAP-Python (snappy) interface. Instructions for this step are given in detail in the SNAP Wiki [17].

Installation of the CAWA Processor modules

The SNAP TCWV and CTP processor modules need to be installed as follows:

- download the *snap-cawa-1.2.jar* and *snap-cawa-io-1.2.jar* into an arbitrary directory
- copy the file *snap-cawa-io-1.2.jar* to $SNAP_INSTALL_DIR/snap/modules
- unpack the *snap-cawa-1.2.jar* into an arbitrary *snap-cawa* directory, e.g. */home/snap-cawa*
- now all required resources should be in another subdirectory */home/snap-cawa/resources_bundle*
- To link the snap-cawa directory to SNAP, edit the file $SNAP_INSTALL_DIR/etc/snap.properties
- **In this file, at the end of the file, add the line:** snap.pythonExtraPaths = *snap-cawa directory*

  e.g. snap.pythonExtraPaths = /home/snap-cawa

How to run the CAWA Processing Software

Test of the Installation

If all installation steps described in *Processing Software Installation* were finished successfully, the CAWA TCWV and CTP GPF processors are now ready to run. First, test their availability with:

```
$SNAP_INSTALL_DIR/bin/gpt -h
```

The three operators

- CawaTCWV.Meris
- CawaTCWV.Modis
- CawaCTP.Meris

should now appear in the listing of available SNAP operators, together with a short description.

The Pixel Classification Step

The pixel classification with IdePix is applied on the L1b input products. The operators for MERIS and MODIS

- Idepix.Meris
- Idepix.Modis

should also appear in the listing of available SNAP operators, together with a short description.

A more detailed information on the distinct operator can be obtained with

```
$SNAP_INSTALL_DIR/bin/gpt -h <operator-name>
```

e.g.

```
$SNAP_INSTALL_DIR/bin/gpt -h Idepix.Modis
```

(see Figure 3).

Processing Parameters

The gpt command given above shows the possible IdePix processing parameters. To ensure a correct workflow towards the TCWV and CTP processing, the following IdePix processing parameters must be set as they deviate from the default (Table 11):

Table 11: Processing parameters deviating from defaults for CAWA IdePix classification step.

Operator	Parameter	Value
Idepix.Meris (for TCWV)	reflBandsToCopy	reflectance_13,reflectance_14,reflectance_15
Idepix.Meris (for CTP)	none	
Idepix.Modis (for TCWV)	outputCawaRefSB	true

```
globalbedo@gamaster:~/od/cawa/snap-cawa-1.2 $ gpt -h Idepix.Modis
Usage:
  gpt Idepix.Modis [options]

Description:
  Pixel identification and classification for MODIS.

Source Options:
  -Ssource=<file>    The source product.
                     This is a mandatory source.

Parameter Options:
  -PcloudBufferWidth=<int>           Sets parameter 'cloudBufferWidth' to <int>.
                                     Default value is '1'.
  -PcloudFlaggingStrength=<string>   Strength of cloud flagging. In case of 'CLOUD_CONSERVATIVE', more pixels might be flagged as cloud.
                                     Value must be one of 'CLEAR_SKY_CONSERVATIVE', 'CLOUD_CONSERVATIVE'.
                                     Default value is 'CLEAR_SKY_CONSERVATIVE'.
  -PoutputEmissive=<boolean>         Write 'Emissive' bands to target product.
                                     Default value is 'false'.
  -PoutputRad2Refl=<boolean>         Write TOA reflective solar bands (RefSB) to target product.
                                     Default value is 'true'.
  -PwaterMaskResolution=<int>        Resolution of used land-water mask in meters per pixel
                                     Value must be one of '1000', '150', '50'.
                                     Default value is '150'.

Graph XML Format:
  <graph id="someGraphId">
    <version>1.0</version>
    <node id="someNodeId">
      <operator>Idepix.Modis</operator>
      <sources>
        <source>${source}</source>
      </sources>
      <parameters>
        <cloudFlaggingStrength>string</cloudFlaggingStrength>
        <cloudBufferWidth>int</cloudBufferWidth>
        <waterMaskResolution>int</waterMaskResolution>
        <outputRad2Refl>boolean</outputRad2Refl>
        <outputEmissive>boolean</outputEmissive>
      </parameters>
    </node>
  </graph>
```

Figure 3: GPF information for Idepix MODIS operator..

Applying these processing parameters, the calls for Idepix.Meris (TCWV, CTP) and Idepix.Modis (TCWV) would look like:

IdePix MERIS TCWV:

```
gpt Idepix.Meris -SsourceProduct=<path-to-MERIS-L1b-product>
-PreflBandsToCopy=reflectance_13,reflectance_14,reflectance_15
-f NetCDF4-BEAM -t <path-to-idepix-meris-for-tcwv-product>
```

IdePix MERIS CTP:

```
gpt Idepix.Meris -SsourceProduct=<path-to-MERIS-L1b-product>
-f NetCDF4-BEAM -t <path-to-idepix-meris-for-ctp-product>
```

IdePix MODIS TCWV:

```
gpt Idepix.Modis -SsourceProduct=<path-to-MERIS-L1b-product>
-PreflBandsToCopy=reflectance_13,reflectance_14,reflectance_15
-f NetCDF4-BEAM -t <path-to-idepix-meris-for-tcwv-product>
```

TCWV Processing

The TCWV processing is applied on the 'extended' IdePix products as described in *The SNAP CAWA Products*. The gpt command in Figure 5 and Figure 5 shows the possible TCWV processing parameters for MERIS and MODIS, respectively.

Processing Parameters

The operators for MERIS and MODIS

- CawaTCWV.Meris
- CawaTCWV.Modis

do not require any non-default parameters and are invoked via the SNAP gpt tool like:

```
<operator-name> -SsourceProduct=<path-to-IdePix-product> -f NetCDF4-CAWA -t
<path-to-target-product>
```

However, as discussed in *The SNAP CAWA Products*, it is strongly recommended to use IdePix 'extended' products containing ERA-Interim data which provides more realistic prior variables. If these are not available, the processor will use processor parameters for prior 2m temperature, mean sea level pressure and AOT. They can be explicitly provided by the user, so a TCWV processor call may look like:

```
CawaTCWV.Meris -SsourceProduct=<path-to-IdePix-product> -Ptemperature=285.0
-Ppressure=990.0 -Paot_13=0.25
-f NetCDF4-CAWA -t <path-to-target-product>
```

However, these values would be constant for all pixels of the given scene, so the resulting TCWV retrieval may be poor.

```
globalbedo@gamaster:~/od/cawa/snap-cawa-1.2 $ gpt -h CawaTCWV.Meris
Usage:
  gpt CawaTCWV.Meris [options]

Description:
  Operator for MERIS total column of water vapour (TCWV) retrieval as developed in CAWA project.


Source Options:
  -Ssource=<file>    Sets source 'source' to <filepath>.
                     This is a mandatory source.

Parameter Options:
  -Paot_13=<double>         A constant AOT at MERIS band 13 prior value.
                            Default value is '0.1'.
  -Ppressure=<double>       A constant mean sea level pressure (hPa) prior value. Used as fallback.
                            Default value is '1013.25'.
  -Ptemperature=<double>    A constant 2m temperature (K) prior value. Used as fallback.
                            Default value is '303.0'.

Graph XML Format:
  <graph id="someGraphId">
    <version>1.0</version>
    <node id="someNodeId">
      <operator>CawaTCWV.Meris</operator>
      <sources>
        <source>${source}</source>
      </sources>
      <parameters>
        <temperature>double</temperature>
        <pressure>double</pressure>
        <aot__13>double</aot__13>
      </parameters>
    </node>
  </graph>
```

Figure 4: GPF information for TCWV MERIS operator.

```
globalbedo@gamaster:~/od/cawa/snap-cawa-1.2 $ gpt -h CawaTCWV.Modis
Usage:
  gpt CawaTCWV.Modis [options]

Description:
  Operator for MODIS total column of water vapour (TCWV) retrieval as developed in CAWA project.


Source Options:
  -Ssource=<file>    Sets source 'source' to <filepath>.
                     This is a mandatory source.

Parameter Options:
  -Ppressure=<double>       A constant mean sea level pressure (hPa) prior value. Used as fallback.
                            Default value is '1003.0'.
  -Pprior_aot=<double>      A constant AOT prior value.
                            Default value is '0.1'.
  -Ptemperature=<double>    A constant 2m temperature (K) prior value. Used as fallback.
                            Default value is '303.0'.

Graph XML Format:
  <graph id="someGraphId">
    <version>1.0</version>
    <node id="someNodeId">
      <operator>CawaTCWV.Modis</operator>
      <sources>
        <source>${source}</source>
      </sources>
      <parameters>
        <temperature>double</temperature>
        <pressure>double</pressure>
        <prior__aot>double</prior__aot>
      </parameters>
    </node>
  </graph>
```

Figure 5: GPF information for TCWV MODIS operator.

CTP Processing

The CTP MERIS processing is applied on the 'extended' IdePix products as described in *The SNAP CAWA Products*. The gpt command in Figure 6 shows the possible CTP MERIS processing options.

```
globalbedo@gamaster:~/od/cawa/snap-cawa-1.2 $ gpt -h CawaCTP.Meris
Usage:
  gpt CawaCTP.Meris [options]

Description:
  Operator for MERIS cloud top pressure (CTP) retrieval as developed in CAWA project.


Source Options:
  -Ssource=<file>    Sets source 'source' to <filepath>.
                     This is a mandatory source.

Graph XML Format:
  <graph id="someGraphId">
    <version>1.0</version>
    <node id="someNodeId">
      <operator>CawaCTP.Meris</operator>
      <sources>
        <source>${source}</source>
      </sources>
      <parameters/>
    </node>
  </graph>
```

Figure 6: GPF information for TCWV MERIS operator.

Processing Parameters

The operators for MERIS

- CawaCTP.Meris

does not require any non-default parameters and is invoked via the SNAP gpt tool like:

```
CawaCTP.Meris -SsourceProduct=<path-to-IdePix-product> -f NetCDF4-CAWA
-t <path-to-target-product>
```

Data Analysis Tools

SNAP Desktop Application

The TCWV and CTP products generated within the CAWA project are provided in CFcompliant NetCDF-4/HDF5 format, which is supported by a variety of tools for further scientific analysis and processing. One of the important tools are the BEAM toolbox and its successor SNAP.

BEAM is the Basic ERS & Envisat (A)ATSR and MERIS Toolbox and is a collection of executable tools and an application programming interface (API) which had been developed to facilitate the use, viewing and processing of data of various sensors. However, it is more recommended to use the latest version of the SNAP toolboxes which do not only provide most of all existing BEAM functionalities

and product support, but also various new features as well as support for the new sensors onboard the Sentinel-x satellites. The SNAP desktop application is directly available after having installed SNAP as described in *Processing Software Installation*.

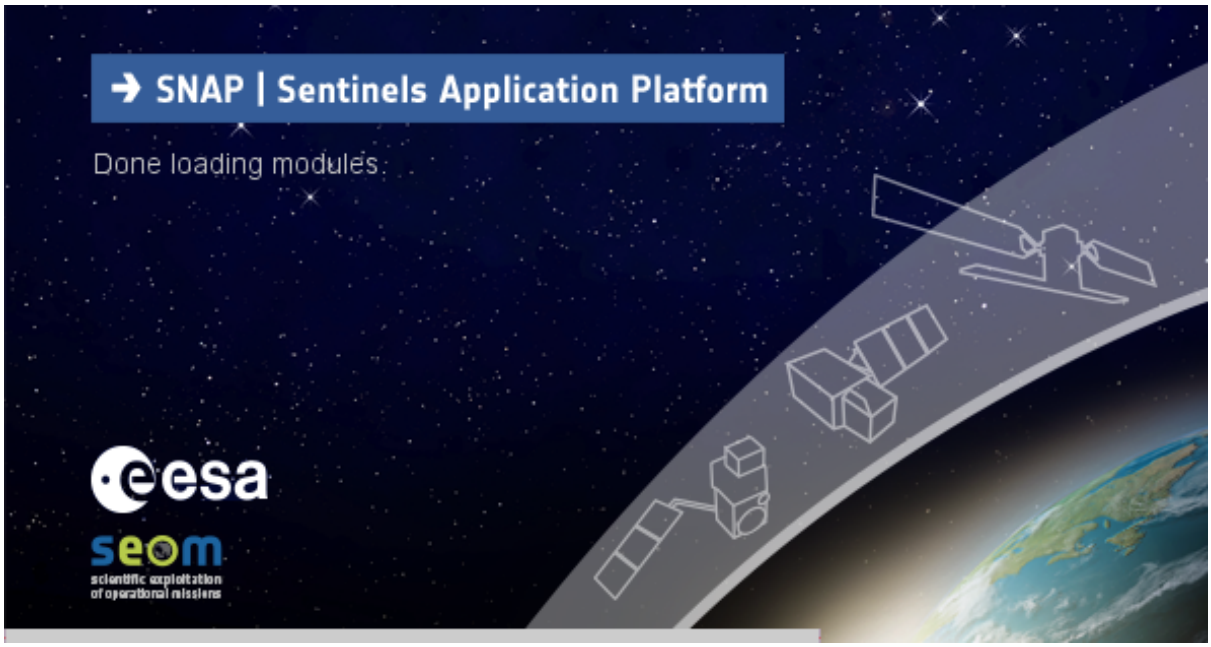

Figure 7: The SNAP desktop application splash screen.

Annex

Example of IdePix NetCDF4 product header:

```
netcdf L2_of_MER_RR__1PNACR20080621_055731_000001512069_00363_32982_0000 {
dimensions:
    y = 865 ;
    x = 1121 ;
    tp_y = 55 ;
    tp_x = 71 ;
variables:
    short cloud_classif_flags(y, x) ;
        cloud_classif_flags:coordinates = "lat lon" ;
        cloud_classif_flags:flag_meanings = "F_INVALID F_CLOUD
        F_CLOUD_AMBIGUOUS F_CLOUD_SURE F_CLOUD_BUFFER F_CLOUD_SHADOW
        F_SNOW_ICE F_GLINTRISK F_COASTLINE F_LAND" ;
        cloud_classif_flags:flag_masks = 1s, 2s, 4s, 8s, 16s, 32s, 64s,
        128s, 256s, 512s ;
        cloud_classif_flags:flag_coding_name = "cloud_classif_flags" ;
        cloud_classif_flags:flag_descriptions = "Invalid pixels\tPixels
        which are either cloud_sure or cloud_ambiguous\tSemi transparent
        clouds, or clouds where the detection level is uncertain\tFully
        opaque clouds with full confidence of their detection\tA buffer
        of n pixels around a cloud. n is a user supplied parameter. Applied
        to pixels masked as \'cloud\'\tPixels is affect by a cloud
        shadow\tSnow/ice pixels\tPixels with glint risk\tPixels at a
        coastline\tLand pixels" ;
        cloud_classif_flags:long_name = "" ;
    short radiance_10(y, x) ;
        radiance_10:long_name = "TOA radiance band 10" ;
        radiance_10:units = "mW/(m^2*sr*nm)" ;
        radiance_10:_Unsigned = "true" ;
        radiance_10:scale_factor = 0.00866463407874107 ;
        radiance_10:coordinates = "lat lon" ;
        radiance_10:bandwidth = 7.495f ;
        radiance_10:wavelength = 753.371f ;
        radiance_10:valid_pixel_expression = "!l1_flags.INVALID" ;
        radiance_10:solar_flux = 1227.051f ;
        radiance_10:spectral_band_index = 9.f ;
    short radiance_11(y, x) ;
        radiance_11:long_name = "TOA radiance band 11" ;
        radiance_11:units = "mW/(m^2*sr*nm)" ;
        radiance_11:_Unsigned = "true" ;
        radiance_11:scale_factor = 0.00887294951826334 ;
        radiance_11:coordinates = "lat lon" ;
        radiance_11:bandwidth = 3.744f ;
        radiance_11:wavelength = 761.5081f ;
        radiance_11:valid_pixel_expression = "!l1_flags.INVALID" ;
        radiance_11:solar_flux = 1215.942f ;
        radiance_11:spectral_band_index = 10.f ;
    short detector_index(y, x) ;
        detector_index:coordinates = "lat lon" ;
        detector_index:long_name = "Detector index" ;
    byte l1_flags(y, x) ;
        l1_flags:_Unsigned = "true" ;
        l1_flags:coordinates = "lat lon" ;
        l1_flags:flag_meanings = "COSMETIC DUPLICATED GLINT_RISK SUSPECT
```

```
        LAND_OCEAN BRIGHT COASTLINE INVALID" ;
        l1_flags:flag_masks = 1b, 2b, 4b, 8b, 16b, 32b, 64b, -128b ;
        l1_flags:flag_coding_name = "l1_flags" ;
        l1_flags:flag_descriptions = "Pixel is cosmetic\tPixel has been
        duplicated (filled in)\tPixel has glint risk\tPixel is
        suspect\tPixel is over land, not ocean\tPixel is bright\tPixel
        is part of acoastline\tPixel is invalid" ;
        l1_flags:long_name = "Level 1b classification and quality flags" ;
    float latitude(tp_y, tp_x) ;
        latitude:offset_y = 0.5f ;
        latitude:subsampling_x = 16.f ;
        latitude:subsampling_y = 16.f ;
        latitude:offset_x = 0.5f ;
    float longitude(tp_y, tp_x) ;
        longitude:offset_y = 0.5f ;
        longitude:subsampling_x = 16.f ;
        longitude:subsampling_y = 16.f ;
        longitude:offset_x = 0.5f ;
    float dem_alt(tp_y, tp_x) ;
        dem_alt:offset_y = 0.5f ;
        dem_alt:subsampling_x = 16.f ;
        dem_alt:subsampling_y = 16.f ;
        dem_alt:offset_x = 0.5f ;
    float dem_rough(tp_y, tp_x) ;
        dem_rough:offset_y = 0.5f ;
        dem_rough:subsampling_x = 16.f ;
        dem_rough:subsampling_y = 16.f ;
        dem_rough:offset_x = 0.5f ;
    float lat_corr(tp_y, tp_x) ;
        lat_corr:offset_y = 0.5f ;
        lat_corr:subsampling_x = 16.f ;
        lat_corr:subsampling_y = 16.f ;
        lat_corr:offset_x = 0.5f ;
    float lon_corr(tp_y, tp_x) ;
        lon_corr:offset_y = 0.5f ;
        lon_corr:subsampling_x = 16.f ;
        lon_corr:subsampling_y = 16.f ;
        lon_corr:offset_x = 0.5f ;
    float sun_zenith(tp_y, tp_x) ;
        sun_zenith:offset_y = 0.5f ;
        sun_zenith:subsampling_x = 16.f ;
        sun_zenith:subsampling_y = 16.f ;
        sun_zenith:offset_x = 0.5f ;
    float sun_azimuth(tp_y, tp_x) ;
        sun_azimuth:offset_y = 0.5f ;
        sun_azimuth:subsampling_x = 16.f ;
        sun_azimuth:subsampling_y = 16.f ;
        sun_azimuth:offset_x = 0.5f ;
    float view_zenith(tp_y, tp_x) ;
        view_zenith:offset_y = 0.5f ;
        view_zenith:subsampling_x = 16.f ;
        view_zenith:subsampling_y = 16.f ;
        view_zenith:offset_x = 0.5f ;
    float view_azimuth(tp_y, tp_x) ;
        view_azimuth:offset_y = 0.5f ;
        view_azimuth:subsampling_x = 16.f ;
        view_azimuth:subsampling_y = 16.f ;
```

```
        view_azimuth:offset_x = 0.5f ;
    float zonal_wind(tp_y, tp_x) ;
        zonal_wind:offset_y = 0.5f ;
        zonal_wind:subsampling_x = 16.f ;
        zonal_wind:subsampling_y = 16.f ;
        zonal_wind:offset_x = 0.5f ;
    float merid_wind(tp_y, tp_x) ;
        merid_wind:offset_y = 0.5f ;
        merid_wind:subsampling_x = 16.f ;
        merid_wind:subsampling_y = 16.f ;
        merid_wind:offset_x = 0.5f ;
    float atm_press(tp_y, tp_x) ;
        atm_press:offset_y = 0.5f ;
        atm_press:subsampling_x = 16.f ;
        atm_press:subsampling_y = 16.f ;
        atm_press:offset_x = 0.5f ;
    float ozone(tp_y, tp_x) ;
        ozone:offset_y = 0.5f ;
        ozone:subsampling_x = 16.f ;
        ozone:subsampling_y = 16.f ;
        ozone:offset_x = 0.5f ;
    float rel_hum(tp_y, tp_x) ;
        rel_hum:offset_y = 0.5f ;
        rel_hum:subsampling_x = 16.f ;
        rel_hum:subsampling_y = 16.f ;
        rel_hum:offset_x = 0.5f ;
    float lat(y, x) ;
        lat:long_name = "latitude coordinate" ;
        lat:standard_name = "latitude" ;
        lat:units = "degrees_north" ;
    float lon(y, x) ;
        lon:long_name = "longitude coordinate" ;
        lon:standard_name = "longitude" ;
        lon:units = "degrees_east" ;
    byte cawa_invalid_mask ;
        cawa_invalid_mask:expression = "cloud_classif_flags.F_INVALID" ;
        cawa_invalid_mask:color = 178, 0, 0, 255 ;
        cawa_invalid_mask:transparency = 0.5 ;
        cawa_invalid_mask:title = "Invalid pixels" ;
    byte cawa_cloud_mask ;
        cawa_cloud_mask:expression = "cloud_classif_flags.F_CLOUD" ;
        cawa_cloud_mask:color = 255, 0, 255, 255 ;
        cawa_cloud_mask:transparency = 0.5 ;
        cawa_cloud_mask:title = "Pixels which are either cloud_sure or
        cloud_ambiguous" ;
    byte cawa_cloud_ambiguous_mask ;
        cawa_cloud_ambiguous_mask:expression =
        "cloud_classif_flags.F_CLOUD_AMBIGUOUS" ;
        cawa_cloud_ambiguous_mask:color = 255, 255, 0, 255 ;
        cawa_cloud_ambiguous_mask:transparency = 0.5 ;
        cawa_cloud_ambiguous_mask:title = "Semi transparent clouds, or
        clouds where the detection level  is uncertain" ;
    byte cawa_cloud_sure_mask ;
        cawa_cloud_sure_mask:expression =
        "cloud_classif_flags.F_CLOUD_SURE" ;
        cawa_cloud_sure_mask:color = 255, 0, 0, 255 ;
        cawa_cloud_sure_mask:transparency = 0.5 ;
```

```
        cawa_cloud_sure_mask:title = "Fully opaque clouds with full
        confidence of their detection" ;
    byte cawa_cloud_buffer_mask ;
        cawa_cloud_buffer_mask:expression =
        "cloud_classif_flags.F_CLOUD_BUFFER" ;
        cawa_cloud_buffer_mask:color = 255, 200, 0, 255 ;
        cawa_cloud_buffer_mask:transparency = 0.5 ;
        cawa_cloud_buffer_mask:title = "A buffer of n pixels around a
        cloud. n is a user supplied parameter.
        Applied to pixels masked as \'cloud\'" ;
    byte cawa_cloud_shadow_mask ;
        cawa_cloud_shadow_mask:expression =
        "cloud_classif_flags.F_CLOUD_SHADOW" ;
        cawa_cloud_shadow_mask:color = 178, 0, 0, 255 ;
        cawa_cloud_shadow_mask:transparency = 0.5 ;
        cawa_cloud_shadow_mask:title =
        "Pixels is affect by a cloud shadow" ;
    byte cawa_snow_ice_mask ;
        cawa_snow_ice_mask:expression = "cloud_classif_flags.F_SNOW_ICE" ;
        cawa_snow_ice_mask:color = 0, 255, 255, 255 ;
        cawa_snow_ice_mask:transparency = 0.5 ;
        cawa_snow_ice_mask:title = "Snow/ice pixels" ;
    byte cawa_glint_risk_mask ;
        cawa_glint_risk_mask:expression =
        "cloud_classif_flags.F_GLINTRISK" ;
        cawa_glint_risk_mask:color = 255, 175, 175, 255 ;
        cawa_glint_risk_mask:transparency = 0.5 ;
        cawa_glint_risk_mask:title = "Pixels with glint risk" ;
    byte cawa_coastline_mask ;
        cawa_coastline_mask:expression = "cloud_classif_flags.F_COASTLINE"␣
↪;
        cawa_coastline_mask:color = 0, 178, 0, 255 ;
        cawa_coastline_mask:transparency = 0.5 ;
        cawa_coastline_mask:title = "Pixels at a coastline" ;
    byte cawa_land_mask ;
        cawa_land_mask:expression = "cloud_classif_flags.F_LAND" ;
        cawa_land_mask:color = 0, 255, 0, 255 ;
        cawa_land_mask:transparency = 0.5 ;
        cawa_land_mask:title = "Land pixels" ;
    byte coastline_mask ;
        coastline_mask:expression = "l1_flags.COASTLINE" ;
        coastline_mask:color = 0, 255, 0, 255 ;
        coastline_mask:transparency = 0. ;
        coastline_mask:title = "Pixel is part of a coastline" ;
    byte land_mask ;
        land_mask:expression = "l1_flags.LAND_OCEAN" ;
        land_mask:color = 51, 153, 0, 255 ;
        land_mask:transparency = 0.75 ;
        land_mask:title = "Pixel is over land, not ocean" ;
    byte water_mask ;
        water_mask:expression = "NOT l1_flags.LAND_OCEAN" ;
        water_mask:color = 153, 153, 255, 255 ;
        water_mask:transparency = 0.75 ;
        water_mask:title = "Not Pixel is over land, not ocean" ;
    byte cosmetic_mask ;
        cosmetic_mask:expression = "l1_flags.COSMETIC" ;
        cosmetic_mask:color = 204, 153, 255, 255 ;
```

```
        cosmetic_mask:transparency = 0.5 ;
        cosmetic_mask:title = "Pixel is cosmetic" ;
    byte duplicated_mask ;
        duplicated_mask:expression = "l1_flags.DUPLICATED" ;
        duplicated_mask:color = 255, 200, 0, 255 ;
        duplicated_mask:transparency = 0.5 ;
        duplicated_mask:title = "Pixel has been duplicated (filled in)" ;
    byte glint_risk_mask ;
        glint_risk_mask:expression = "l1_flags.GLINT_RISK" ;
        glint_risk_mask:color = 255, 0, 255, 255 ;
        glint_risk_mask:transparency = 0.5 ;
        glint_risk_mask:title = "Pixel has glint risk" ;
    byte suspect_mask ;
        suspect_mask:expression = "l1_flags.SUSPECT" ;
        suspect_mask:color = 204, 102, 255, 255 ;
        suspect_mask:transparency = 0.5 ;
        suspect_mask:title = "Pixel is suspect" ;
    byte bright_mask ;
        bright_mask:expression = "l1_flags.BRIGHT" ;
        bright_mask:color = 255, 255, 0, 255 ;
        bright_mask:transparency = 0.5 ;
        bright_mask:title = "Pixel is bright" ;
    byte invalid_mask ;
        invalid_mask:expression = "l1_flags.INVALID" ;
        invalid_mask:color = 255, 0, 0, 255 ;
        invalid_mask:transparency = 0. ;
        invalid_mask:title = "Pixel is invalid" ;

// global attributes:
        :Conventions = "CF-1.4" ;
        :TileSize = "16:1121" ;
        :product_type = "mergedClassif" ;
        :metadata_profile = "beam" ;
        :metadata_version = "0.5" ;
        :auto_grouping = "radiance:rho_toa" ;
        :tiepoint_coordinates = "longitude latitude" ;
        :start_date = "21-JUN-2008 05:57:31.155941" ;
        :stop_date = "21-JUN-2008 06:00:03.209572" ;
}
```

Example of CAWA TCWV product header:

```
netcdf L2_of_L2_of_MER_RR__1PNUPA20060102_141100_000026182043_00497_20090_
↪7596 {
dimensions:
    y = 14881 ;
    x = 1121 ;
    tp_y = 931 ;
    tp_x = 71 ;
variables:
    float tcwv(y, x) ;
        tcwv:units = "mm" ;
        tcwv:_FillValue = -999.f ;
        tcwv:long_name = "Total column of water vapour" ;
    byte tcwv_flags(y, x) ;
        tcwv_flags:units = "1" ;
        tcwv_flags:long_name = "TCWV flags band" ;
```

```
    short cloud_classif_flags(y, x) ;
        cloud_classif_flags:units = "1" ;
        cloud_classif_flags:flag_meanings = "F_INVALID F_CLOUD
        F_CLOUD_AMBIGUOUS F_CLOUD_SURE F_CLOUD_BUFFER F_CLOUD_SHADOW
        F_SNOW_ICE F_GLINTRISK F_COASTLINE F_LAND" ;
        cloud_classif_flags:flag_masks = 1s, 2s, 4s, 8s, 16s, 32s, 64s,
        128s, 256s, 512s ;
        cloud_classif_flags:flag_coding_name = "cloud_classif_flags" ;
        cloud_classif_flags:flag_descriptions = "Invalid pixels\tPixels
        which are either cloud_sure or cloud_ambiguous\tSemi transparent
        clouds, or clouds where the detection level is uncertain\tFully
        opaque clouds with full confidence of their detection\tA buffer
        of n pixels around a cloud. n is a user supplied parameter. Applied
        to pixels masked as \'cloud\'\tPixels is affect by a cloud
        shadow\tSnow/ice pixels\tPixels with glint risk\tPixels at a
        coastline\tLand pixels" ;
        cloud_classif_flags:long_name = "" ;
    float latitude(tp_y, tp_x) ;
        latitude:offset_y = 0.5 ;
        latitude:subsampling_x = 16. ;
        latitude:subsampling_y = 16. ;
        latitude:units = "degree" ;
        latitude:standard_name = "latitude" ;
        latitude:offset_x = 0.5 ;
    float longitude(tp_y, tp_x) ;
        longitude:offset_y = 0.5 ;
        longitude:subsampling_x = 16. ;
        longitude:subsampling_y = 16. ;
        longitude:units = "degree" ;
        longitude:standard_name = "longitude" ;
        longitude:offset_x = 0.5 ;
    byte cawa_invalid_mask ;
        cawa_invalid_mask:description = "Invalid pixels" ;
        cawa_invalid_mask:expression = "cloud_classif_flags.F_INVALID" ;
        cawa_invalid_mask:color = 178, 0, 0, 255 ;
        cawa_invalid_mask:transparency = 0.5 ;
        cawa_invalid_mask:long_name = "cawa_invalid" ;
    byte cawa_cloud_mask ;
        cawa_cloud_mask:description = "Pixels which are either cloud_sure
        or cloud_ambiguous" ;
        cawa_cloud_mask:expression = "cloud_classif_flags.F_CLOUD" ;
        cawa_cloud_mask:color = 255, 0, 255, 255 ;
        cawa_cloud_mask:transparency = 0.5 ;
        cawa_cloud_mask:long_name = "cawa_cloud" ;
    byte cawa_cloud_ambiguous_mask ;
        cawa_cloud_ambiguous_mask:description = "Semi transparent clouds,
        or clouds where the detection level is uncertain" ;
        cawa_cloud_ambiguous_mask:expression =
        "cloud_classif_flags.F_CLOUD_AMBIGUOUS" ;
        cawa_cloud_ambiguous_mask:color = 255, 255, 0, 255 ;
        cawa_cloud_ambiguous_mask:transparency = 0.5 ;
        cawa_cloud_ambiguous_mask:long_name = "cawa_cloud_ambiguous" ;
    byte cawa_cloud_sure_mask ;
        cawa_cloud_sure_mask:description =
        "Fully opaque clouds with full confidence of their detection" ;
        cawa_cloud_sure_mask:expression = "cloud_classif_flags.F_CLOUD_SURE
↪" ;
```

```
        cawa_cloud_sure_mask:color = 255, 0, 0, 255 ;
        cawa_cloud_sure_mask:transparency = 0.5 ;
        cawa_cloud_sure_mask:long_name = "cawa_cloud_sure" ;
    byte cawa_cloud_buffer_mask ;
        cawa_cloud_buffer_mask:description = "A buffer of n pixels around
        a cloud. n is a user supplied parameter. Applied to pixels masked
        as \'cloud\'" ;
        cawa_cloud_buffer_mask:expression =
        "cloud_classif_flags.F_CLOUD_BUFFER" ;
        cawa_cloud_buffer_mask:color = 255, 200, 0, 255 ;
        cawa_cloud_buffer_mask:transparency = 0.5 ;
        cawa_cloud_buffer_mask:long_name = "cawa_cloud_buffer" ;
    byte cawa_cloud_shadow_mask ;
        cawa_cloud_shadow_mask:description = "Pixels is affect by a
        cloud shadow" ;
        cawa_cloud_shadow_mask:expression =
        "cloud_classif_flags.F_CLOUD_SHADOW" ;
        cawa_cloud_shadow_mask:color = 178, 0, 0, 255 ;
        cawa_cloud_shadow_mask:transparency = 0.5 ;
        cawa_cloud_shadow_mask:long_name = "cawa_cloud_shadow" ;
    byte cawa_snow_ice_mask ;
        cawa_snow_ice_mask:description = "Snow/ice pixels" ;
        cawa_snow_ice_mask:expression = "cloud_classif_flags.F_SNOW_ICE" ;
        cawa_snow_ice_mask:color = 0, 255, 255, 255 ;
        cawa_snow_ice_mask:transparency = 0.5 ;
        cawa_snow_ice_mask:long_name = "cawa_snow_ice" ;
    byte cawa_glint_risk_mask ;
        cawa_glint_risk_mask:description = "Pixels with glint risk" ;
        cawa_glint_risk_mask:expression = "cloud_classif_flags.F_GLINTRISK
↪" ;
        cawa_glint_risk_mask:color = 255, 175, 175, 255 ;
        cawa_glint_risk_mask:transparency = 0.5 ;
        cawa_glint_risk_mask:long_name = "cawa_glint_risk" ;
    byte cawa_coastline_mask ;
        cawa_coastline_mask:description = "Pixels at a coastline" ;
        cawa_coastline_mask:expression = "cloud_classif_flags.F_COASTLINE"␣
↪;
        cawa_coastline_mask:color = 0, 178, 0, 255 ;
        cawa_coastline_mask:transparency = 0.5 ;
        cawa_coastline_mask:long_name = "cawa_coastline" ;
    byte cawa_land_mask ;
        cawa_land_mask:description = "Land pixels" ;
        cawa_land_mask:expression = "cloud_classif_flags.F_LAND" ;
        cawa_land_mask:color = 0, 255, 0, 255 ;
        cawa_land_mask:transparency = 0.5 ;
        cawa_land_mask:long_name = "cawa_land" ;

// global attributes:
        :Conventions = "CF-1.4" ;
        :title = "CAWA TCWV product" ;
        :product_type = "CAWA TCWV" ;
        :start_date = "02-JAN-2006 14:11:00.727666" ;
        :stop_date = "02-JAN-2006 14:54:39.429106" ;
        :TileSize = "64:1121" ;
        :metadata_profile = "beam" ;
        :metadata_version = "0.5" ;
        :tiepoint_coordinates = "longitude latitude" ;
```

```
}
```

Example of CAWA CTP product header:

```
netcdf L2_of_L2_of_MER_RR__1PNUPA20050701_072830_000026412038_00350_17438_
↪5743 {
dimensions:
    y = 15009 ;
    x = 1121 ;
    tp_y = 939 ;
    tp_x = 71 ;
variables:
    float ctp(y, x) ;
        ctp:units = "hPa" ;
        ctp:_FillValue = -999.f ;
        ctp:long_name = "Cloud Top Pressure" ;
    byte ctp_flags(y, x) ;
        ctp_flags:units = "1" ;
        ctp_flags:long_name = "CTP flags band" ;
    short cloud_classif_flags(y, x) ;
        cloud_classif_flags:units = "1" ;
        cloud_classif_flags:flag_meanings = "F_INVALID F_CLOUD
        F_CLOUD_AMBIGUOUS F_CLOUD_SURE F_CLOUD_BUFFER F_CLOUD_SHADOW
        F_SNOW_ICE F_GLINTRISK F_COASTLINE F_LAND" ;
        cloud_classif_flags:flag_masks = 1s, 2s, 4s, 8s, 16s, 32s, 64s,
        128s, 256s, 512s ;
        cloud_classif_flags:flag_coding_name = "cloud_classif_flags" ;
        cloud_classif_flags:flag_descriptions = "Invalid pixels\tPixels
        which are either cloud_sure or cloud_ambiguous\tSemi transparent
        clouds, or clouds where the detection level is uncertain\tFully
        opaque clouds with full confidence of their detection\tA buffer
        of n pixels around a cloud. n is a user supplied parameter.
        Applied to pixels masked as \'cloud\'\tPixels is affect by a cloud
        shadow\tSnow/ice pixels\tPixels with glint risk\tPixels at a
        coastline\tLand pixels" ;
        cloud_classif_flags:long_name = "" ;
    float latitude(tp_y, tp_x) ;
        latitude:offset_y = 0.5 ;
        latitude:subsampling_x = 16. ;
        latitude:subsampling_y = 16. ;
        latitude:units = "degree" ;
        latitude:standard_name = "latitude" ;
        latitude:offset_x = 0.5 ;
    float longitude(tp_y, tp_x) ;
        longitude:offset_y = 0.5 ;
        longitude:subsampling_x = 16. ;
        longitude:subsampling_y = 16. ;
        longitude:units = "degree" ;
        longitude:standard_name = "longitude" ;
        longitude:offset_x = 0.5 ;
    byte cawa_invalid_mask ;
        cawa_invalid_mask:description = "Invalid pixels" ;
        cawa_invalid_mask:expression = "cloud_classif_flags.F_INVALID" ;
        cawa_invalid_mask:color = 178, 0, 0, 255 ;
        cawa_invalid_mask:transparency = 0.5 ;
        cawa_invalid_mask:long_name = "cawa_invalid" ;
    byte cawa_cloud_mask ;
```

```
        cawa_cloud_mask:description = "Pixels which are either cloud_sure
        or cloud_ambiguous" ;
        cawa_cloud_mask:expression = "cloud_classif_flags.F_CLOUD" ;
        cawa_cloud_mask:color = 255, 0, 255, 255 ;
        cawa_cloud_mask:transparency = 0.5 ;
        cawa_cloud_mask:long_name = "cawa_cloud" ;
    byte cawa_cloud_ambiguous_mask ;
        cawa_cloud_ambiguous_mask:description = "Semi transparent clouds,
        or clouds where the detection level is uncertain" ;
        cawa_cloud_ambiguous_mask:expression =
        "cloud_classif_flags.F_CLOUD_AMBIGUOUS" ;
        cawa_cloud_ambiguous_mask:color = 255, 255, 0, 255 ;
        cawa_cloud_ambiguous_mask:transparency = 0.5 ;
        cawa_cloud_ambiguous_mask:long_name = "cawa_cloud_ambiguous" ;
    byte cawa_cloud_sure_mask ;
        cawa_cloud_sure_mask:description = "Fully opaque clouds with full
        confidence of their detection" ;
        cawa_cloud_sure_mask:expression = "
        cloud_classif_flags.F_CLOUD_SURE" ;
        cawa_cloud_sure_mask:color = 255, 0, 0, 255 ;
        cawa_cloud_sure_mask:transparency = 0.5 ;
        cawa_cloud_sure_mask:long_name = "cawa_cloud_sure" ;
    byte cawa_cloud_buffer_mask ;
        cawa_cloud_buffer_mask:description = "A buffer of n pixels around
        a cloud. n is a user supplied parameter. Applied to pixels masked
        as \'cloud\'" ;
        cawa_cloud_buffer_mask:expression =
        "cloud_classif_flags.F_CLOUD_BUFFER" ;
        cawa_cloud_buffer_mask:color = 255, 200, 0, 255 ;
        cawa_cloud_buffer_mask:transparency = 0.5 ;
        cawa_cloud_buffer_mask:long_name = "cawa_cloud_buffer" ;
    byte cawa_cloud_shadow_mask ;
        cawa_cloud_shadow_mask:description = "Pixels is affect by a
        cloud shadow" ;
        cawa_cloud_shadow_mask:expression =
        "cloud_classif_flags.F_CLOUD_SHADOW" ;
        cawa_cloud_shadow_mask:color = 178, 0, 0, 255 ;
        cawa_cloud_shadow_mask:transparency = 0.5 ;
        cawa_cloud_shadow_mask:long_name = "cawa_cloud_shadow" ;
    byte cawa_snow_ice_mask ;
        cawa_snow_ice_mask:description = "Snow/ice pixels" ;
        cawa_snow_ice_mask:expression = "cloud_classif_flags.F_SNOW_ICE" ;
        cawa_snow_ice_mask:color = 0, 255, 255, 255 ;
        cawa_snow_ice_mask:transparency = 0.5 ;
        cawa_snow_ice_mask:long_name = "cawa_snow_ice" ;
    byte cawa_glint_risk_mask ;
        cawa_glint_risk_mask:description = "Pixels with glint risk" ;
        cawa_glint_risk_mask:expression =
        "cloud_classif_flags.F_GLINTRISK" ;
        cawa_glint_risk_mask:color = 255, 175, 175, 255 ;
        cawa_glint_risk_mask:transparency = 0.5 ;
        cawa_glint_risk_mask:long_name = "cawa_glint_risk" ;
    byte cawa_coastline_mask ;
        cawa_coastline_mask:description = "Pixels at a coastline" ;
        cawa_coastline_mask:expression =
        "cloud_classif_flags.F_COASTLINE" ;
        cawa_coastline_mask:color = 0, 178, 0, 255 ;
```

```
        cawa_coastline_mask:transparency = 0.5 ;
        cawa_coastline_mask:long_name = "cawa_coastline" ;
    byte cawa_land_mask ;
        cawa_land_mask:description = "Land pixels" ;
        cawa_land_mask:expression = "cloud_classif_flags.F_LAND" ;
        cawa_land_mask:color = 0, 255, 0, 255 ;
        cawa_land_mask:transparency = 0.5 ;
        cawa_land_mask:long_name = "cawa_land" ;

// global attributes:
        :Conventions = "CF-1.4" ;
        :title = "CAWA product" ;
        :product_type = "CAWA CTP" ;
        :start_date = "01-JUL-2005 07:28:30.062937" ;
        :stop_date = "01-JUL-2005 08:12:31.290841" ;
        :TileSize = "64:1121" ;
        :metadata_profile = "beam" ;
        :metadata_version = "0.5" ;
        :tiepoint_coordinates = "longitude latitude" ;
}
```

List of Figures

List of Tables