

Report on HDF access revisited Framework



Eng. Daniele Romagnoli
Eng. Simone Giannecchini

Contents

| | | |
|----------|--|-----------|
| 1 | Scope of this document | 3 |
| 2 | HDF files access: H4File | 3 |
| 3 | Scientific Datasets Access: H4SDSCollection | 4 |
| 3.1 | Scientific Dataset: H4SDS | 6 |
| 3.1.1 | SDS dimension: H4Dimension | 8 |
| 4 | Images Access: H4GRImageCollection | 10 |
| 4.1 | General Raster Image: H4GRImage | 11 |
| 4.1.1 | Image palette: H4Palette | 13 |
| 5 | Group structure access: H4VGroupCollection | 14 |
| 5.1 | VGroup: H4VGroup | 16 |
| 6 | Annotations Access: H4AnnotationManager | 17 |
| 6.1 | Annotation: H4Annotation | 17 |
| 7 | Other classes | 20 |
| 7.1 | Parent class: AbstractHObject | 20 |
| 7.2 | HDF Object reference: H4ReferencedObject | 20 |
| 7.3 | HDF Object with Attributes: H4DecoratedObject | 21 |
| 7.3.1 | Attribute: H4Attribute | 22 |
| 7.4 | HDF variables: H4Variable | 24 |
| 7.5 | HDF datatypes and data allocation: H4DatatypeUtilities | 24 |
| 8 | Limitations | 25 |

List of Figures

| | | |
|----|--|----|
| 1 | H4File class | 4 |
| 2 | H4SDSCollection class | 7 |
| 3 | Read Operation | 8 |
| 4 | H4SDS class | 9 |
| 5 | H4Dimension class | 10 |
| 6 | H4GRImageCollection class | 12 |
| 7 | H4GRImage class | 13 |
| 8 | H4Palette class | 14 |
| 9 | H4VGroupCollection class | 15 |
| 10 | H4VGroup class | 16 |
| 11 | H4AnnotationManager class | 18 |
| 12 | H4Annotation class | 18 |
| 13 | Class Diagram | 19 |
| 14 | AbstractHObject class and IHObject interface | 20 |
| 15 | H4ReferencedObject class and IH4ReferencedObject interface | 21 |
| 16 | H4DecoratedObject class | 22 |
| 17 | H4Attribute class | 23 |
| 18 | H4Variable class | 24 |
| 19 | H4DatatypeUtilities class | 24 |

1 Scope of this document

This document will introduce some basic explanations about the revisited framework which will be used to access and manage HDF data sources. Actually, only HDF4 sources are supported, anyway, extending the framework to support HDF5 would be not a really difficult task.

It is worthwhile remarking that in this document, when talking about a HDF item, we are mainly referring to a HDF4 one.

Let us start introducing the main class which should be used whenever you need to access a HDF source.

2 HDF files access: H4File

By means of the `H4File`, we are able of retrieving SDS (Scientific Data Set), Images, Annotations, Attributes, Group structures and any other objects contained within a HDF source. Prior to perform any type of access operation with a HDF source, we need to instantiate a `H4File`, by providing the path where the HDF file is located.

As an instance:

```
final String filePath = new String (c:/HDFSamples/myData.hdf);  
H4File myFile = new H4File (filePath);
```

A `H4File` instance provides access capabilities by means of different classes which will be instantiated¹ and initialized only when their usage is requested², depending on the type of operations we need to perform or the type of data structures we need to access to. These classes are: `H4SDSCollection`, `H4GRImageCollection`, `H4VGroupCollection` and `H4AnnotationManager`. You can obtain access to the required class by means of the proper `H4File` getter method.

Prior to introduce these classes, it is worthwhile remarking that a HDF file may have some attached annotations, which will be introduced in section 6. By means of the `getAnnotations (int annotationType)` method we may retrieve a `List` containing all the annotations of the required type for

¹A single instance of each class within the same `H4File` instance

²this types of mechanism are also referred as lazy creation/lazy initialization

this file. The `H4File` class is depicted in Figure 1

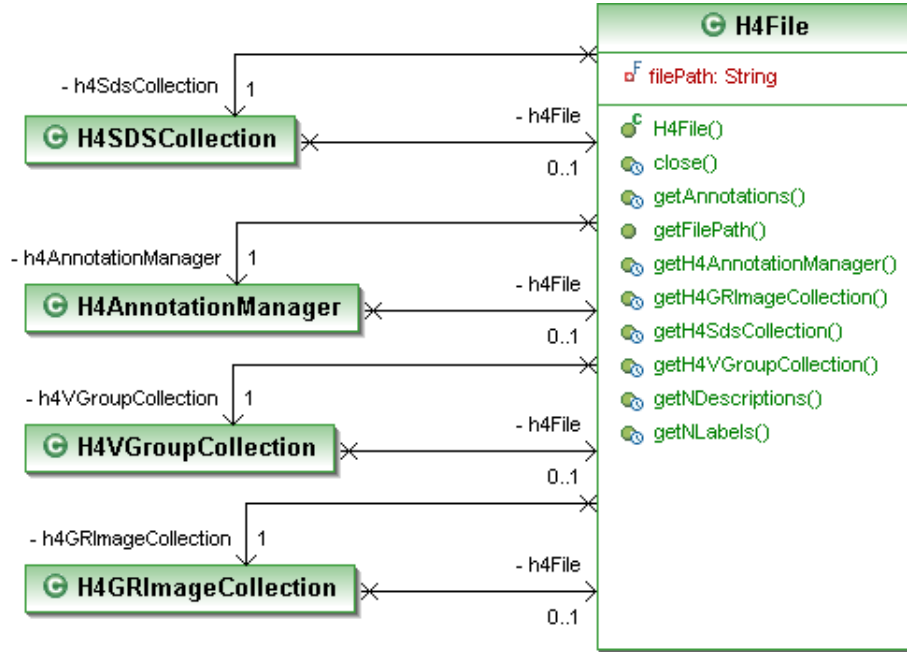


Figure 1: `H4File` class

Let us now introduce the class needed to access Scientific Datasets.

3 Scientific Datasets Access: `H4SDSCollection`

The `H4SDSCollection` class allows to get access to any SDS stored within the related source file leveraging on the underlying SD Interface³. Prior to proceed with further explanations, let us briefly introduce the SDS data model.

The SDS data model requires several mandatory components, which describe a scientific dataset. They are:

³In such a context, the term "Interface" has nothing to do with the Java "Interface" element. The SD interface is a set of API which allow to access to the Scientific Dataset stored in the underlying HDF source. A Java Interface is instead an abstract type which allows to specify interaction with the outside world by exposing a set of method signatures. All classes implementing the interface need to specify a body for these methods

- the *name* of the SDS
- the *dimensions* of the SDS (its size and shape⁴)
- the *data array*
- the *type* of data contained in the SDS array (the datatype)

Furthermore, a SDS may also contains optional components:

- *attributes*
- *dimension scales*⁵

Basically, the `H4SDSCollection` allows to:

- retrieve the number of SDS contained within the source
- manage the attributes related to the whole set of SDS
- get access to a specific SDS

It is worth to point out that when a dimension scale is set for a dimension of a SDS, it is internally implemented as an SDS⁶. However, the `H4SDSCollection` only provides access to the SDS representing a real Scientific Data Set⁷. The way of querying and managing dimension and dimension scales will be discussed afterwards in section 3.1.1.

Let us now briefly provide some explanations about how the `H4SDSCollection` works in order to better understand how to manage SDS.

When initialized, it performs a full scan of all the SDSs contained within the file and it checks whether a SDS represents a dimension scale. In case of a real SDS, a new `H4SDS` object will be created (but not initialized in order to avoid time and memory consumption). Then, access to the underlying SDS will be immediately closed to avoid having unnecessary SDSs opened. In

⁴With the term *size* we are referring to the rank of the SDS which represents the number of its dimensions. The *shape* represents the extent of the SDS along each dimension

⁵dimension scales are sequence of values placed along a related dimension to specify intervals along it

⁶The monodimensional SDS data array will contain the dimension scale values

⁷Furthermore, the number of SDS returned by the `size()` method of the collection does not take on account of the number of SDS representing dimension scales

such a context, it is worth to point out that each SDS contained in the HDF file has a unique identifier. For this reason a future "open operation" on the same SDS will return the same identifier. When you need to get access to a specific SDS you have to make a request to the **H4SDSCollection** by means of the `get`⁸ method. After such a request, the **H4SDSCollection** will return a **H4SDS** object properly opened and initialized.

The **H4SDSCollection** class is depicted in Figure 2 which also illustrates the main relationships with the previously introduced classes and some other classes which will be discussed afterwards. Let us now introduce the class representing and managing a Scientific DataSet.

3.1 Scientific Dataset: **H4SDS**

The **H4SDS** class, representing a SDS, allows to retrieve and manage all the components introduced in the SDS data model. First of all, it has a set of getter methods for accessing main properties of the represented SDS such as *name*, *datatype*, *rank* and *dimensions sizes*, as well as its *index*⁹.

Through the **H4SDS** class you can also manage the attributes and the available annotations of the SDS. Attributes will be introduced in 7.3 and annotations will be introduced in section 6.

Obviously, **H4SDS** also allows to perform read operations to load data values stored in the SDS data array. You have to specify the required starting points, the sizes of the required portion of data and the strides along each dimension of the SDS by means of the parameters **start**, **edge**, **stride** where:

- **start[i]** represents the first value which will be loaded along the i-th dimension
- **edge[i]** represents the number of values which will be loaded along the i-th dimension

⁸You are allowed to specify both the name of the required SDS or the index within the collection. When specifying the index, remember that the range of supported ones starts from 0 to (**size**-1) where, as stated before, **size** does not take on account of the SDS related to dimensions scales.

⁹It is worth to point out that the index field does not represent the position of the SDS within the SDS list stored by the **H4SDSCollection** but it represents the index of the SDS within the HDF source file

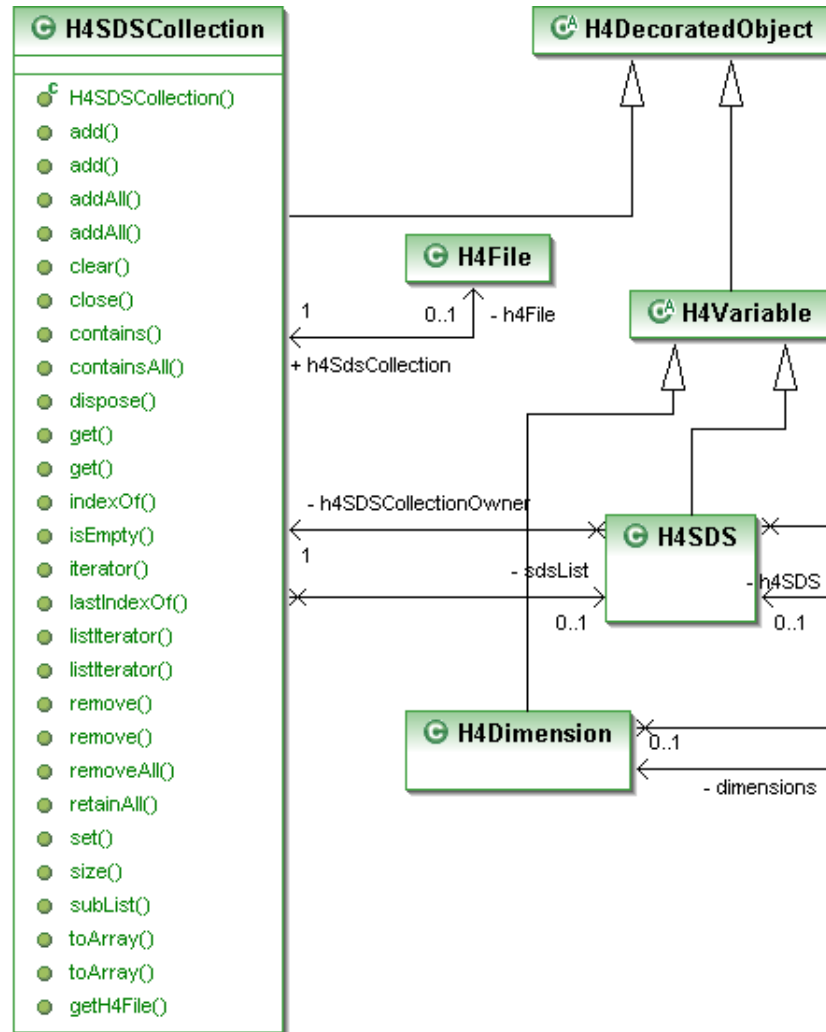


Figure 2: H4SDSCollection class

- `stride[i]` represents the interval between the values which will be loaded along the i-th dimension

A proper customization of these parameters allows read operations with subsampling/subregion-selection mechanisms.

Let us provide an example of a parametrized read operation. In Figure 3 is depicted the set of values which will be loaded with a read operation of a SDS having rank = 2 and dimension sizes = [12,12]. Set parameters are: `start[0]=0; start[1]=1; edge[0]=6 edge[1]=4; stride[0]=2; stride[1]=3`

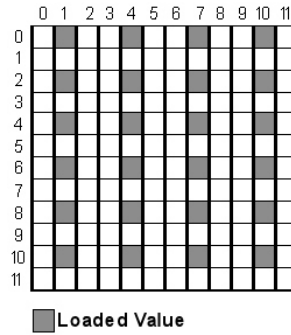


Figure 3: Read Operation

Finally, *H4SDS* allows to retrieve and query dimensions of the SDS which are represented by instances of the *H4Dimension* class. The *H4SDS* class is depicted in Figure 4.

3.1.1 SDS dimension: *H4Dimension*

Each instance of the *H4Dimension* class stores properties of a dimension of the owner SDS. They are: *name*, *size*, *index* and *number of attributes*. You can retrieve these properties by means of the proper getter method.

As previously stated, a dimension may have a set dimension scale. You can query the *H4Dimension* to know whether a dimension scale exists for it. In case it exists, you can get the dimension scale values by means of the `getDimensionScaleValues()` method which returns a Java *Object*¹⁰ containing the values.

The *H4Dimension* class is depicted in Figure 5.

¹⁰a proper data array depending on the datatype of the dimension scale

3.1 Scientific Data SCIENTIFIC DATASETS ACCESS: H4SDSCOLLECTION

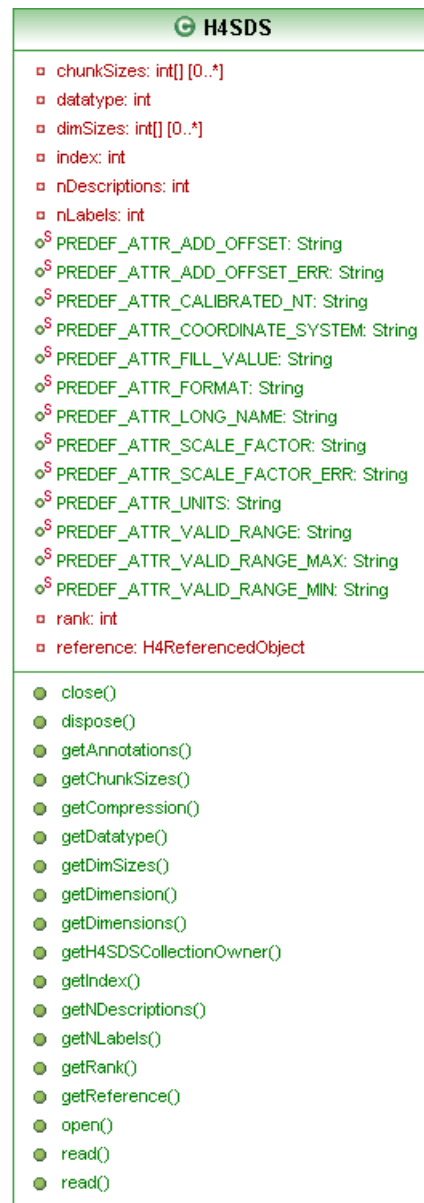


Figure 4: H4SDS class

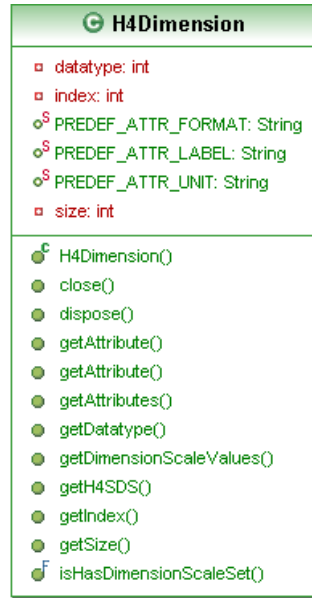


Figure 5: H4Dimension class

Let us now introduce the class used to access images stored within a HDF file.

4 Images Access: H4GRImageCollection

The `H4GRImageCollection` class almost provides the same capabilities of the `H4SDSCollection` class, with the difference that it relies on General Rasters Images instead of SDS.

Prior to proceed with further explanations, let us briefly introduce the General Raster data model.

The GR data model requires several mandatory components describing a General Raster. They are:

- the *name* of the Image
- the *dimensions* size of the Image (which is always 2D)
- the *image array* (a 2D array of pixels)

- the *pixel type* of the Image

Finally, a GR may also contains optional components:

- *attributes*
- *palettes*¹¹

Basically, the **H4GRImageCollection** allows to:

- retrieve the number of Images contained within the HDF source
- manage the attributes related to the whole set of Images (also called file attributes since they are not referred to a specific image).
- get access to a specific GR Image

To require the desired GR Image you need to use the **get**¹² method. After such a request, the **H4GRImageCollection** will return a **H4GRImage** object.

The **H4GRImageCollection** class is depicted in Figure 6 which also illustrates the main relationships with the previously introduced classes and some other classes which will be discussed afterwards. Let us now introduce the class representing and managing a General Raster Image.

4.1 General Raster Image: **H4GRImage**

The **H4GRImage** class represents a GR Image allowing to retrieve and manage all components introduced in the GR data model. First of all, it has a set of getter methods returning the main properties of the represented Image such as *name*, *pixel type*¹³, *interlace mode* and *dimensions sizes*, as well as the *index* of the Image within the source file. Through the **H4GRImage** class you can also manage the attributes and the available annotations of the Image. Obviously, **H4GRImage** also allows to perform read operations to load data values stored in the pixel array. Similarly to the **H4SDS**'s read operation, you have to specify the required starting points, the sizes of required data and

¹¹palettes are lookup tables used to define a set of color values for each pixel value of the image

¹²You are allowed to specify both the name of the required image or the index within the collection

¹³the pixel type is described by the number of components composing the pixel (as an instance, RGB) and the data type of pixel

4.1 General Raster Image: ~~H4GRImage~~ ~~H4GRImage~~ ACCESS: H4GRIMAGECOLLECTION

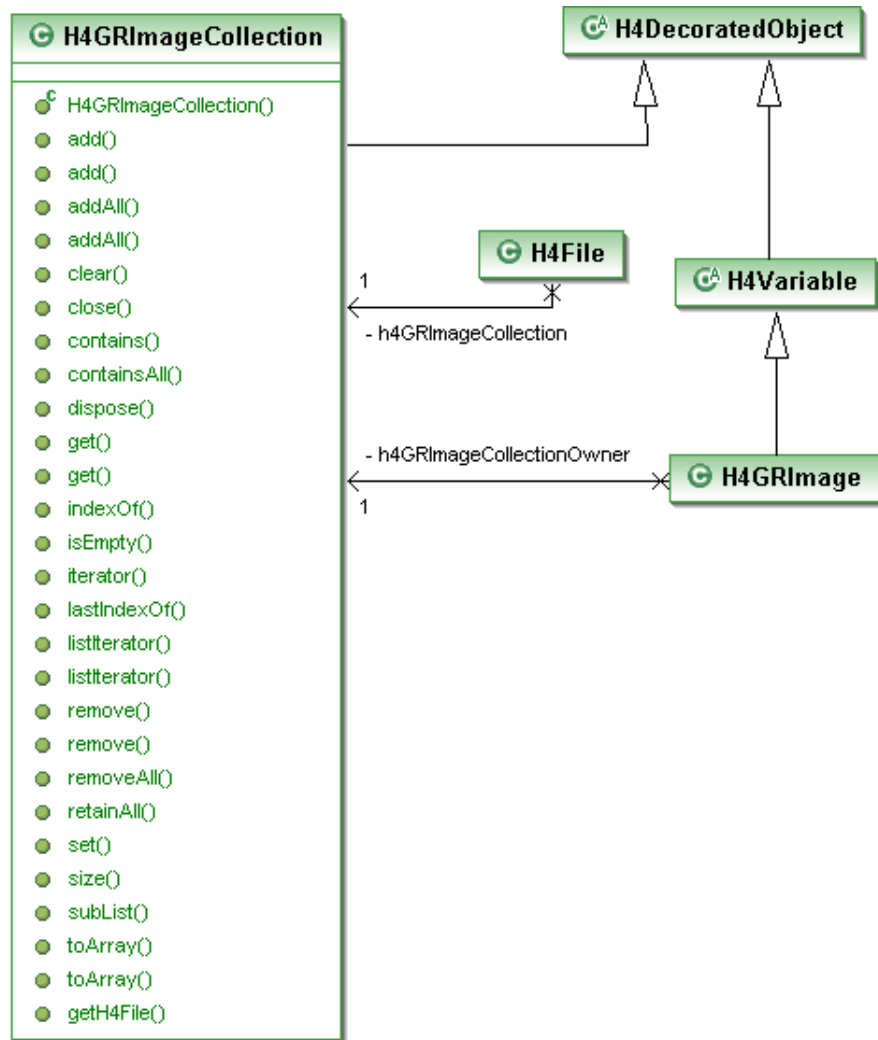


Figure 6: H4GRImageCollection class

the strides along the 2 dimensions.

Finally **H4GRImage** allows to retrieve the number of palettes and in case they exist, get access to a specific one (by means of the **getH4Palette** method) which is represented by an instance of **H4Palette**. The **H4GRImage** class is depicted in Figure 7.



Figure 7: **H4GRImage** class

4.1.1 Image palette: **H4Palette**

Each instance of the **H4Palette** class stores properties about a palette of the Image and provides access to the palette values. *Number of components,*

number of entries, *interlace model* and *datatype* of the palette data are main property of a palette. To obtain the values of the palette you need to use the `getValues()` method. The `H4Palette` classes is depicted in Figure 8.

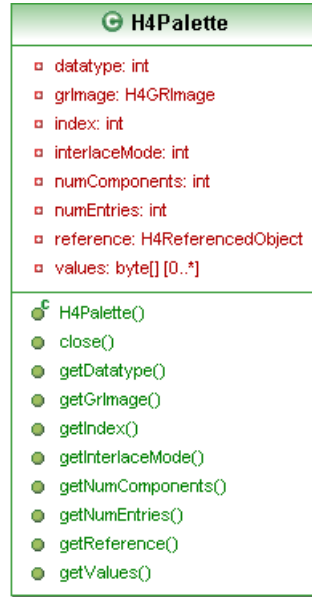


Figure 8: `H4Palette` class

Let us now introduce the class used to retrieve information about grouping structures stored within a HDF file.

5 Group structure access: `H4VGroupCollection`

The `H4VGroupCollection` class allows to get access to some specific `VGroups` contained within a HDF source. Since a `VGroup` may be children of several other `VGroups` and the relations between groups may be very complicated, this class only provides direct access to the top groups (the ones which have no fathers) also called *lone groups*. You can get access to a specific `VGroup` by means of the `get` method which returns a `H4VGroup` object.

The `H4VGroupCollection` class is depicted in Figure 9 which also illustrates the main relationships with the previously introduced classes and some other classes which will be discussed afterwards.

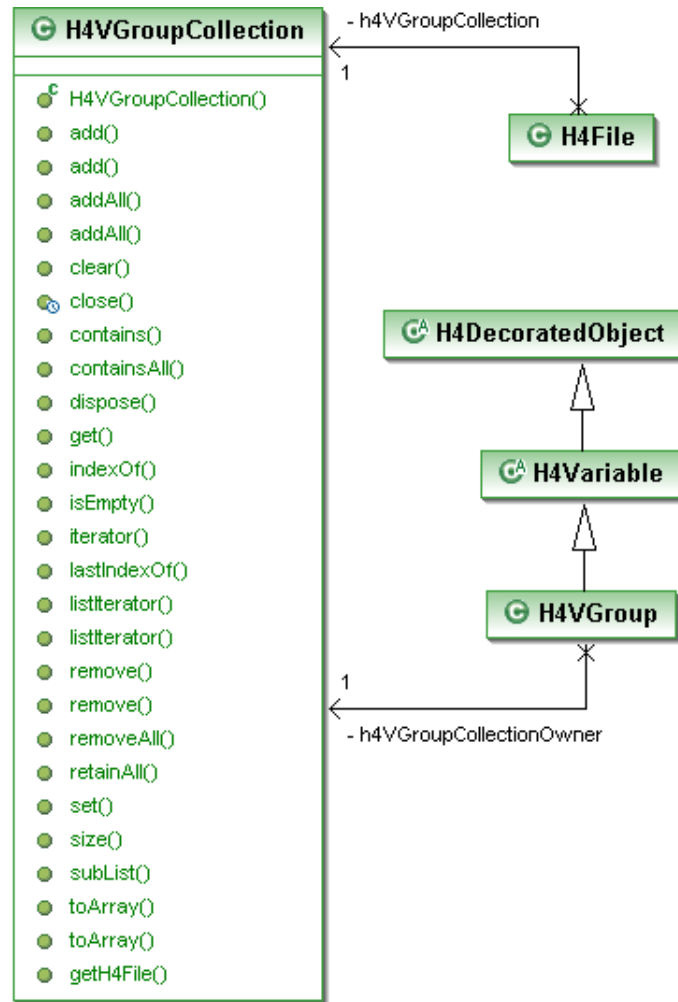


Figure 9: H4VGroupCollection class

5.1 VGroup: H4VGroup

The `H4VGroup` class, representing a `VGroup`, allows management and retrieval of the main properties of a `VGroup`. A `VGroup` has a *name* and, optionally, a *class*¹⁴ but it may also contain several other objects for which you can query the number by means of the `getNumObjects()`. Furthermore, by means of the `getTagRefList()`, you can retrieve the {tag,ref} couples list referencing to these objects.

The `H4VGroup` provides an additional capability. By means of the `isAVGroup` method, you can gain knowledge of whether an object referred by an element of the returned {tag,ref} list is a child `VGroup` of a parent one and then, build a new `H4VGroup` given the parent and the reference of the child. This capability is achieved by means of the specialized constructor `H4VGroup(H4VGroup parentGroup, int ref)`.

Finally, you can also manage the set of attributes related to the `VGroup`. The `H4VGroup` class is depicted in Figure 10



Figure 10: `H4VGroup` class

¹⁴The class of a `VGroup` has nothing to do with the class concept of a object oriented programming language

Let us now introduce annotations as well as the main class involved when annotations access is required.

6 Annotations Access: *H4AnnotationManager*

An HDF annotation represents textual information attached to a specific HDF data object or a HDF file. There are 2 types of annotations: *labels* and *descriptions*. Labels are short annotations which are commonly used to set the title or similar things to a file or a data object while descriptions are more longer annotations which may contain more extensive information.

Basically, *H4AnnotationManager* allows to get information about the total number of different annotations contained within a HDF source (file labels/descriptions and total data object labels/descriptions). Furthermore, by means of this class, other previously introduced classes, such as *H4SDS*, *H4GRImage* or *H4File* are capable of retrieving annotations related to the represented object.

An HDF Annotation is represented by an instance of *H4Annotation* class.

6.1 Annotation: *H4Annotation*

The *H4Annotation* class basically stores annotation properties which are *type* (expressed by an integer specifying if this annotation is a File label or a File description or a DataObject label or a DataObject description), *TAG* and *Reference*, and obviously the textual *content* of the annotation itself.

The *H4AnnotationManager* and *H4Annotation* classes are depicted in Figure 11 and Figure 12 respectively.

Let us now introduce a slightly more detailed description of the class hierarchy as well as the remaining classes and interfaces of the framework which have not yet been discussed. The class diagram of the whole framework is depicted in Figure 13.

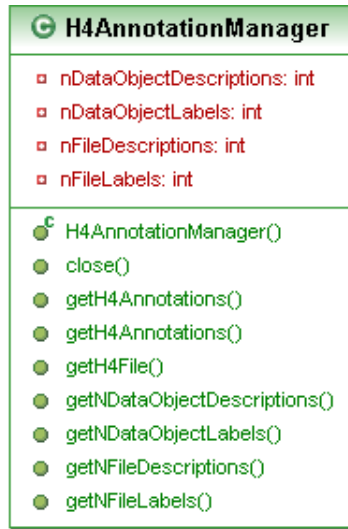


Figure 11: H4AnnotationManager class

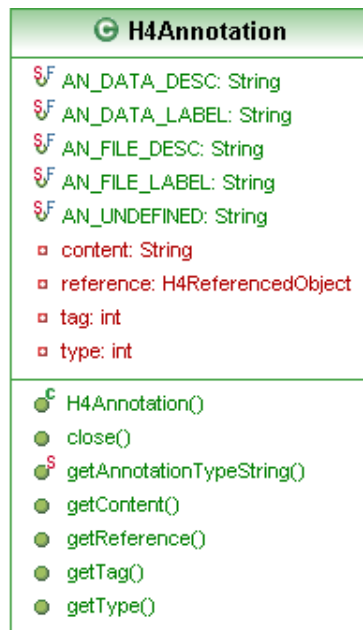


Figure 12: H4Annotation class

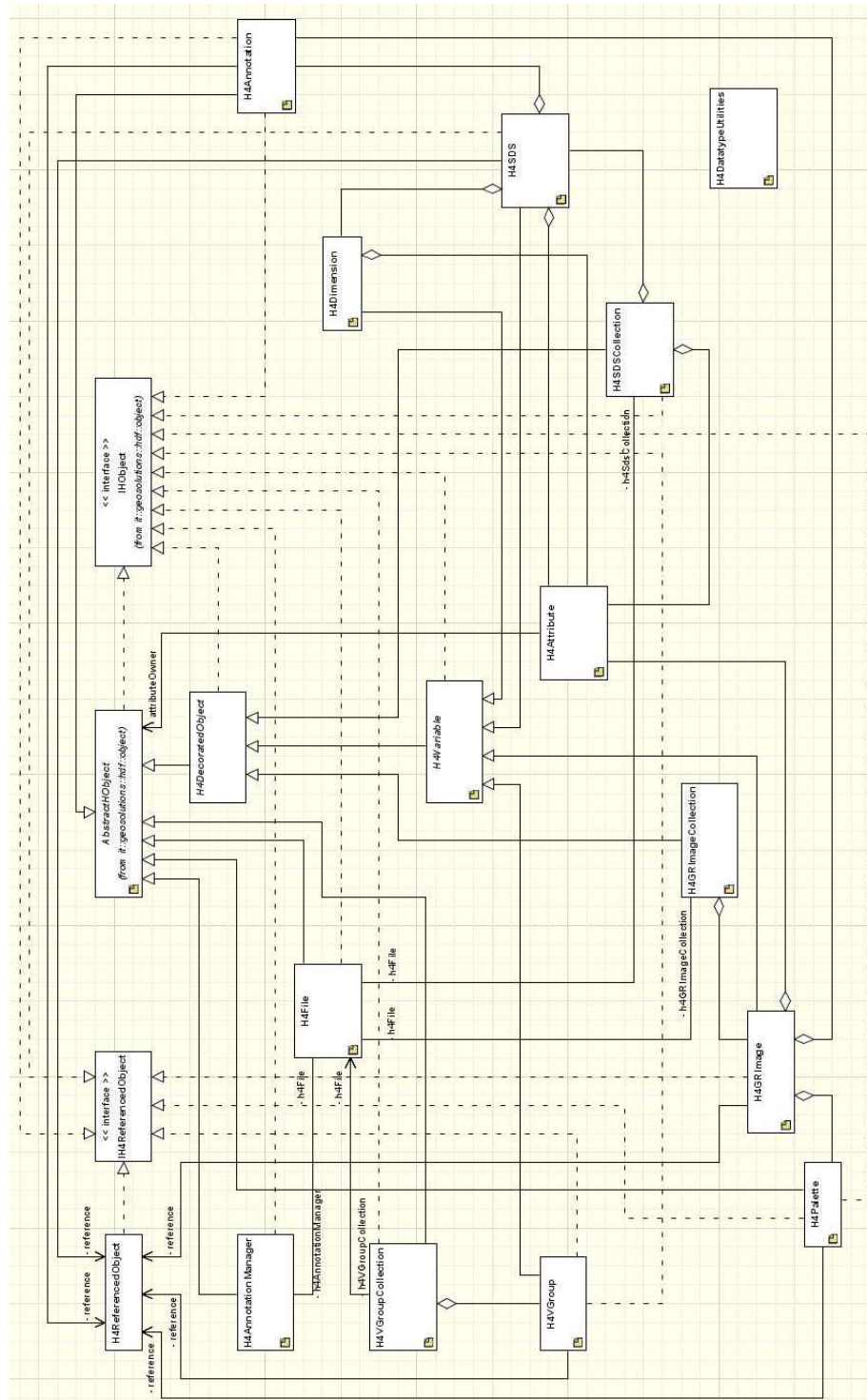


Figure 13: Class Diagram

7 Other classes

7.1 Parent class: AbstractHObject

The parent class of almost all the classes is the abstract **AbstractHObject** class which essentially stores the HDF Object identifier. This class implements the **IHObject** interface which has 2 methods:

- `getIdentifier()`
- `close()`

The first one retrieves the identifier which is the integer provided by the proper underlying HDF routine when obtaining access to a specific HDF Object. The second one needs to be implemented (when needed) to terminate access to the object. It is worth to point out that with the actual implementation, closing a HDF Object (a proper subclass of **AbstractHObject**) will attempt to properly end access to its owned objects. As an instance, closing a **H4File**, will attempt to close the **H4SDSCollection** (if this has been opened) which will attempt to close all **H4SDS** which will attempt to close all opened **H4Annotations**.

In such a context, it is worth to point out that whenever you manually build a new **H4VGroup** from a parent one, you have to close it.

The **AbstractHObject** class and the **IHObject** interface are depicted in Figure 14

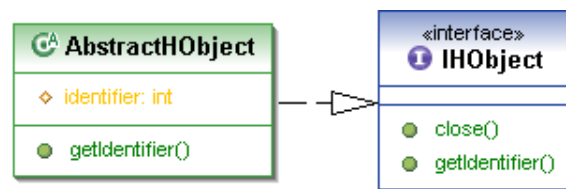


Figure 14: **AbstractHObject** class and **IHObject** interface

7.2 HDF Object reference: H4ReferencedObject

The reference number of an HDF object (such as an Annotations, a SDS, a GRImage, a Palette or a VGroup), can be retrieved by means of an inner instance of the **H4ReferencedObject** class which implements the **IH4ReferencedObject**.

7.3 HDF Object with Attributes: *H4DecoratedObject* OTHER CLASSES

The `H4ReferencedObject` class and the `IH4ReferencedObject` interface are depicted in Figure 15

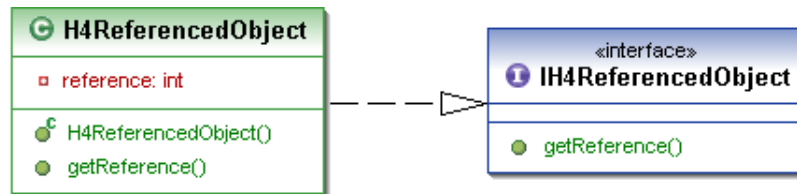


Figure 15: `H4ReferencedObject` class and `IH4ReferencedObject` interface

7.3 HDF Object with Attributes: `H4DecoratedObject`

In the previous explanations, we have frequently talked about attributes. They represent auxiliary information about the related object (such as, as an instance, a File, a specific SDS, a SDS's dimensions or an Image). Each class representing a HDF object which may have attached attributes, extends `H4DecoratedObject`.

Such a class allows to obtain access to the set of attributes available for the referred object. Basic capabilities of this class are:

- retrieving the number of attributes by means of the `getNumAttributes()` method.
- retrieving the whole set of attributes by means of the `getAttributes()` method.
- retrieving a single attribute by specifying its name by means of the `getAttribute(String attributeName)` method
- retrieving a single attribute by specifying its index in the owner object by means of the `getAttribute(int index)` method

The `H4DecoratedObject` class is depicted in Figure 16

Let us now provide more explanations about how these attributes are represented and managed.

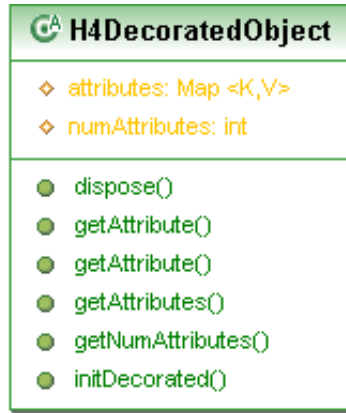


Figure 16: H4DecoratedObject class

7.3.1 Attribute: H4Attribute

Attributes may be of 2 types: *User-Defined* or *Predefined*. The first ones are defined by the calling program while the last ones have reserved names and depend on the specific object to which they are attached (SDS, Dimension,...).

Predefined attributes names are expressed as a **static String** in the proper extended *H4DecoratedObject*. As an instance, *H4SDS* has several **PREDEF_ATTRIB_XXXX** which should be used to get access to the proper predefined attribute.

Attributes are represented by instances of the *H4Attribute* class which internally holds the attribute name, the datatype and the size¹⁵ of the attribute. Usually, attribute values will be loaded only when explicitly required, except the predefined attributes of a SDS's Dimension¹⁶. Given a *H4Attribute* you can get its values by means of the *getValues()* method which returns a proper Java *Object* (an array of elements of a proper Java type).

The *H4Attribute* class is depicted in Figure 17

¹⁵Expressed in terms of the datatype. As an instance, an attribute with **FLOAT32** as datatype having 2 floating point values, has size=2. Instead, an attribute with **CHAR** as datatype having value **TEMP** has size=4.

¹⁶Due to the particular behaviour of the routine used to retrieve dimension's predefined attributes

7.3 HDF Object with Attributes: *H4DecoratedObject* OTHER CLASSES

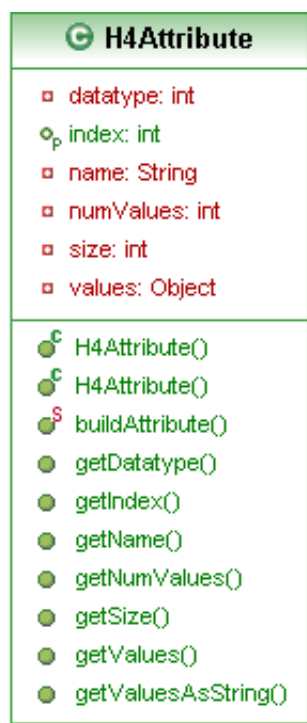


Figure 17: H4Attribute class

7.4 HDF variables: *H4Variable*

H4SDS, *H4GRImage*, *H4Dimension* and *H4VGroup* extend *H4Variable* class which has a *name* field holding the name of the represented variable. *H4Variable* class is depicted in Figure 18



Figure 18: *H4Variable* class

7.5 HDF datatypes and data allocation: *H4DatatypeUtilities*

The datatype of a specific HDF Object (as an instance, a *H4SDS* or a *H4Dimension*) is internally stored as an `int datatype` field. *H4DatatypeUtilities* may be used (by means of its `static` methods) to retrieve size and other significant properties of data having type represented by `datatype`. Furthermore, it will be used every time we need to pre-emptively allocate a proper data array prior to perform a data access¹⁷.

The *H4DatatypeUtilities* is depicted in Figure 19

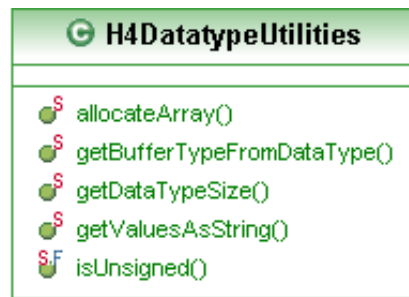


Figure 19: *H4DatatypeUtilities* class

¹⁷as an instance, when we need to read a SDS data array, or when we need to retrieve the dimension scales values

8 Limitations

As already stated in the first section, actually HDF5 is not supported. Furthermore, VData access is not allowed since the main objective of this framework is allowing to retrieve and manage raster data which are usually stored as SDS or Images. Finally, this framework does not allow writing capabilities.