EUMETNET OPERA weather radar information model for implementation with the HDF5 file format

Version 2.0

Daniel B. Michelson[1], Rafał Lewandowski[2], Maciej Szewczykowski[2], and Hans Beekhuis[3]

[1]Swedish Meteorological and Hydrological Institute, Norrköping, Sweden
[2]Institute of Meteorology and Water Management, Warsaw, Poland
[3]Royal Netherlands Meteorological Institute, De Bilt, Netherlands
on behalf of EUMETNET OPERA

OPERA Working Document WD_2008_03

June 9, 2009

Abstract

This document specifies an information model with which the encoding, decoding and management of data and products from weather radar systems may be facilitated, primarily for the purposes of international exchange in Europe. An implementation of this information model is also specified which makes use of the HDF5 file format developed and maintained by the HDF Group. The result manifests itself in the form of truly self-describing weather radar data files highly suitable for environments where data exchange between radars from different manufacturers, different organizations, and/or different countries is conducted. The ability to include quality information, in the forms of metadata and binary arrays, is included in a powerful and flexible manner. This information model constitutes an official second-generation European standard exchange format for weather radar datasets. Because the NetCDF file format is built on HDF5, we also try to ensure that our information model will be compliant with NetCDF.

RELEASE NOTES

Version 2.0, 9 June 2009

A few inconsistancies between metadata in the tables and the diagrams in the UML representation have been identified and corrected. These fixes do not motivate a version number increment.

Version 2.0, 1 June 2009

This is the first version of ODIM_H5 to be officially accepted as an OPERA standard. Notwithstanding this status, the official parts are the definitions of polar scans and volumes, along with all associated metadata. All other objects restain their draft status, and will achieve official status in due course, subject to approval in OPERA.

Contents

List of Tables

1 Introduction and motivation

The goal of OPERA work package 2.1 “HDF5 exchange software development” is to formulate a second-generation information model for use with weather radar data and the Hierarchical Data Format version 5 (HDF5) file format. This information model will then be implemented in operational software.

This document presents an information model developed for use with weather radar data and products. Its implementation is also presented, which makes use of the HDF5 file format. HDF5 is developed and maintained by the HDF Group. All references to attributes, data, types, and so on, are in relation to those defined and used in the HDF5 documentation. The official HDF5 format documentation should be consulted for details on this format. This information model is an elaboration of the model presented by COST 717 and used in real-time operations in the Nordic countries[1]. Several enhancements have been made based on operational experience, and based on data quality issues addressed in OPERA II and the EU Voltaire project. The model presented here is not backwardly compatible with previous versions, but the advantages of its new structure outweigh this disadvantage.

The information model given here is designed from the point of view of trying to harmonize all relevant information independently of the radar manufacturer and organization from which the data originates. While the information model is intended to enable the representation of the data and products agreed upon today within the framework of EUMETNET OPERA, we also look ahead to future needs and have tried to ensure that they are appropriately met with this information model. This means that the known products are supported, but that it is easy to add forthcoming information types like polarization diversity variables and virtually any quality-related information characterizing a given dataset. It is also vital to recognize the importance of being able to represent polar (spherical coordinate space) data, and this is accomodated as well in a flexible manner.

This information model and use of HDF5 are not intended to compete with existing standards used in meteorology. Instead, we stress that this new standard primarily should be used in forthcoming meteorological standard exchange mechanisms, ie. the WMO Information System (WIS) and its infrastructure. There is also an important link to OPERA’s operational data centre (ODC), in that quality information in the information model will become available to the ODC, thus providing the potential for improving the quality of the operational products covering the European continent. This new standard is compliant with Open Geospatial Consortium (OGC) metadata standards, which means that the metadata accompanying the weather radar datasets are mappable to international *de facto* standards for management of environmental data based on OGC software. Examples of such systems are the basis of the Global Monitoring for Environment and Security (GMES a k a KOPERNIKUS) initiative which is the primary European contribution to the Global Earth Observation System of Systems (GEOSS). Care has also been taken to ensure that this new information model will be compliant with the EU INSPIRE Directive.

In this way, we have prepared future weather radar exchange for future data exchange infrastructure, and have tried to ensure that the use of the new technology will facilitate access to and use of European radar data both within and outside the meteorological community.

[1] Michelson D.B., Holleman I., Hohti H., and Salomonsen M., 2003: HDF5 information model and implementation for weather radar data. COST 717 working document WDF_02_200204_1. version 1.2.

2 Information model concept

The information model attempts to achieve a general-purpose model for storing both individual scans, images and products, while also allowing series (time and/or space) of these types of information to be stored using the same building-blocks. The heirarchical nature of HDF5 can be described as being similar to directories, files, and links on a hard-drive. Actual metadata are stored as so-called “attributes”, and these attributes are organized together in so-called “groups”. Binary data are stored as so-called “datasets”. The challenge in formulating an information model is in defining the way in which these general building-blocks are organized. This section illustrates the information model defined in detail later in this document, in an attempt to present and clarify its structure.

The first example is given in Figure 1, which shows how a simple cartesian product, for example a CAPPI, is represented. At the top level, the HDF5 file contains the so-called “root” group. This group is always there. Following that, three groups contain metadata; these are called “what” (object, information model version, and date/time information), “where” (geographical information), and “how” (quality and optional/recommended metadata). The data is organized in a group called “dataset1” which contains another group called “data1” where the actual binary data are found in “data”. This organization may seem overly complicated at first, but we will see shortly how powerful and necessary this organization is.

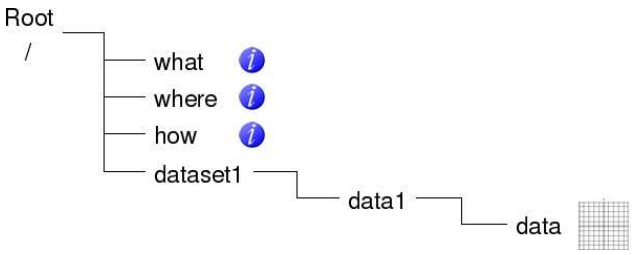

Figure 1: Cartesian product.

In terms of the analogy with a file system on a hard-disk, the HDF5 file containing this simple cartesian product is organized like this:

```
/
/what
/where
/how
/dataset1
/dataset1/data1
/dataset1/data1/data
```

In Figure 2, the exact same structure is used to represent a polar scan of data.

When we use this structure to represent a complete scan of data containing three parameters, the seemingly complicated organization used to store the binary data becomes easier to understand. Figure 3 shows how “dataset1” now contains three “data” groups, each containing a binary array of data. What we also see is the way in which the metadata groups “what” and “where” can be used recursively to hold metadata which are unique to their place in the hierarchy. In this case, “where” in “dataset1” contains the elevation angle of the scan used to collect the three parameters, and the three local “what” groups contain the information on which parameter is stored in each. In this way, “dataset1” can contain Z, V, and W in this case, but it can also contain an arbitrary number of other parameters.

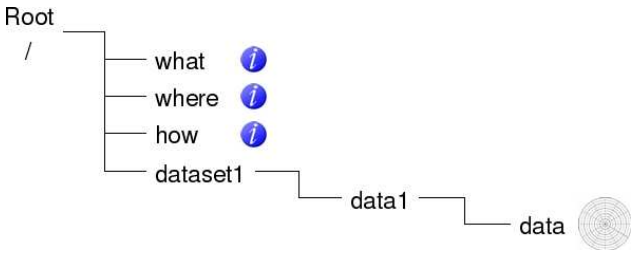

Figure 2: Simple polar scan.

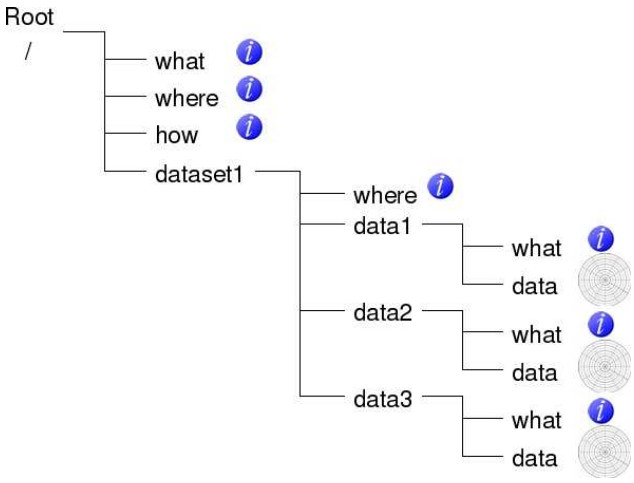

Figure 3: A complete polar scan containing three parameters.

The use of dataset-specific metadata groups is illustrated in Figure 4. Both methods of using “what”, “where” and “how” are acceptable. In the example on the left, the metadata groups contain attributes which are valid for every dataset in that group. In the case on the right, the metadata groups contain attributes which are valid only for that single dataset. In Figure 3, “where” is valid for all three datasets in the group, whereas “what” is only valid locally. The local metadata always have top priority, so if a mistake is made where a file contains the same metadata at different levels, the most local level will always take precedence.

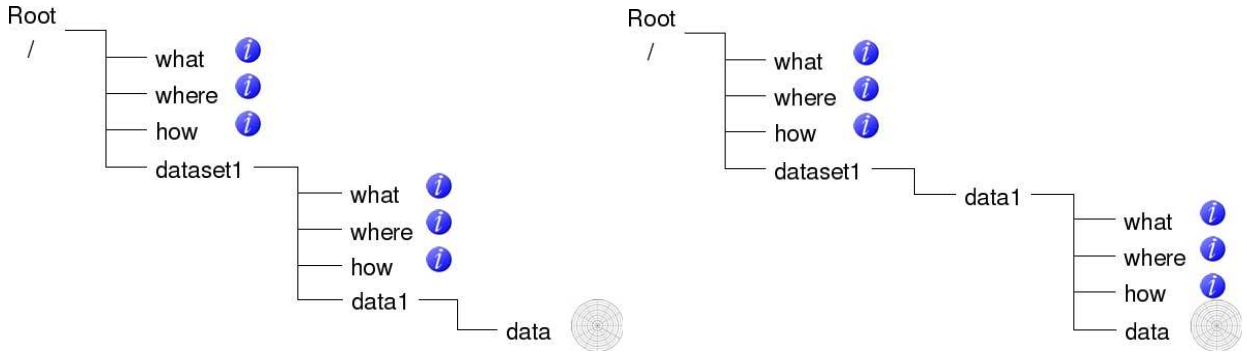

Figure 4: Use of dataset-specific metadata groups.

Continuing from the single scan in Figure 3, we can easily extend the same structure every time we want to add a new scan. This is illustrated in Figure 5. Here, the new elevation angle will be stored in `/dataset2/where`, and the information on the parameters stored in the datasets are found in each local

"what". Note that optional metadata can be added in a "how" group either directly under "dataset2" or together with each parameter, but this hasn't been included in this example. A complete volume containing an arbitrary number of scans, each containing an arbitrary number of parameters, is organized using this structure. A cartesian volume can be constructed in exactly the same way. Time series of polar or cartesian products can be constructed in the same way too.

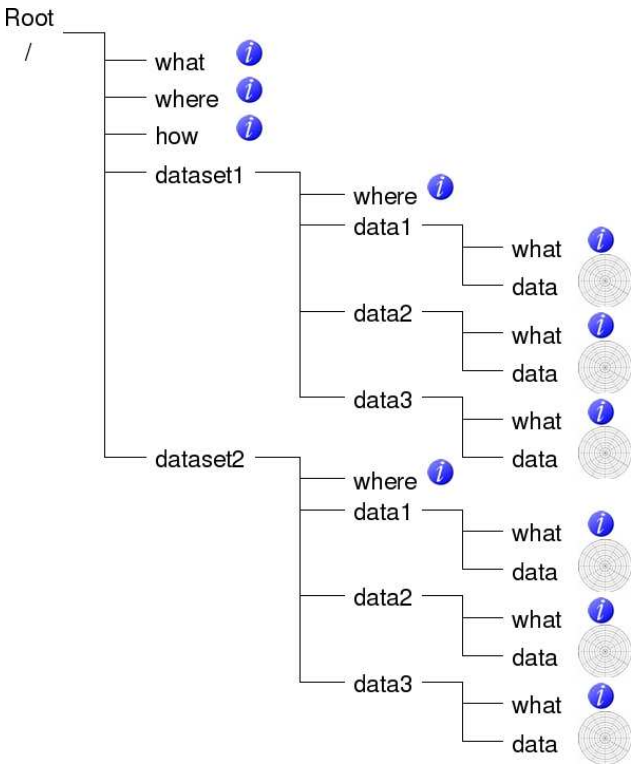

Figure 5: A polar volume containing two scans.

This information model uses the same logic to include the representation of **quality information**. Figure 6 illustrates how quality data can be added to polar data in a scan containing two parameters. In the same way that metadata can be applicable to all parameters in the scan, so too can quality information be representative either generally or locally. In this example, `/dataset1/quality1` can contain quality based on beam blockage since this is applicable to all parameters collected at the same elevation angle. However there may be different quality metrics which are applicable to each individual radar parameter, and these are stored locally. In this case, "data1" contains two such local quality metrics which are unique to that parameter, whereas "data2" only contains one. And we also see that each quality metric can be described using local metadata groups. The quality metrics should follow the guidelines set out in OPERA II[2].

Using this hierarchical structure, it becomes clear that we now have an information model capable of representing a volume containing an arbitrary number of scans/images, each of which can contain an arbitrary number of parameters which, in turn, can be characterized by an arbitrary number of quality metrics.

The examples provided in this discussion have focused on polar data, but they can also be applied to all the objects supported in this information model. These objects are polar scans, cartesian images, profiles, RHIs, cross-sections, side-panels, individual rays, sectors, and even embedded graphics images in an industrial

[2]Holleman I., Michelson D., Galli G., Germann U., and Peura M., 2006: Quality information for radars and radar data. OPERA II deliverable OPERA_2005_19.

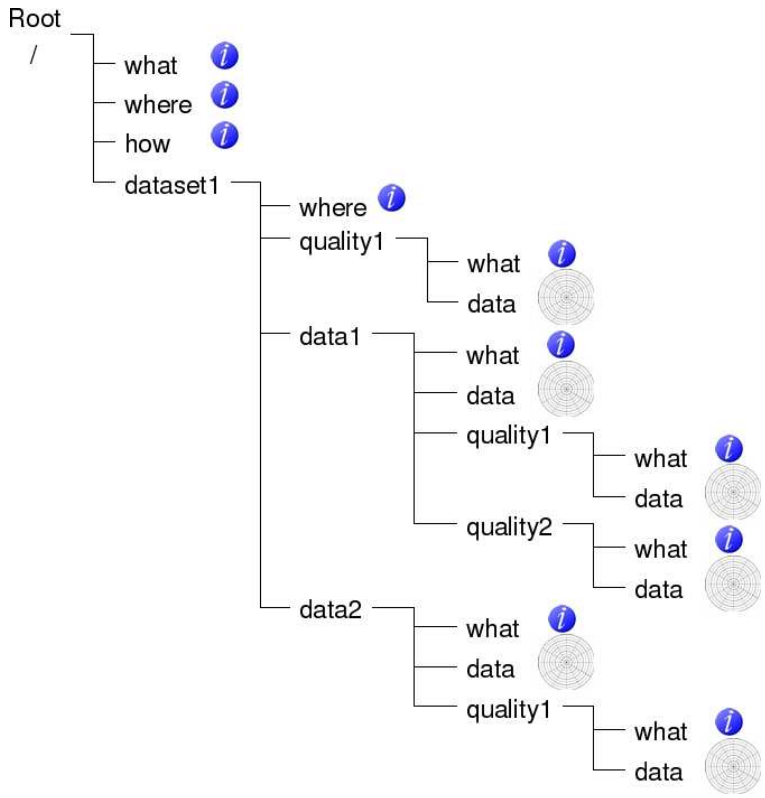

Figure 6: A polar scan containing two parameters and associated quality metrics.

graphics file format.

There are additional objects which are *complementary* to this information model and which are included since they are quite useful in various situations. One of them is the ability to add a color palette to an 8-bit dataset. There are standard mechanisms for this in the HDF5 library, and these are used without modification. Another object is a legend. This is a useful object when the data in a dataset is classified, discrete, or just level-sliced. The legend provides the ability to describe which quantitative values are represented by each qualitative value in the dataset, or which qualitative class is represented by a given value. The legend is used to tell the user that a certain value in the data represents e.g. “0.1–0.5 mm/h” in the case of a rainrate product, and e.g. “sleet” in the case of a hydrometeor classification. Both the palette and the legend objects are illustrated in Figure 7.

Finally, this information model enables the exchange of graphics representations of data. Using the same structure as that shown in Figure 1, Figure 8 illustrates this.

In the rest of this document, the detail specification of the information model is presented. With this specification, it shall be possible to read, write, and understand HDF5 files using this information model.

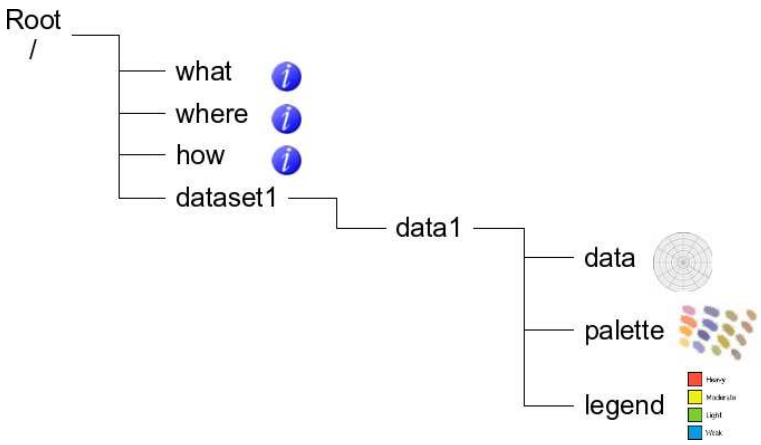

Figure 7: A polar scan containing a color palette and a legend.

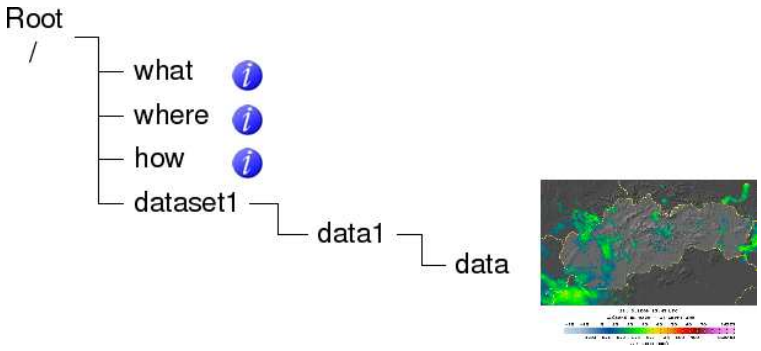

Figure 8: An embedded Slovak picture in an industrial graphics file format.

3 Definitions

HDF5 allows data to be stored as `Attribute`s, `Dataset`s, `Group`s, and user-defined `Compound` types. For use here, all types except user-defined `Compound` are permitted for use. Only `Compound` types defined in this document are permitted, in practise the optional legend object described in Section 6.2. In practice, all of these types are manipulated through the use of general-purpose `Node` objects, i.e., a `Node` may be any of the above mentioned types.

3.1 Scalars

Scalar values are stored in `Attribute` objects and may be any of the following, as defined in the HDF5 documentation:

```
H5T_STD_I8BE, H5T_STD_I8LE, H5T_STD_I16BE, H5T_STD_I16LE,
H5T_STD_I32BE, H5T_STD_I32LE, H5T_STD_I64BE, H5T_STD_I64LE,
H5T_STD_U8BE, H5T_STD_U8LE, H5T_STD_U16BE, H5T_STD_U16LE,
H5T_STD_U32BE, H5T_STD_U32LE, H5T_STD_U64BE, H5T_STD_U64LE,
H5T_NATIVE_SCHAR, H5T_NATIVE_UCHAR, H5T_NATIVE_SHORT,
H5T_NATIVE_USHORT, H5T_NATIVE_INT, H5T_NATIVE_UINT,
H5T_NATIVE_LONG, H5T_NATIVE_ULONG, H5T_NATIVE_LLONG,
H5T_NATIVE_ULLONG, H5T_IEEE_F32BE, H5T_IEEE_F32LE,
H5T_IEEE_F64BE, H5T_IEEE_F64LE, H5T_NATIVE_FLOAT,
H5T_NATIVE_DOUBLE, H5T_NATIVE_LDOUBLE, H5T_STRING
```

In practice, this means that the native type on a given platform may be any of (pseudo-C syntax):

```
char, schar, uchar, short, ushort, int, uint, long, ulong, llong,
ullong, float, double, hsize, hssize, herr, hbool, string
```

It also means that a native type written on one platform will be read and automatically returned as the corresponding native type on another platform.

Strings can be created using characters in any character set, as long as the encoding is documented; the default encoding is otherwise ASCII. Using UNICODE might be preferable to specifying a non-ASCII character set. Strings should never be terminated by the application, because they will be automatically null-terminated by the HDF5 library.

3.1.1 Booleans

A string is used to store truth value information. The string "`True`" is used to represent true, and "`False`" is used to represent false.

3.1.2 Sequences

A special kind of string may be used to store sequences. A sequence contains comma-separated scalar values in string notation. For example, a sequence is useful for storing the radar stations contributing to a composite

image, and in storing the elevation angles used in a polar volume of data.

3.2 Attributes

An `Attribute` is an HDF5 object used to store a header attribute or data. For our purposes, it is only used to store header attributes. In order to facilitate the management of so-called "atomic" attributes, ie. individual values, we use double precision for both integers (long) and floating-point (double) values. **Note** that the specification of strings is intended to be case sensitive.

3.3 Datasets

An HDF5 `Dataset` is a self-describing data object containing an *n*-dimensional binary array and attributes describing it. The array type may be any of `char`, `schar`, `uchar`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`, `llong`, `ullong`, `float`, `double` on a given platform.

In this document the text formatting of `Dataset` (in `Courier` font) means an HDF5 dataset, whereas the text formatting of **dataset** (in **bold face**) refers to the binary data in that dataset.

3.4 Groups

A `Group` is a top-level object which is used to organize other objects. For example, a `Group` may contain a collection of header `Attribute`s, a collection of `Dataset`s, or be the root object for the complete contents of an HDF5 file.

4 Metadata attribute specification

Header attributes are collected in three `Group` objects, all of which may be used recursively. `Attribute`s which describe a given file's contents and the time and place for which it represents are collected in a `Group` called `what`. `Attribute`s which describe a given file's geographical characteristics (projection, corner coordinates, dimensions, scales) are collected in a `Group` called `where`. The `what` and `where` `Group`s both contain mandatory `Attribute`s only. Those `Attribute`s which collectively describe additional data/product characteristics, such as radar system, chosen algorithm, and quality-related metadata, are stored in a `Group` called `how`. All `Attribute`s found in `how` may be considered optional, although the use of those given in this document is recommended. Additional `Attribute`s not specified in this document may only be stored in the `how` `Group`.

Top-level header attributes are collected in `what`, `where` and `how` `Group` objects located directly under the root `Group` '`/`'. Each attribute is stored as an `Attribute` object containing a scalar value. Some data/products may have `Dataset`-specific header attributes, in which case a the `Dataset` and its header `Attribute`s are collected in lower-level `what`, `where` and `how` `Group`s which are associated with that `Dataset`.

The concept used here is that each `Attribute` is identified with a string, the idea being that this is a more intuitive means of organizing data compared to numeric descriptors. It also means that a file may be queried using the *h5dump* utility to easily see and understand the contents of a file.

Mandatory header attributes for all files are given in Table 1. Note that all date and time information is for the **nominal time** of the data, ie. the time for which the data are valid. (The nominal time is not the exact acquisition time which is found elsewhere in the file.)

Note that all geographical longitude/latitude coordinates are specified with positive easting values east of the Greenwich meridian and positive northing values north of the equator.

It is possible to add new attributes, but they will not be official until they are approved and published in the following tables. Any new and unofficial attribute name must contain the prefix "`NEW:`"

4.1 Root `Group`

No HDF5 file can exist without this `Group`, since it is the starting point of the hierarchy. However, in order to make this information model compliant with the Network Common Data Form (NetCDF) file format, we require an `Attribute` in the root `Group`. This `Attribute` is called "`Conventions`" and its value is a string containing the acronym of the information model being used, and its version, in a format which lends itself to a directory structure. This is done to comply with the documentation policy maintained by UNIDATA.

This information model is called the "OPERA Data Information Model for HDF5" which gives the acronym "ODIM_H5". Recognizing the history of this information model, we start at version 2.0. The resulting value for "`/Conventions`" is "`ODIM_H5/V2_0`", and this `Attribute` must be present in all ODIM_H5 files.

The present version is 2.0.

4.2 Top-level `what Group`

In this section the content of the top-level `what` is described.

This `Group` contains mandatory `Attributes` only which collectively describe a given file's contents and time of acquisition. These `Attributes` are given in Tables 1-14.

Table 1: Mandatory top-level `what` header `Attributes` for all weather radar files.

Name	Type	Format	Description
object	string	-	According to Table 2
version	string	H5rad M.m	Format or information model version. "M" is the major version. "m" is the minor version. Software is encouraged to warn if it receives a file stored in a version which is different from the one it expects. The software should, however, proceed to read the file, ignoring `Attributes` it does not understand.
date	string	YYYYMMDD	Nominal Year, Month, and Day of the data/product
time	string	HHmmss	Nominal Hour, Minute, and Second, in UTC of the data/product
source	string	TYP:VALUE	Variable-length string containing pairs of identifier types and their values, separated by a colon. Several pairs can be concatenated, separated by commas, in the form TYP:VALUE,TYP:VALUE, etc. Types and their values are given in Table 3.

Object strings may be any of those given in Table 2. For each `Dataset`, there may be an accompanying `what Group` with information specific to that `Dataset`, according to Table 13.

Table 2: File object strings and their meanings

String	Description
PVOL	Polar volume
CVOL	Cartesian volume
SCAN	Polar scan
RAY	Single polar ray
AZIM	Azimuthal object
IMAGE	2-D cartesian image
COMP	Cartesian composite image(s)
XSEC	2-D vertical cross section(s)
VP	1-D vertical profile
PIC	Embedded graphical image

Note that an RHI polar volume can be represented using a number of `Datasets` and the XSEC object.

Table 3: Source type identifiers and their associated values. It is mandatory for at least one of WMO, RAD, ORG, or CTY to be present.

Identifier	Description	Example
WMO	Combined WMO block and station number in the form A_1b_wnnnnn, or 0 if none assigned. The first two digits represent the block number, where the first digit A_1 is the regional association area and the second digit b_w is the sub-area. Remaining digits are the station number. (According to the WMO, numbers in the form A_1b_wnnn are considered equivalent to the form A_1b_w00nnn). One mandatory way to identify single-site data. References: 1, 2.	WMO:02954
RAD	Radar site as indexed in the OPERA database. One mandatory way to identify single-site data.	RAD:FI44
ORG	Originating centre. One mandatory way to identify composites.	ORG:247
PLC	Place according to Table 9 of this document	PLC:Anjalankoski
CTY	Country according to BUFR tables 14 0 1 101	CTY:613
CMT	Comment: allowing for a variable-length string	CMT:Suomi tutka

4.3 `where` Group

In this section the `Attributes` for the `where Group` are described. These are different for polar or cartesian `Datasets`, i.e. containing `Dataset` and `Dataset Groups` respectively.

Note that the use of the `where Group` is mandatory but that its placement will be at the top level of a given file, and at a lower level associated with a given `Dataset`. This is because some attributes are valid globally (e.g. the coordinates of a radar) whereas others are local (e.g. the elevation angle used for a given sweep).

4.3.1 `where` for polar data `Groups`

This `where Group` contains mandatory `Attributes` only which collectively describe geographical and geometrical characteristics of a given `Dataset` **dataset**. **Note** that the **dataset** itself is the object containing the binary data and that `where` in this context describes that **dataset** but is located in the corresponding `Dataset` which contains all these objects. Section 2 illustrates how these two objects are related to each other.

Polar data, i.e. raw radar data as a function of azimuth and range, are stored in a `Dataset Group`, and polar volume data are stored as a stack of these `Dataset Groups`. Each `Dataset Group` contains azimuthal data from a single elevation.

Table 4: `where` `Attribute`s for polar data objects.

Name	Type	Description
lon	double	Longitude position of the radar antenna (degrees). Fractions of a degree are given in decimal notation.
lat	double	Latitude position of the radar antenna (degrees). Fractions of a degree are given in decimal notation.
height	double	Height of the centre of the antenna in meters above sea level.
`Dataset` specific		
elangle	double	Antenna elevation angle (degrees) above the horizon.
nbins	long	Number of range bins in each ray
rstart	double	The range (km) of the start of the first range bin
rscale	double	The distance in meters between two successive range bins
nrays	long	Number of azimuth gates (rays) in the object
a1gate	long	Index of the first azimuth gate radiated in the scan
Sector specific		
startaz	double	The azimuth angle of the start of the first gate in the sector (degrees)
stopaz	double	The azimuth angle of the end of the last gate in the sector (degrees)

4.3.2 `where` for geographically referenced image `Group`s

This `where Group` contains mandatory `Attribute`s only which collectively describe a given `Dataset`'s geographical and geometrical characteristics. **Note** that the **dataset** is the object containing the binary data and that `where` in this context describes that **dataset** but is located in the corresponding `Dataset` which contains all these objects. Section 2 illustrates how these two objects are related to each other.

The PROJ.4 cartographic projections library[3] is a comprehensive means of managing geographically referenced information which has become a *de facto* standard. PROJ.4 is being used increasingly throughout Europe and the world. As a result, the most straightforward way of representing projection information in radar files is by means of the projection definition string which is used with the library itself. For example, in a Swedish composite product, the Polar Stereographic projection may be used with a spherical earth model (radius 6376177 m), an origin at the north pole and 14°E, and a latitude of true-scale at 60°N. The arguments used with PROJ.4 to define this simple projection are `+proj=stere +R=6376177 +lat_0=90 +lon_0=14 +lat_ts=60` so this is what should be found as an `Attribute` in the `where Group` associated with the `Dataset` used to store the composite data.

Note that PROJ.4 contains a complete set of arguments for specifying a given projection, including false easting/northing, rescaling of coordinates, choice of ellipsoid (or defining your own), choice of geodetic datum, and defining oblique (rotated) projections.

[3] originally from the United States Geological Survey, now from MapTools

Table 5: `where` `Attributes` for geographical image data `Groups`

Name	Type	Description
projdef	string	The projection definition arguments, described above, which can be used with PROJ.4. See the PROJ.4 documentation for usage
xsize	long	Number of pixels in the X dimension
ysize	long	Number of pixels in the Y dimension
xscale	double	Pizel size in the X dimension, in projection-specific coordinates (often meters)
yscale	double	Pixel size in the Y dimension, in projection-specific coordinates (often meters)
LL_lon	double	Longitude of the lower left pixel corner
LL_lat	double	Latitude of the lower left pixel corner
UL_lon	double	Longitude of the upper left pixel corner
UL_lat	double	Latitude of the upper left pixel corner
UR_lon	double	Longitude of the upper right pixel corner
UR_lat	double	Latitude of the upper right pixel corner
LR_lon	double	Longitude of the lower right pixel corner
LR_lat	double	Latitude of the lower right pixel corner

Note that the corner coordinates given must be for the actual corners of the respective pixels.

4.3.3 `where` for cross-section data `Group`

RHI and cross-sections are treated as a special form of cartesian image. The x-dimension of the image represents the coordinate in the x/y-plane, while the y-dimension describes the vertical coordinate of the RHI or cross-section. To describe the geographical orientation and extend of a RHI or cross-section a dedicated set of `Attributes` in the `where` `Group` has been defined. The geographical location of cross-sections is just given by longitudes and latitudes of start and stop positions. The cross-sections are thus assumed to be taken along great-circles. In case they are taken along a line in a plane of a geographical projection, the deviation from the great-circle will be negligible for visualization purposes.

Table 6: `where` `Attributes` for cross-section data.

Name	Type	Description
Common attributes		
xsize	long	Number of pixels in the horizontal dimension
ysize	long	Number of pixels in the vertical dimension
xscale	double	Horizontal resolution in m
yscale	double	Vertical resolution in m
minheight	double	Minimum height in meters above mean sea level
maxheight	double	Maximum height in meters above mean sea level
RHI specific		

continued on next page

continued from previous = page

Name	Type	Description
lon	double	Longitude position of the radar antenna (degrees). Fractions of a degree are given in decimal notation.
lat	double	Latitude position of the radar antenna (degrees). Fractions of a degree are given in decimal notation.
az_angle	double	Azimuth angle
angles	sequence	Elevation angles in the order of acquisition.
range	double	Maximum range in km
Cross section and side panel specific		
start_lon	double	Start position's longitude
start_lat	double	Start position's latitude
stop_lon	double	Stop position's longitude
stop_lat	double	Stop position's latitude

4.3.4 `where` for vertical profiles

This `where` `Group` contains mandatory `Attribute`s only which collectively describe the geographical and geometrical characteristics of vertical profiles of horizontal winds and/or radar reflectivity.

Table 7: `where` `Attribute`s for vertical profiles.

Name	Type	Description
lon	double	Longitude position of the radar antenna (degrees). Fractions of a degree are given in decimal notation.
lat	double	Latitude position of the radar antenna (degrees). Fractions of a degree are given in decimal notation.
height	double	Height of the centre of the antenna in meters above sea level.
levels	long	Number of points in the profile
interval	double	Vertical distance (m) between height intervals, or 0.0 if variable
minheight	double	Minimum height in meters above mean sea level
maxheight	double	Maximum height in meters above mean sea level

4.4 `how` `Group`

This `Group` contains recommended and other `Attribute`s which provide additional and complimentary information which can be used to describe a given `Dataset` object, for example information related to an object's quality. Attributes in `how` can be added, but they won't offially constitute the OPERA standard until they are added to the tables in this information model. a **Note** that the use of the `how` `Group` is optional and that its placement may be either at the top level of a given file, or at a lower level associated with a given `Dataset`. If `how` is found at the top level, then it is assumed that its contents apply to all `Dataset`s in the file. If `how` is found at a lower level, then it must be located in the `Dataset` `Group` to which its contents apply. If `how` exists at both the top and lower levels, then the contents of the local `how` override the contents of the top level `how`.

Note as well that there can often be cases where some `Attributes` apply for all `Datasets` in a file and others may be different for each `Dataset`. In such cases, the `Attributes` which apply to all `Datasets` may be held in a top level `where Group` and those that change may be held in local `where Groups`.

For clarity, Table 8 containing recommended `how Attributes` is partitioned such that different partitions contain different product-specific `Attributes`.

Table 8: `how Attributes` for all objects.

Name	Type	Description
General		
task	string	Name of the acquisition task (polar data), or product generator (cartesian product)
startepochs	long	Seconds after a standard 1970 epoch for which the starting time of the data/product is valid. A compliment to "date" and "time" in Table 1.
endepochs	long	Seconds after a standard 1970 epoch for which the ending time of the data/product is valid. A compliment to "date" and "time" in Table 1.
system	string	According to Table 10
software	string	According to Table 11
sw_version	string	Software version in string format, e.g. "5.1" or "8.11.6.2"
zr_a	double	Z-R constant A in $Z = A\ R^b$, applicable to any product containing reflectivity or precipitation data
zr_b	double	Z-R exponent b in $Z = A\ R^b$, applicable to any product containing reflectivity or precipitation data
kr_a	double	K_{dp}-R constant A in $R = A\ {K_{dp}}^b$
kr_b	double	K_{dp}-R exponent b in $R = A\ {K_{dp}}^b$
simulated	boolean	Data are simulated
Data from individual radars		
beamwidth	double	The radar's half-power beamwidth (degrees)
wavelength	double	Wavelength in cm
rpm	double	The antenna speed in revolutions per minute, positive for clockwise scanning, negative for counter-clockwise scanning
pulsewidth	double	Pulse width in μs
lowprf	long	Low pulse repetition frequency in Hz
highprf	long	High pulse repetition frequency in Hz
Polar data		
azmethod	string	How raw data in azimuth are processed to arrive at the given value, according to Table 12
binmethod	string	How raw data in range are processed to arrive at the given value, according to Table 12
azangles	sequence	Azimuthal start and stop angles (degrees) used for each azumith gate in a scan. The format for each start-stop pair of angles is 'start:stop' and each pair is separated by a comma.
elangles	sequence	Elevation angles (degrees above the horizon) used for each azumith gate in an "intelligent" scan that e.g. follows the horizon.

continued on next page

continued from previous page

Name	Type	Description
aztimes	sequence	Acquisition start and stop times for each azimuth gate in the sector or scan. The format for each start-stop pair of times is 'HHMMSS.sss:HHMMSS.sss' and each pair is separated by a comma. The required precision is to the millisecond.
Cartesian images including composites		
angles	sequence	Elevation angles in ascending order, used to generate the product
arotation	sequence	Antenna rotation speed
camethod	string	How cartesian data are processed, according to Table 12
nodes	sequence	Radar nodes (Table 9) which have contributed data to the composite, e.g. "'arl', 'osl', 'ase', 'kor"'
ACCnum	long	Number of images used in precipitation accumulation
Vertical profile specific		
minrange	double	Minimum range at which data is used when generating profile (km)
maxrange	double	Maximum range at which data is used when generating profile (km)
NI	double	Unambiguous velocity (Nyquist) interval (±m/s)
dealiased	long	1 if data has been dealiased, 0 if not
Quality (subject to modification by OPERA WP 1.2)		
pointaccEL	double	Antenna pointing accuracy in elevation (degrees)
pointaccAZ	double	Antenna pointing accuracy in azimuth (degrees)
malfunc	boolean	Radar malfunction indicator
radar_msg	string	Radar malfunction message
radhoriz	double	Radar horizon (maximum range in km)
MDS	double	Minimum detectable signal (dBm) at 10 000 m
OUR	double	Overall uptime reliability (%)
Dclutter	sequence	Doppler clutter filters used when collecting data
comment	string	Free text description. Anecdotal quality information
SQI	double	Signal Quality Index threshold value
CSR	double	Clutter-to-signal ratio threshold value
LOG	double	Security distance above mean noise level (dB) threshold value
VPRCorr	boolean	Vertical reflectivity profile correction applied
freeze	double	Freezing level (km) above sea level
min	double	Minimum value for continuous quality data
max	double	Maximum value for continuous quality data
step	double	Step value for continuous quality data
levels	long	Number of levels in discrete data legend
peakpwr	double	Peak power (kW)
avgpwr	double	Average power (W)
dynrange	double	Dynamic range (dB)
RAC	double	Range attenuation correction (dBm)
BBC	boolean	Bright-band correction applied
PAC	double	Precipitation attenuation correction (dBm)
S2N	double	Signal-to-noise ratio (dBm) threshold value
polarization	string	Type of polarization (H, V)

Table 9: Examples of radar "places" and their node designations. The first two letters in the "node" represent the country, following the Internet convention. The "place" name may use UNICODE characters. The "node" must only use ASCII. Radars not in this table can be added.

Place	Node
Hasvik	nohas
Andøya	noand
Røst	norst
Rissa	norsa
Bømlo	nobml
Hægebostad	nohgb
Oslo	noosl
Kiruna	sekir
Luleå	selul
Örnsköldsvik	seovi
Östersund	seosu
Hudiksvall	sehud
Leksand	selek
Arlanda	searl
Vilebo	sevil
Vara	sevar
Ase	sease
Karlskrona	sekkr
Ängelholm	seang
Korpo	fikor
Vantaa	fivan
Anjalankoski	fianj
Ikaalinen	fiika
Kuopio	fikuo
Vimpeli	fivim
Utajärvi	fiuta
Luosto	filuo
Bornholm	dkbor
Stevns	dkste
Sindal	dksin
Rømø	dkrom
De Bilt	nldbl
Den Helder	nldhl
Tallinn	eetal
Sürgavere	eesur
Riga	lvrix
Legionowo	plleg
Poznan	plpoz
Gdansk	plgda
Swidwin	plswi

continued on next page

continued from previous page

Place	Node
Rzeszow	plrze
Brzuchania	plbrz
Ramza	plram
Pastewnik	plpas

Table 10: Radar System abbreviations and their meanings. Radars not in this table can be added.

String	Meaning
GEMAxxx	Gematronik Meteor ACxxx Radar
EECxxx	EEC xxx Radar
ERICxxx	Ericsson xxx Radar
VAISxxx	Vaisala xxx Radar

Table 11: Processing Software abbreviations and their meanings. Systems not in this table can be added.

String	Meaning
CASTOR	Météo France's system
EDGE	EEC Edge
FROG	Gamic FROG, MURAN ...
IRIS	Sigmet IRIS
NORDRAD	NORDRAD
RADARNET	UKMO's system
RAINBOW	Gematronik Rainbow

Table 12: Method abbreviations and their meanings.

String	Meaning
NEAREST	Nearest neighbour or closest radar
INTERPOL	Interpolation
AVERAGE	Average of all values
RANDOM	Random
MDE	Minimum distance to earth
LATEST	Most recent radar
MAXIMUM	Maximum value
DOMAIN	User-defined compositing
VAD	Velocity azimuth display
VVP	Volume velocity processing
RGA	Gauge-adjustment

4.5 `what Group` for `Dataset` objects

In this section the content of the `what Group` to be used with each `Dataset` is described. Note that the linear transformation coefficients "gain" and "offset" should be set to 1 and 0 respectively if they are not intended for use with a given `Dataset`.

Table 13: `Dataset`-specific `what` header `Attributes`.

Name	Type	Format	Description
product	string	-	According to Table 14
prodpar	Tab. 15	-	According to Table 15 for products. Only used for cartesian products.
quantity	string	-	According to Table 16
startdate	string	Starting YYYYMMDD	Year, Month, and Day for the product
starttime	string	Starting HHmmss	Hour, Minute, and Second for the product
enddate	string	Ending YYYYMMDD	Year, Month, and Day for the product
endtime	string	Ending HHmmss	Hour, Minute, and Second for the product
gain	double	-	Coefficient 'a' in y=ax+b used to convert to unit. Default value is 1.0.
offset	double	-	Coefficient 'b' in y=ax+b used to convert to unit. Default value is 0.0.
nodata	double	-	Raw value used to denote areas void of data (never radiated). **Note** that this `Attribute` is always a float even if the data in question is in another format.
undetect	double	-	Raw value used to denote areas below the measurement detection threshold (radiated but nothing detected). **Note** that this `Attribute` is always a float even if the data in question is in another format.

Table 14: Product abbreviations and their meanings.

String	Meaning
SCAN	A scan of polar data
PPI	Plan position indicator
CAPPI	Constant altitude PPI
PCAPPI	Pseudo-CAPPI
ETOP	Echo top
MAX	Maximum
RR	Accumulation
VIL	Vertically integrated liquid water
COMP	Composite
VP	Vertical profile
RHI	Range height indicator
XSEC	Arbitrary vertical slice
VSP	Vertical side panel

continued on next page

continued from previous page

String	Meaning
HSP	Horizontal side panel
RAY	Ray
AZIM	Azimuthal type product
QUAL	Quality metric

Table 15: Product parameters.

Product	Type	Product parameter
CAPPI	double	Layer height (meters above the radar)
PPI	double	Elevation angle used (degrees)
ETOP	double	Reflectivity level (dBZ)
RHI	double	Azimuth angle (degrees)
VIL	sequence	Bottom and top heights (m) of the integration layer

Table 16: Quantity (variable) identifiers.

String	Quantity [Unit]	Description
TH	T_h [dBZ]	Logged horizontally-polarized total (uncorrected) reflectivity factor
TV	T_v [dBZ]	Logged vertically-polarized total (uncorrected) reflectivity factor
DBZH	Z_h [dBZ]	Logged horizontally-polarized (corrected) reflectivity factor
DBZV	Z_v [dBZ]	Logged vertically-polarized (corrected) reflectivity factor
ZDR	ZDR [dBZ]	Logged differential reflectivity
RHOHV	ρ_{hv} [0-1]	Correlation between Z_h and Z_v
LDR	L_{dr} [dB]	Linear depolarization ratio
PHIDP	ϕ_{dp} [degrees]	Differential phase
KDP	K_{dp} [degrees/km]	Specific differential phase
SQI	SQI [0-1]	Signal quality index
SNR	SNR [0-1]	Normalized signal-to-noise ratio
RATE	RR [mm/h]	Rain rate
ACRR	$\mathrm{RR}_{accum.}$ [mm]	Accumulated precipitation
HGHT	H [km]	Height (of echotops)
VIL	VIL [kg/m^2]	Vertical Integrated Liquid water
VRAD	V_{rad} [m/s]	Radial velocity
WRAD	W_{rad} [m/s]	Spectral width of radial velocity
UWND	U [m/s]	Component of wind in x-direction
VWND	V [m/s]	Component of wind in y-direction
BRDR	0 or 1	1 denotes a border where data from two or more radars meet in composites, otherwise 0
QIND	Quality [0-1]	Spatially analyzed quality indicator, according to OPERA II, normalized to between 0 (poorest quality) to 1 (best quality)

continued on next page

continued from previous page

String	Quantity [Unit]	Description
CLASS	Classification	Indicates that data are classified and that the classes are specified according to the associated legend object (Section 6.2) which must be present.
Vertical profile specific		
ff	[m/s]	Mean horizontal wind velocity
dd	[degrees]	Mean horizontal wind direction (degrees)
ff_dev	[m/s]	Velocity variability
dd_dev	[m/s]	Direction variability
n	–	Sample size
dbz	[dBZ]	Logged radar reflectivity factor
dbz_dev	[dBZ]	Variability of logged radar reflectivity factor
z	[Z]	Linear radar reflectivity factor
z_dev	[Z]	Variability of linear radar reflectivity factor
w	[m/s]	Vertical velocity (positive upwards)
w_dev	[m/s]	Vertical velocity variability
div	$[s^{-1}]$	Divergence
div_dev	$[s^{-1}]$	Divergence variation
def	$[s^{-1}]$	Deformation
def_dev	$[s^{-1}]$	Deformation variation
ad	[degrees]	Axis of dialation (0-360)
ad_dev	[degrees]	Variability of axis of dialation (0-360)
chi2	–	Chi-square value of wind profile fit
rhohv	ρ_{hv} [0-1]	Correlation between Z_h and Z_v
rhohv_dev	ρ_{hv} [0-1]	ρ_{hv} variation

5 Data specification

All data arrays, for polar, cartesian, and profile data, are stored as **dataset** objects. Any of compression levels 1 to 6 are recommended. (HDF5's built-in compression levels range from 0 (none) to 9 (maximum); levels above 6 tend to result in the algorithm using disproportionate resources relative to gains in file size, which is why their use is not encouraged.) HDF5 provides support for SZIP compression in addition to default ZLIB compression, but SZIP compression library is proprietary and will therefore not be supported in any official OPERA software.

All `Dataset Group`s are called `dataset`*n*, no matter which kind of data they hold. The character *n* represents the index of the `Dataset Group` in ascending order (both in terms of elevation angle and height) starting at 1. A `Dataset Group` can hold an arbitrary number of binary **dataset**s. In the case of radar parameters (e.g. Z, V, W, etc.) the **dataset**s containing each parameter are each contained in a `Group` called `data`*n*, where *n* is the index of the **dataset** holding the binary data. This **dataset** is simply called `data`.

Within each `data`*n* `Group`, there may be an arbitrary number of quality indicators, each of which is held in a `Group` called `quality`*n*, where *n* is the index of the **dataset** holding the binary data. This **dataset** is simply called `data`.

Each `Dataset Group` can store local `what`, `where` and `how Group`s to store metadata which are unique to that `Dataset`.

Note that all data with 8-bit unsigned depth (`uchar`) shall be represented as an HDF5 Image (H5IM) (Table 17). This applies to any 2-D `Dataset`, be it polar, cartesian, sector, or cross-section. This is a `Dataset` with a few added `Attribute`s which facilitate the Image's management by third-party software.

Table 17: Eight-bit Image attributes. Note that these are part of a `Dataset` object.

Attribute	Type	Description
CLASS	string	Should be "IMAGE"
IMAGE_VERSION	string	Should be "1.2", the current version number

There are several optional `Attribute`s describing an HDF5 Image, including Palette information (see Section 6.1).

A schematic of this organization is found in Section 2. What follows is details pertaining to each type of data that may be represented as binary data.

5.1 Polar Data

The start of the first azimuth gate (the first pulse) always points due north and the first range bin is that starting at the radar. This applies to data representing a full sweep, but not sector scans. For full sweeps, this implies that the first azimuth gate covers the interval 0-1° assuming 360 azimuth gates per scan. Azimuth gates are ordered clockwise except for sector scans which can be clockwise or counter-clockwise. In other

words, range bins are stored in the array's equivalent X-dimension and azimuth gates in the Y-dimension. There is no mechanism for specifying missing azimuth gates or range bins in SCAN objects. This means that partial scans/rays must be filled-in in order to complete them. Filled-in areas are identified by the "nodata" value specified in the corresponding `what Group`. Radiated areas with no echo are represented with the "undetect" value also in the corresponding `what Group`. Alternatively, collections of rays can be stored in the RAY object, and sectors can be stored in the AZIM object. The first azimuth gate in the scan does not have to be the first ray of data collected for that scan, but the `/dataset`*n*`/where/a1gate Attribute` contains the information on this, thereby allowing a temporal reconstruction of the data collection.

All full-sweep data must be sorted according to the above description: clockwise and starting from north, even if they haven't been acquired that way. It is therefore the responsibility of each supplier to comply with these specification.

All parameters collected in a single scan of data are contained in one `Dataset Group` with the same index number, as seperate `data Dataset Group`s.

5.2 Image Data

An image product in this context refers to 2-D cartesian quantitative data and not a visual graphic product (PIC, see Section 5.6 below).

Binary arrays are stored as one long unpadded binary string starting in the upper-left corner and proceeding row by row (north to south), from left (west) to right (east). Areas void of the specified variable are flagged using the "nodata" value specified in the corresponding `what Group`. Radiated areas with no echo are represented with the "undetect" value also in the corresponding `what Group`.

5.3 RHIs, cross sections and side panels

RHI, cross sections and side panels are a special form of image. RHIs taken with the antenna pointing east start on the left and end on the right. If the antenna points west, then the RHI starts on the right and ends on the left. RHIs pointing exactly south or north always start on the left and end on the right. For cross sections, the left side of the image is the starting point and the right side is the finishing point, regardless of the antenna's azimuth angle. For side panels, the starting point is north for vertical panels and west for horizontal panels. As with other image files, the first pixel is the upper-left one and image content is ordered row-wise from top to bottom and left to right.

5.4 Profiles

In contrast to polar or cartesian data which use `Dataset`s to store 2-D arrays, profiles use several `Dataset`s to store 1-D arrays representing different variables along a given profile. One variable is stored in each `Dataset`. Levels in the profile are ordered sequentially in ascending order. A profile is a `Group` containing several `Dataset`s, each of which stores a variable for that level. This profile formulation is not restricted to wind variables, but can easily accommodate reflectivity and other variables as well.

Each `Dataset` containing a parameter making up the the profile contains a quantity designator according to Table 16.

Note that all **dataset**s must be of equal length, in order to match the heights given in the **dataset** containing the quantity HGHT.

5.5 Rays and sectors

It is unlikely that individual rays or sectors of data will be exchanged operationally and internationally, but these objects are added for the sake of completeness and in cases where quality information can be efficiently represented using these objects. Also, there may be production chains where the radar transmits each ray individually, so the availability of this means of representation can support them.

An individual ray of data is stored as a **dataset** with the most proximate data first. Missing data along the ray must be assigned the “nodata” and/or “undetect” values.

A sector is similar to a scan with the difference being that the sector doesn’t cover the horizon completely. A sector is stored the same was as a scan, the only clarification being the object name (Table 2 or 14) and the metadata in

Alternatively, sectors can be be represented as a collection of rays. This might be handy if the rays are few and with variable starting and ending rays, and/or with different range bin spacing from one ray to the next.

5.6 Embedded graphical images

Image files in industrial graphics formats, e.g. PNG, JPEG, GIF, TIFF, etc., can be written directly into a **dataset** as is. When this file is then retrieved, all that is necessary is to read the **dataset** contents as is and write a new file containing them.

6 Optional objects

There are two kind of objects which may be included in the HDF5 files and which are considered optional in this information model. These are palettes which may be attached to a given **dataset** to "color" them intuitively, and discrete legends which may complement a given **dataset** with what can be considered quality-related information. Each of these objects is specified below.

6.1 Palettes

The HDF5 Image API contains functionality for associating palettes (or color tables) with 8-bit and 24-bit arrays. Since we have not defined 24-bit graphic images explicitely in this information model, the treatment of palettes will be limited to 8-bit **dataset**s. Most radar data exchanged in Europe today is 8-bit, so the ability to complement them with palettes can be considered an enhancement. This is particularly interesting since the standard HDF5 binary tool *h52gif* converts an 8-bit **dataset** to a GIF file with or without an associated palette. And the *hdfview* tool visualizes the data with the palette if one is available. The reader is referred to the HDF5 documentation for complete information.

In order for an 8-bit `Dataset` to work with a palette, it must be defined as an `Image`. This is done by adding a few attributes to the `Dataset`, and this process does not impact at all on the binary array data. In other words, applications that only read the data won't be affected. This means that those applications that wish to use the palette can do so and those that don't can do so without having to modify any routines. The palette is defined as a seperate `Dataset` and then the palette is linked to the binary array `Dataset`. This does not affect the binary data either. The result is a straight-forward and unobtrusive mechanism for adding color to data.

An example of how a palette complements an 8-bit dataset is given in Table 19.

6.2 Legends

The HDF5 library enables users to define and store so-called "compound" data type objects. Compound data type objects are heterogeneous elements - a multi-dimensional array of fields. Such arrays consist of fields that either represent atomic data types defined in the HDF5 API, or compound data types. In the latter case such construction is referred to as "nested compound type". The reader is referred to the HDF5 documentation for complete information; what follows here is summarized.

Compound data type objects are handled by the HDF5 library which provides functions for reading and writing data to and from a **dataset** of a compound type. With these functions, either the whole record (an instance of compond data type) can be read or written or particular fields can be accessed in both read and write mode.

The compound data type is designed for use mainly in C language applications. This is a direct consequence of the approach to memory organization and access, which is based on C structures. However, it is still possible to use compound data types in Java applications, as well as to map compound data types to XML node structures.

The compound data type is suitable for holding and processing legend data relating to discrete quality information. In most cases, such a legend would contain two elements: *key* and *value*. *Key* is an abbreviated

character code for a given level of quality parameter it describes. *Value* holds the actual parameter value in the form of a character string. This pair of fields can be represented by the following C structure:

```
struct legend
{
    char[] key;
    char[] value;
};
```

The structure describes a given discrete quality parameter which, for example, can be a data quality indicator depending on distance to the radar. Such quality indicator can be stratified into several discrete categories, represented by character strings of a given contstant length. Another example is a hydrometeor classification based on polarimetric variables (rain, snow, hail, clutter ...). In both case, the key field of the legend structure will hold the category string, while the corresponding value will be stored in value field, which is also an array of characters. The length of particular fields can be either given explicitly (e.g. described in dedicated quality parameters table) or can be determined at runtime.

Since HDF5 approaches data object in the way the C language does, it is crutial to meet C standards while designing and processing compound data structures. Because C does not support string objects (strings of characters of variable length), the string object can only be represented by an array of characters. In addtition, to provide the possibility to determine the length of a legend dataset at runtime, an instance of a legend struture object should have fixed and constant byte length. This can be achieved by using character arrays of constant length. For the purpose of legend dataset, it seems reasonable to use 64-byte array for the key field, and 32-byte array for value field. The space used by the actual content of each legend structure depends on the user. The only requirement is that character is null terminated (the last character in array is null character).

```
struct legend
{
    char[64] key;
    char[32] value;
};
```

Once the legend structure is defined, instances of the legend structure are stored in a dedicated **dataset**. When defining a **dataset** data type, `H5T_COMPOUND` type must be used.

Endnote on software compatibility and versions

The use of the NetCDF file format has also gained momentum within the meteorological and climatological communities, yet it lags behind HDF5 as general-purpose format for spatially-distributed environmental data. Today NetCDF is a high-level layer built upon HDF5. Starting with HDF5 version 1.8.0 and NetCDF version 4, HDF5 is able to manage NetCDF files. There are also Java tools with NetCDF which enable reading and writing of some kinds of HDF5 files. Since we recognize the benefits of being able to manage HDF5 files with NetCDF (and vice versa), we require that any implementation of the information model specified in this document must use HDF5 version 1.8.0 or later. By organizing data in HDF5 in the straight-forward way specified in this document, the simplicity should facilitate for NetCDF to manage such attributes and datasets.

7 Mandatory metadata per product

The purpose of this section is to clarify exactly which metadata attributes are mandatory for each type of product. So far, we have only mentioned that `what` and `where` `Group`s are mandatory, whereas `how` is optional/voluntary. Here, we are specific about how which metadata **must always** be present for each kind of product. In order for this to be a reasonable level of ambition, only those metadata which are fundamentally necessary to be able to manage the product at all, ie. at a purely functional level, are mandatory.

The method used to present the mandatory metadata is in the form of terse listings, using the file-system analogy shown in Section 2 on page 2. The exact format of each node in the HDF5 file is according to the tables referred to in this document.

Datasets are also included in the following tables, for clarity.

7.1 Polar volume

The following example represents a polar volume consisting of two scans, each of which contains two parameters. The polar geometry is the same in both scans, the only differences being the elevation angle and the first measured azimuth gate in each scan.

Table 18: Polar volume

Node	Type
`/`	Root Group
`/Conventions`	Attribute, Section 4.1
`/what`	Group
`/what/object`	Atttribute, Table 2
`/what/version`	Atttribute, Table 1
`/what/date`	Atttribute, Table 1
`/what/time`	Atttribute, Table 1
`/what/source`	Atttribute, Table 3
`/where`	Group
`/where/lon`	Attribute, Table 4
`/where/lat`	Attribute, Table 4
`/where/height`	Attribute, Table 4
`/dataset1`	Group
`/dataset1/what`	Group
`/dataset1/what/product`	Attribute, Table 14
`/dataset1/what/startdate`	Attribute, Table 13
`/dataset1/what/starttime`	Attribute, Table 13
`/dataset1/what/enddate`	Attribute, Table 13
`/dataset1/what/endtime`	Attribute, Table 13
`/dataset1/where`	Group
`/dataset1/where/elangle`	Attribute, Table 4
`/dataset1/where/a1gate`	Attribute, Table 4
`/dataset1/where/nbins`	Attribute, Table 4

continued on next page

continued from previous page

Node	Type
/dataset1/where/rstart	Attribute, Table 4
/dataset1/where/rscale	Attribute, Table 4
/dataset1/where/nrays	Attribute, Table 4
/dataset1/data1	Group
/dataset1/data1/what	Group
/dataset1/data1/what/quantity	Attribute, Table 16
/dataset1/data1/what/gain	Attribute, Table 13
/dataset1/data1/what/offset	Attribute, Table 13
/dataset1/data1/what/nodata	Attribute, Table 13
/dataset1/data1/what/undetect	Attribute, Table 13
/dataset1/data1/data	Dataset
/dataset1/data1/data/CLASS	Attribute, Table 17
/dataset1/data1/data/IMAGE_VERSION	Attribute, Table 17
/dataset1/data2	Group
/dataset1/data2/what	Group
/dataset1/data2/what/quantity	Attribute, Table 16
/dataset1/data2/what/gain	Attribute, Table 13
/dataset1/data2/what/offset	Attribute, Table 13
/dataset1/data2/what/nodata	Attribute, Table 13
/dataset1/data2/what/undetect	Attribute, Table 13
/dataset1/data2/data	Dataset
/dataset1/data2/data/CLASS	Attribute, Table 17
/dataset1/data2/data/IMAGE_VERSION	Attribute, Table 17
/dataset2	Group
/dataset2/what	Group
/dataset2/what/product	Attribute, Table 14
/dataset2/what/product	Attribute, Table 14
/dataset2/what/startdate	Attribute, Table 13
/dataset2/what/starttime	Attribute, Table 13
/dataset2/what/enddate	Attribute, Table 13
/dataset2/what/endtime	Attribute, Table 13
/dataset2/where	Group
/dataset2/where/elangle	Attribute, Table 4
/dataset2/where/a1gate	Attribute, Table 4
/dataset2/where/nbins	Attribute, Table 4
/dataset2/where/rstart	Attribute, Table 4
/dataset2/where/rscale	Attribute, Table 4
/dataset2/where/nrays	Attribute, Table 4
/dataset2/data1	Group
/dataset2/data1/what	Group
/dataset2/data1/what/quantity	Attribute, Table 16
/dataset2/data1/what/gain	Attribute, Table 13
/dataset2/data1/what/offset	Attribute, Table 13
/dataset2/data1/what/nodata	Attribute, Table 13
/dataset2/data1/what/undetect	Attribute, Table 13

continued on next page

continued from previous page

Node	Type
/dataset2/data1/data	Dataset
/dataset2/data1/data/CLASS	Attribute, Table 17
/dataset2/data1/data/IMAGE_VERSION	Attribute, Table 17
/dataset2/data2	Group
/dataset2/data2/what	Group
/dataset2/data2/what/quantity	Attribute, Table 16
/dataset2/data2/what/gain	Attribute, Table 13
/dataset2/data2/what/offset	Attribute, Table 13
/dataset2/data2/what/nodata	Attribute, Table 13
/dataset2/data2/what/undetect	Attribute, Table 13
/dataset2/data2/data	Dataset
/dataset2/data2/data/CLASS	Attribute, Table 17
/dataset2/data2/data/IMAGE_VERSION	Attribute, Table 17

7.2 Cartesian image and composite

The following example shows how an image is represented. It also contains the standard HDF5 mechanisms for including a palette, assuming the image payload is 8-bit.

Table 19: Cartesian image with palette

Node	Type
/	Root Group
/Conventions	Attribute, Section 4.1
/what	Group
/what/object	Atttribute, Table 2
/what/version	Atttribute, Table 1
/what/date	Atttribute, Table 1
/what/time	Atttribute, Table 1
/what/source	Atttribute, Table 3
/where	Group
/where/projdef	Attribute, Table 5
/where/xsize	Attribute, Table 5
/where/ysize	Attribute, Table 5
/where/xscale	Attribute, Table 5
/where/yscale	Attribute, Table 5
/where/LL_lon	Attribute, Table 5
/where/LL_lat	Attribute, Table 5
/where/UL_lon	Attribute, Table 5
/where/UL_lat	Attribute, Table 5
/where/UR_lon	Attribute, Table 5
/where/UR_lat	Attribute, Table 5
/where/LR_lon	Attribute, Table 5
/where/LR_lat	Attribute, Table 5

continued on next page

continued from previous page

Node	Type
/dataset1	Group
/dataset1/what	Group
/dataset1/what/product	Attribute, Table 14
/dataset1/what/prodpar	Attribute, Table 15
/dataset1/what/quantity	Attribute, Table 16
/dataset1/what/startdate	Attribute, Table 13
/dataset1/what/starttime	Attribute, Table 13
/dataset1/what/enddate	Attribute, Table 13
/dataset1/what/endtime	Attribute, Table 13
/dataset1/what/gain	Attribute, Table 13
/dataset1/what/offset	Attribute, Table 13
/dataset1/what/nodata	Attribute, Table 13
/dataset1/what/undetect	Attribute, Table 13
/dataset1/data1	Group
/dataset1/data1/data	Dataset
/dataset1/data1/data/CLASS	Attribute, Table 17
/dataset1/data1/data/IMAGE_VERSION	Attribute, Table 17
H5IM optional attributes	
/dataset1/data1/data/PALETTE	Link to /dataset1/data1/palette
/dataset1/data1/data/IMAGE_SUBCLASS	Attribute
/dataset1/data1/palette	Dataset
/dataset1/data1/palette/CLASS	Attribute
/dataset1/data1/palette/PAL_VERSION	Attribute

7.3 Vertical profile

Since the vertical profile is a general object, this simple example is of a conventional WRWP containing only height, wind speed, and wind direction at each point along the profile.

Table 20: Vertical profile

Node	Type
/	Root Group
/Conventions	Attribute, Section 4.1
/what	Group
/what/object	Atttribute, Table 2
/what/version	Atttribute, Table 1
/what/date	Atttribute, Table 1
/what/time	Atttribute, Table 1
/what/source	Atttribute, Table 3
/where	Group
/where/lon	Attribute, Table 7
/where/lat	Attribute, Table 7
/where/height	Attribute, Table 7

continued on next page

continued from previous page

Node	Type
/where/levels	Attribute, Table 7
/where/interval	Attribute, Table 7
/where/minheight	Attribute, Table 7
/where/maxheight	Attribute, Table 7
/dataset1	Group
/dataset1/what	Group
/dataset1/what/product	Attribute, Table 14
/dataset1/what/startdate	Attribute, Table 13
/dataset1/what/starttime	Attribute, Table 13
/dataset1/what/enddate	Attribute, Table 13
/dataset1/what/endtime	Attribute, Table 13
/dataset1/data1	Group
/dataset1/data1/what	Group
/dataset1/data1/what/quantity	Attribute, Table 16
/dataset1/data1/what/gain	Attribute, Table 13
/dataset1/data1/what/offset	Attribute, Table 13
/dataset1/data1/what/nodata	Attribute, Table 13
/dataset1/data1/what/undetect	Attribute, Table 13
/dataset1/data1/data	Dataset
/dataset1/data2	Group
/dataset1/data2/what	Group
/dataset1/data2/what/quantity	Attribute, Table 16
/dataset1/data2/what/gain	Attribute, Table 13
/dataset1/data2/what/offset	Attribute, Table 13
/dataset1/data2/what/nodata	Attribute, Table 13
/dataset1/data2/what/undetect	Attribute, Table 13
/dataset1/data2/data	Dataset
/dataset1/data3	Group
/dataset1/data3/what	Group
/dataset1/data3/what/quantity	Attribute, Table 16
/dataset1/data3/what/gain	Attribute, Table 13
/dataset1/data3/what/offset	Attribute, Table 13
/dataset1/data3/what/nodata	Attribute, Table 13
/dataset1/data3/what/undetect	Attribute, Table 13
/dataset1/data3/data	Dataset

7.4 Cross-section and side-panel

Table 21: Cross section

Node	Type
/	Root Group
/Conventions	Attribute, Section 4.1

continued on next page

continued from previous page

Node	Type
/what	Group
/what/object	Atttribute, Table 2
/what/version	Atttribute, Table 1
/what/date	Atttribute, Table 1
/what/time	Atttribute, Table 1
/what/source	Atttribute, Table 3
/where	Group
/where/xsize	Attribute, Table 6
/where/ysize	Attribute, Table 6
/where/xscale	Attribute, Table 6
/where/yscale	Attribute, Table 6
/where/start_lon	Attribute, Table 6
/where/start_lat	Attribute, Table 6
/where/stop_lon	Attribute, Table 6
/where/stop_lat	Attribute, Table 6
/where/minheight	Attribute, Table 6
/where/maxheight	Attribute, Table 6
/dataset1	Group
/dataset1/what	Group
/dataset1/what/product	Attribute, Table 14
/dataset1/what/quantity	Attribute, Table 16
/dataset1/what/startdate	Attribute, Table 13
/dataset1/what/starttime	Attribute, Table 13
/dataset1/what/enddate	Attribute, Table 13
/dataset1/what/endtime	Attribute, Table 13
/dataset1/what/gain	Attribute, Table 13
/dataset1/what/offset	Attribute, Table 13
/dataset1/what/nodata	Attribute, Table 13
/dataset1/what/undetect	Attribute, Table 13
/dataset1/data1	Group
/dataset1/data1/data	Dataset
/dataset1/data1/data/CLASS	Attribute, Table 17
/dataset1/data1/data/IMAGE_VERSION	Attribute, Table 17

7.5 RHI

Table 22: Range-height indicator

Node	Type
/	Root Group
/Conventions	Attribute, Section 4.1
/what	Group
/what/object	Atttribute, Table 2
/what/version	Atttribute, Table 1

continued on next page

continued from previous page

Node	Type
/what/date	Atttribute, Table 1
/what/time	Atttribute, Table 1
/what/source	Atttribute, Table 3
/where	Group
/where/xsize	Attribute, Table 6
/where/ysize	Attribute, Table 6
/where/xscale	Attribute, Table 6
/where/yscale	Attribute, Table 6
/where/lon	Attribute, Table 6
/where/lat	Attribute, Table 6
/where/az_angle	Attribute, Table 6
/where/range	Attribute, Table 6
/dataset1	Group
/dataset1/what	Group
/dataset1/what/product	Attribute, Table 14
/dataset1/what/quantity	Attribute, Table 16
/dataset1/what/startdate	Attribute, Table 13
/dataset1/what/starttime	Attribute, Table 13
/dataset1/what/enddate	Attribute, Table 13
/dataset1/what/endtime	Attribute, Table 13
/dataset1/what/gain	Attribute, Table 13
/dataset1/what/offset	Attribute, Table 13
/dataset1/what/nodata	Attribute, Table 13
/dataset1/what/undetect	Attribute, Table 13
/dataset1/data1	Group
/dataset1/data1/data	Dataset
/dataset1/data1/data/CLASS	Attribute, Table 17
/dataset1/data1/data/IMAGE_VERSION	Attribute, Table 17

EUMETNET OPERA weather radar information model for implementation with the HDF5 file format – an approach to UML modeling –

1. Introduction

This document presents an application of the Unified Modeling Language (UML) by presenting the Opera weather radar information model in UML diagrams. This information model has been developed within the framework of EUMETNET OPERA Work Package 2.1a. The subsequent work package 2.1b, aiming on operational use, will support the information model by providing additional documentation and tooling.

UML is a standardized general-purpose modeling language used in the field of software engineering. It includes a set of graphical notation techniques to create abstract models of specific systems. UML is especially suitable for modeling object-oriented systems. The aim of this document is to show that the Opera Data Information Model can be approached as a dynamic, object oriented structure, and by presenting it in UML diagrams to show this in full extent.

Moreover this document can be used as an (very concise) UML tutor, To clarify the used notations a look on http://en.wikipedia.org/wiki/Class_diagram will be useful.

2. Using class concept for radar data information model

The general purpose of modeling the information model in UML is to show the object oriented nature of the information model by presenting the elements of the model as components of an object-oriented system. In this approach, elements of the information model will be represented by classes. Diagram 1 shows the *root* element of the information model marked with a slash character. In terms of object oriented modeling, the *root* element represents a class. A class consists of two types of elements: fields and methods. Fields (which are also referred to as properties or attributes) constitute a structure of data belonging to the particular class.

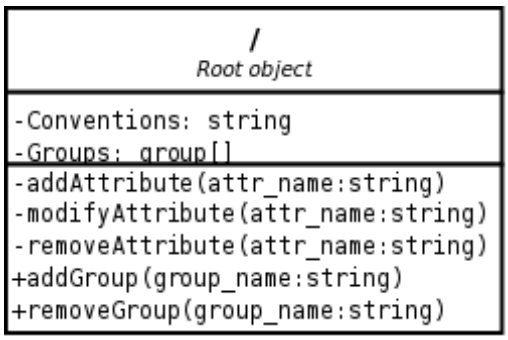

Diagram 1: Root class

Fields may store constants, variables, pointers or references (depending on the programming language) representing basic and compound data types (including references to other classes). Methods define the behavior of the class by implementing functions which operate on the data belonging to that class and other data. The methods belonging to the class define its interface.

The information model is based on the elements of HDF5 file structure, such as *groups*, *datasets* and *attributes*. In an object-oriented approach these elements, defining the data structure of the class, are called Fields. Functions operating on these elements are called Methods.

By definition, *root* class contains two fields:

-Conventions: string
-Groups: group[]

The *Conventions* field is preceded with a minus sign thus stating, according to the UML standard, private access mode. This means this field can be accessed by methods belonging to the *root* class only. The *root* class defines the following methods for this task:

-addAttribute(attr_name: string)
-modifyAttribute(attr_name: string)
-removeAttribute(attr_name: string)

These methods allow for adding and removing the *Conventions* attribute, as well as for modifying its value. The methods are private, therefore available for the objects of this class only. The methods take the attribute name as parameter.

According to the information model, the *root* element is a container for other group objects. The number of *groups* defined in *root* element can vary and depends on the number of *datasets* stored in the file. To achieve such flexibility, the *root* class defines the *Groups* attribute, which is an array of references to further *group* objects. In order to access this field and modify the array, *root* class interface has to be complemented by the following 2 methods:

+addGroup(group_name: string)
+removeGroup(group_name: string)

The above methods take *group* name as parameter and allow for adding and removing references to *group* objects. These methods are public and can be accessed by objects of this class as well as by objects of any class which inherits from the *root* class.

It is important to notice that the *root* object is treated as top-level *group*. This means that *group* objects may store *attributes* as well as references to other *groups*. In this case *root* object is a generalized *group* class, consisting of 2 private fields and defining a public interface which will be implemented by classes derived from the root class.

Diagram 2 shows the diagram of top-level *what group* class, formed by the following 5 fields:

-object: string
-version: string
-date: string
-time: string
-source: string

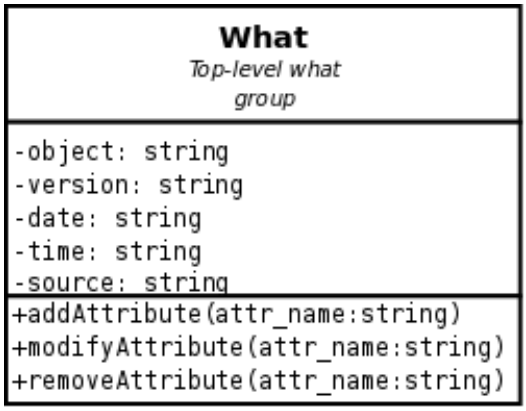

Diagram 2: Top-level what class

All the fields in this class are private (access mode marked with minus sign) that is available only for methods belonging to this class. The top-level *what* class defines an interface consisting of the following public methods:

+addAttribute(attr_name: string)
+modifyAttribute(attr_name: string)
+removeAttribute(attr_name: string)

3. Generalization / inheritance

This section shows how generalization and inheritance mechanisms are used to describe relations between objects of particular classes defined in the information model.

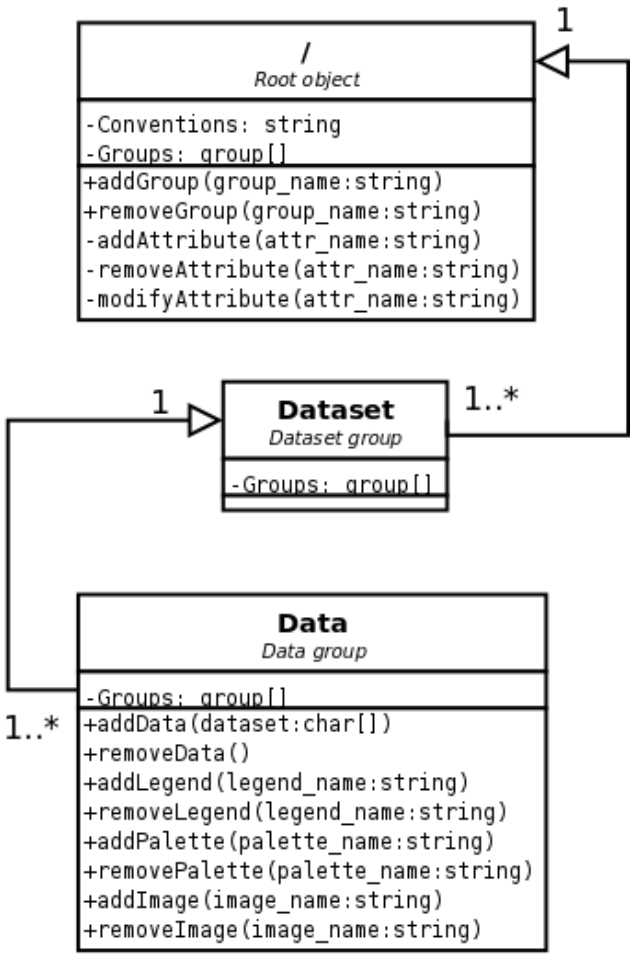

Diagram 3: Group class inheritance

The diagram shows relations between r*oot, dataset* and *data* class objects. In terms of the information model, these classes represent *groups*. As a part of its interface, the *root group* defines public methods that allow for adding and removing references to other *group* objects. According to the information model, *dataset* and *data groups* can store references to further *groups*. Therefore it is necessary that each of these *groups* implements the same interface allowing to add or remove references from the private array *Groups*. This can be achieved by inheriting the above methods from *root group*.

According to the information model, the *dataset group* is stored in the *root* object. In terms of the model it means that a reference to the *dataset group* is set in the *Groups* field of the *root* object. The *Dataset group* defines one private field and inherits the interface from *root* class. Because only the public methods are inherited, *dataset group* implements only methods allowing for operating adding and deleting references from *Groups* array, that is: addGroup() and removeGroup().

According to the assumptions of the information model, *dataset group* cannot store attributes. In other words, only groups can be stored in *dataset group*. This explains why not all of the methods defined in the *root group* are inherited.

the *Data group* in turn inherits from the *dataset* class. Although the *dataset* class does not define its own methods, it inherits public methods from the *root* class as described above. These methods are now inherited by *data* class. In addition, the *data* class defines several methods which complement its interface.The arrows connecting the classes indicate generalization. The notation at each end of the arrow indicates the multiplicity relation between objects of these classes. As indicated, *root group* can store reference to one or more *dataset groups*, and *dataset group* can be assigned to one *root group,* which is an example of one-to-many relation. The same applies to relation between *dataset* and *data* class objects: *dataset* object can store references to one or more *data groups*, *data group* is assigned to one *dataset* only.

4. Association

The information model defines two types of relation between objects. The previous example illustrates inheritance relationship between the root group, dataset and data group. The other type of relation is association.

Diagram 4 shows the association between the root group and the top-level what group. The notation at each end of the line connecting class diagrams indicates the multiplicity relation between objects of these classes. As indicated, root group can store reference to exactly one top-level what group. The relation is the same in the opposite direction – top-level what group can be assigned to one root group. This is an example of one-to-one relation which is compliant with one of the basic assumptions of the information model: only one top-level what group is allowed.
In case of association there is no inheritance process. Although root class defines 2 public methods, these methods will not be available in the associated what class. Therefore it is necessary that top-level what class defines its own interface in order to provide methods operating on its attributes.

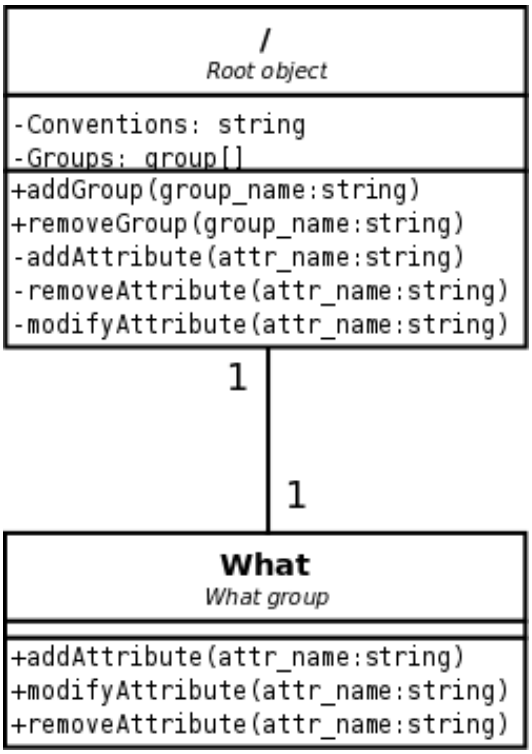

Diagram 4: Root class and associated top-level what class

This illustrates the mechanism that works on all top-level groups, what, where and how. All these groups implement the same interface and keep the same one-to-one multiplicity relation.

5. Basic class diagram of the information model

In the 5th diagram a basic class diagram for the information model is shown. Class diagrams are simplified in order to provide general overview on relationship between classes.

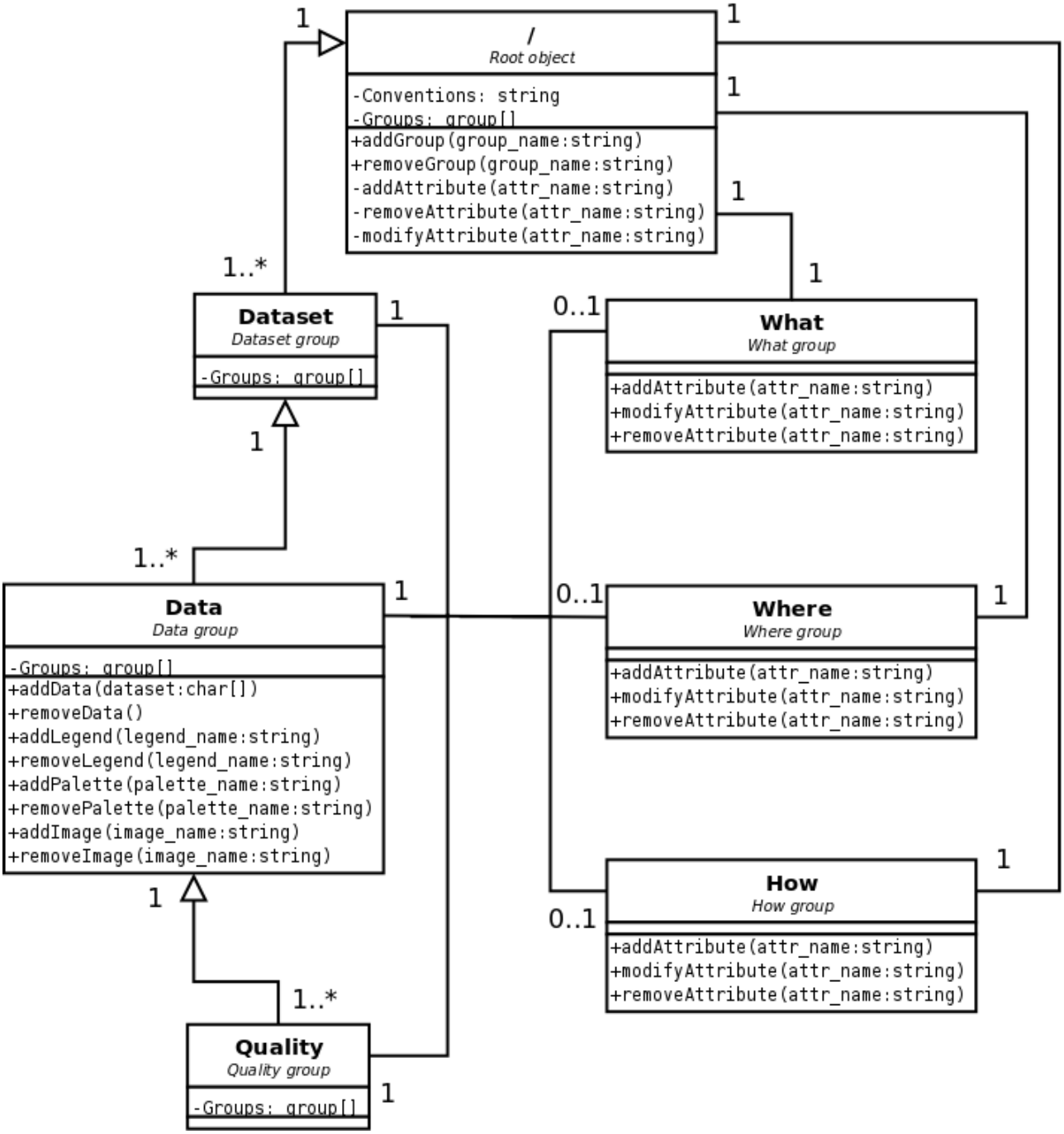

Diagram 5: Basic class diagram for ODIM_H5 2.0 radar data information model

According to the model, the *root group* represents the top-level class in the hierarchy. The *Root group* is a container which holds one attribute and a given number of references to other *groups*, depending on data type.

The diagram shows two basic types of relationship between classes. The first type is inheritance. The *Dataset* class is a child class of *Root* and inherits its public interface. The *Data* class on its turn is a child of *Dataset*. The *Data* class inherits its public interface from *Dataset* and defines its own public methods that allow acces to the actual data of different types. *Quality* class is the last one in inheritance chain. This class does not introduce its specific methods, but it inherits all the public methods available in the *Data* class. Numbers at the ends of each arrow determine multiplicity relation between classes. As shown in the diagram, one or more *Dataset* class objects can be assigned to the *Root* class, but the *Dataset* class is assigned only to the *root* class. In terms of HDF5 objects, *root group* can store one ore more *Dataset groups*. The same type of relationship is found at each level of the hierarchy, that is between *Dataset* and *Data* classes, as well as between *Data* and *Quality*.

The second type of relationship is association: there are 3 classes associated with *root* class: these are top-level *What, Where* and *How* classes. Each of the above classes define its own interface, dealing with the attributes. As indicated in the diagram, no inheritance process takes place here. Multiplicity relation is defined by the model: there can be only one top-level class of each type assigned to the *root* class. In model terminology it means that *root group* can store one top-level *group* of each type: *What, Where* and *How*.

A key property of the information model is that *What, Where* and *How groups* can occur at any level in the data structure. In other words, the attributes stored in these *groups* can refer to any node in the HDF5 object's hierarchy. If, for instance, a *How group* is found in a *Dataset group*, its attributes are valid for each element available in this *Dataset group* (e.g. for any *Data group* here). The above diagram reflects this possibility by defining relationships between *What, Where* and *How* classes and the classes to which they can be assigned – that is *Dataset, Data* and *Quality* classes. As defined by the model, each of the classes - *Dataset, Data* and *Quality* – can have zero or one *What, Where* or *How* associated classes. These association relationships are illustrated by the lines connecting *Dataset, Data* and *Quality* class diagrams with *What, Where* and *How* classes.

6. Detailed class diagrams for *what, where* and *how groups*

What, Where and *How groups* may contain different set of attributes depending on the type of actual data they refer to and on the group's level in HDF5 object hierarchy. For instance, the set of mandatory *What group* attributes will be different for top-level *What group* and for *What group* stored at the *Dataset* or *Data* level. In the first case, top-level *What* contains a set of mandatory attributes which is universal for any kind of data and is always required in order to be able to correctly determine the file's content. Attributes of top-level *What group* have global scope and are valid for all datasets.

On the other hand the information model defines a *What group* which is specific to a given *Dataset*. Such *groups* contain attributes which are valid within a given *Dataset*. Therefore it is necessary to split the general *What* class into 2 subclasses: top-level *What* class and the *Dataset*-specific *What* class. Diagram 6 shows a *Dataset*-specific *What* defined as a child class of top-level *What*. It inherits public interface and introduces several private fields. Within the structure of the information model, one or several *Dataset*-specific *What groups* can be assigned to one top-level *What*. This relationship is described by symbols at each end of the arrow connecting class diagrams.

Something similar applies to the general *Where group, which* can be split into a set of classes, as shown in Diagram 7. On the top of the hierarchy there will be the top-level *Where* class defining global scope attributes. Several child classes can be derived from the top-level *Where, in* classes associated with given data type: polar scan data, sector data, vertical profile, Cartesian image and cross sections.

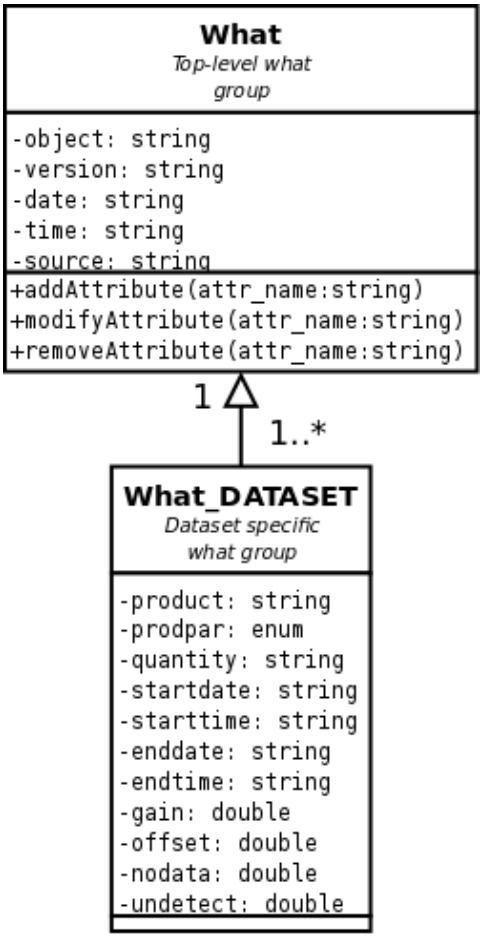

Diagram 6: Detailed class diagram for What group

Therefore the set of attributes encapsulated by each of these classes defines a data structure (a set of fields) which is unique for a given data object as defined by the information model.

There are two more elements of detailed *Where* class diagram. These are child classes of *Where_XSEC* class, associated with cross–section data. These classes encapsulate attributes specific to the RHI scan (Range – Height Indicator) and side panel data. *Where_XSEC* class defines a set of public attributes which are common for cross-section data. These attributes are inherited by *Where_RHI* and *Where_PANEL* child classes.

The *How group* contains attributes which provide additional and complimentary information which can be used to describe a given dataset object, for example information related to an object's quality. Again, this *group* can be represented by a set of related classes that define attributes specific for a given data object (Diagram 8).

The top-level *How* class contains attributes valid for any type of data. It is referenced from *Root* class and its attributes have global scope. All attributes in this class are public and are inherited by any derived class. There are 3 child classes describing particular types of data: polar volumes, Cartesian images and vertical profiles, each class encapsulates a set of attributes specific to a given type of data. According to the model, zero or more objects of these classes can be assigned to the top-level *How* class object. For instance, a HDF5 data structure containing simple polar scan with one dataset only, will implement top-level *How* class only. Structure containing several *datasets*, e.g. polar scan with data acquired with 10 antenna elevations, may *contain How* class objects referring individually to each of these datasets. This relationship between top-level *How* and its child classes is described by appropriate symbols at each end of inheritance arrow.

The last element of the detailed *How* class diagram is *How_Q* class. This class is designed for the purpose of storing attributes related to quality *only. How_Q* is associated with top-level *How* and does not inherit any elements. It defines private fields allowing to store quality factors and private methods which operate on these attributes. Based on the assumptions of ODIM_H5 2.0 model, quality attributes are global and affect all *datasets* stored in HDF5 data structure. Therefore the top-

level *How* class can be associated with zero or one *How_Q* class.

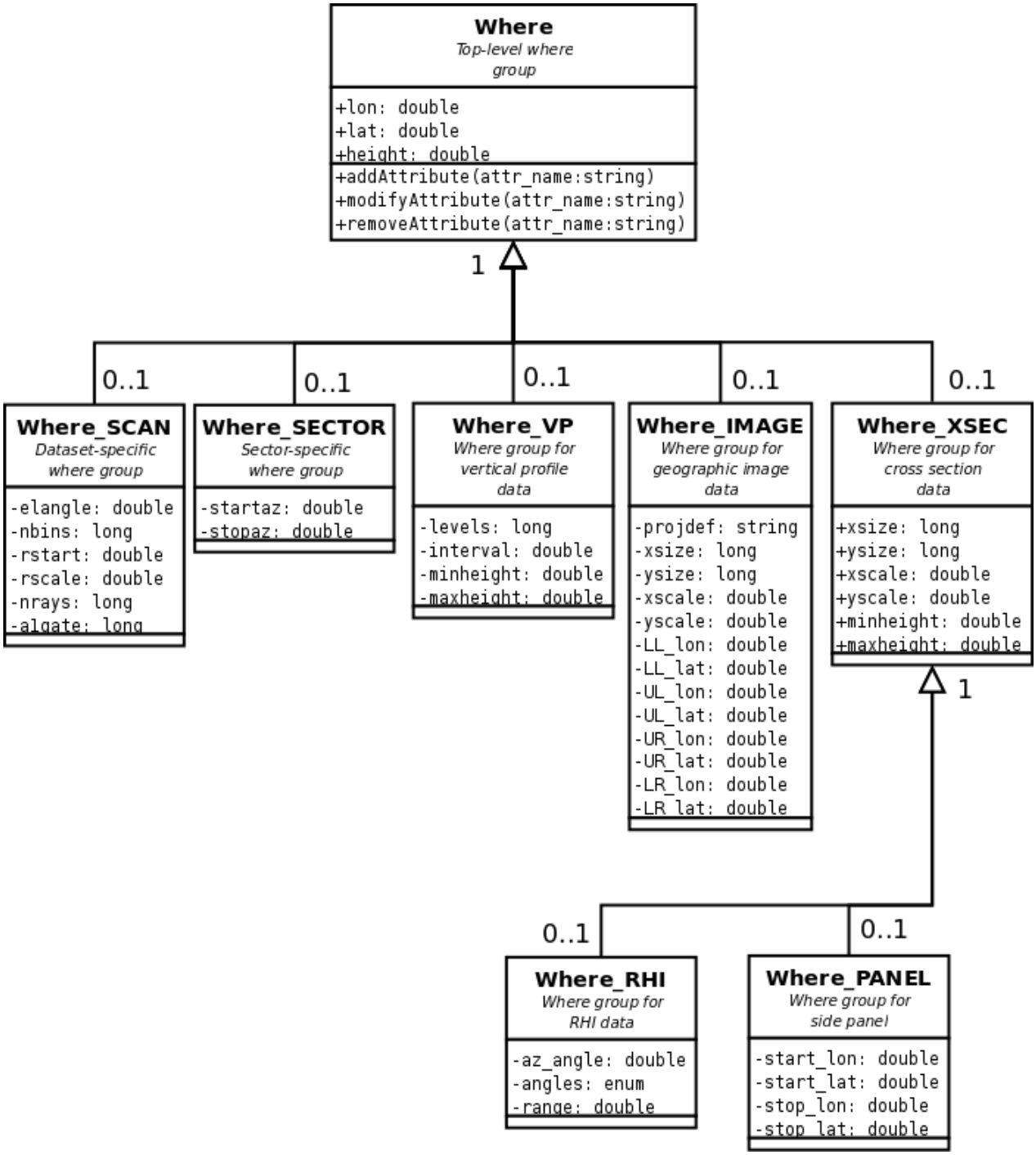

Diagram 7: Detailed class diagram for Where group

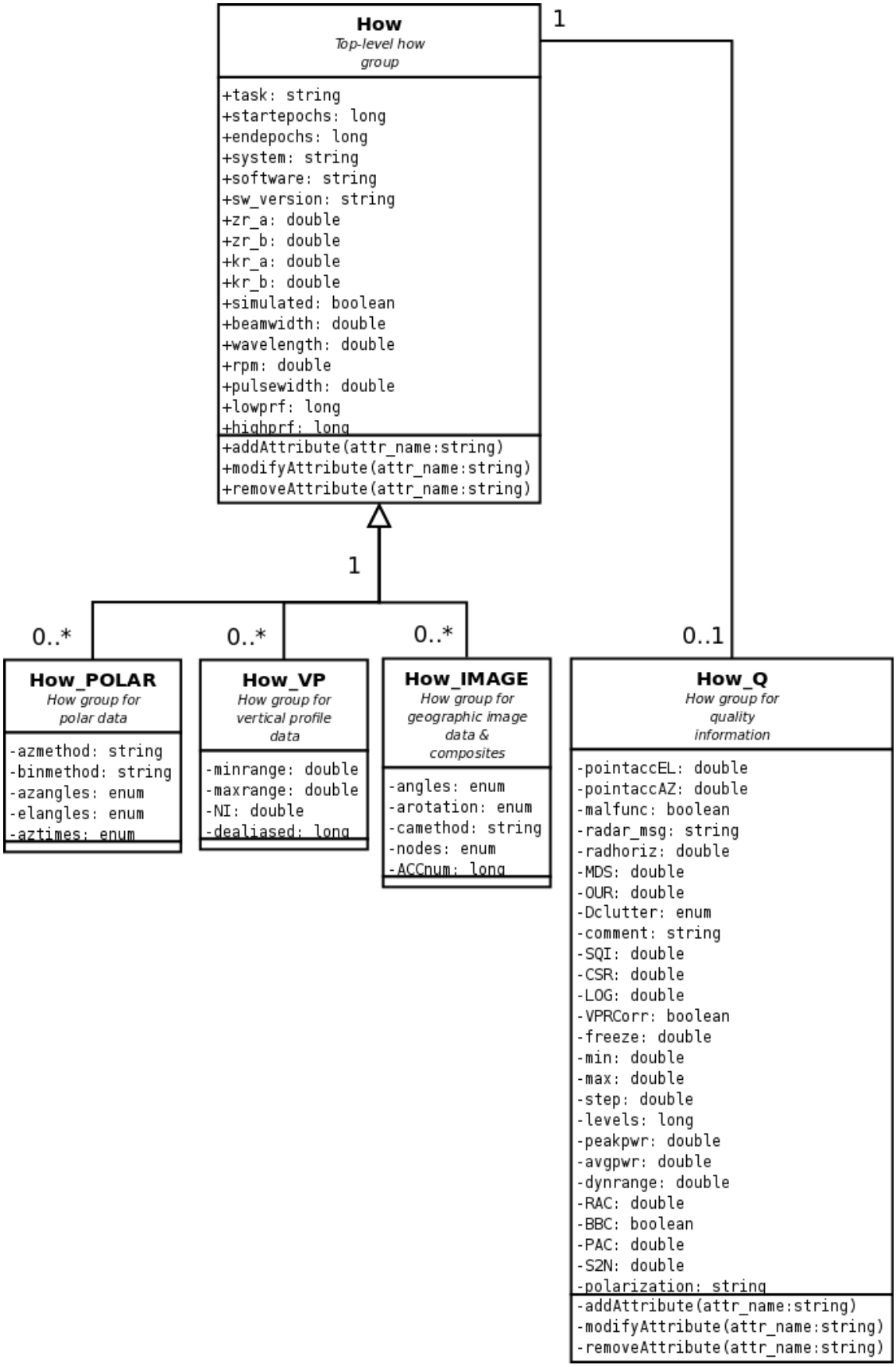

Diagram 8: Detailed class diagram for How group