HDF4 File Content Map Writer User's Guide

Prepared for NASA GES DISC

02/01/2017

HDF4 Mapping Project

Version 1.0.8

www.hdfgroup.org

Contents

1. Introduction 4

2. Installing the Writer 5
   2.1 Release Files 5
   2.2 Installation 5

3. Using the Writer 6
   3.1 Description 6
   3.2 Usage 7
   3.3 Mapped Objects 9

4. Testing the Writer 10
   4.1 Description 10
   4.2 Validating Map with Schema 10
   4.3 Reading Data with Map 10

5. Limitations 12
   5.1 Unmapped Objects 12
   5.2 Error Handling 13
   5.3 Reporting bugs 13

1. Introduction

This document provides basic instructions on the installation and use of the HDF4 File Content Map Writer. Throughout the document, we'll abbreviate the HDF4 File Content Map Writer as "writer" and the HDF4 file content map file that the writer generates as "map".

The writer supports most NASA HDF4 and HDF-EOS2 files. In this document, instructions and examples focus on NASA data, since the writer project has been funded by NASA and NASA data centers are the primary users of the writer.

The writer includes two distinct components:

- **h4mapwriter**: a tool for generating a file content map from an HDF4 file.
- **testsuite**: a set of sample HDF4 files, maps, XML schema, and a sample prototype map reader. This set is an optional component for testing the writer.

The goal of the writer is to produce a map for long term preservation of the original HDF4 data. Long term preservation means storing both *accurate* and *complete* information about the objects in the original file. Thus, the writer doesn't produce any map when it encounters any object that it cannot support. If a user provides an option to ignore certain objects that cannot be mapped by the writer, the writer will insert a warning message at the end of the incomplete map. We'll designate any HDF4 object that the current writer cannot handle as "unmapped object".

2. Installing the Writer

2.1 Release Files

The writer is available for download from The HDF Group's web site:

http://hdfgroup.org/projects/h4map/h4map_writer.html

The writer release includes only source code. The web site provides neither binaries nor RPMs. It provides the release files shown in Figure 1.

```
h4map.tar.gz

h4map-testsuite.tar.gz

h4mapwriter-user-guide.pdf
```

Figure 1: The Writer Release Files

The **h4map.tar.gz** file has the source codes for the writer. The file **h4map-testsuite.tar.gz** has test data files, maps, and schema files. The file **h4mapwriter-user-guide.pdf** is this user's guide.

2.2 Installation

The writer was implemented and tested for CentOS5 32-bit and 64-bit Linux platforms only. The writer may work on other UNIX platforms such as MacOS X but we cannot guarantee this.

You can install the writer from the source distribution by following these steps:

- Download the distribution package from the above URL.
- Unpack the package using **tar –zxvf h4map.tar.gz**.
- Change directory to **h4map** and run **BUILD_AND_INSTALL** shell script.
- Check the **h4mapwriter** binary under **install/bin** directory.

For testing, you may want to install the writer testsuite. The testsuite requires **libxml2** package for the **xmllint** XML validation tool and for the prototype HDF4 File Content Map Reader. If you don't have the **libxml2** package in your system, please install it first. Then, please follow these steps to test the writer:

- Download the testsuite distribution package from the above URL.
- Unpack the package using **tar –zxvf h4map-testsuite.tar.gz** on the *same* directory where your **h4map** directory exists.
- Change directory to **h4map** and run **BUILD_AND_TEST** shell script.

3. Using the Writer

3.1 Description

The writer is a tool that reads an HDF4 file and produces a corresponding map in XML. The map will contain key information about the HDF4 objects in the original HDF4 file. Two examples of key information items are the offset and byte size. For example, you can see in Figure 2 that **h4:Array** element has **h4:arrayData** element, and it contains a **offset** and **nBytes** attributes. The HDF4 objects

```
        <h4:Array name="TotO3_A" path="/ascending/Data Fields" nDimensions="2"
id="ID_A10">
            <h4:ArrayAttribute name="_FillValue" id="ID_AA10">
              <h4:datum dataType="float32" byteOrder="bigEndian" floatingPointFormat="IEEE"/>
              <h4:attributeData>
                <h4:byteStream offset="63123439" nBytes="4"/>
              </h4:attributeData>
              <h4:numericValues>-9999.000000</h4:numericValues>
            </h4:ArrayAttribute>
            <h4:dataDimensionSizes>180 360</h4:dataDimensionSizes>
            <h4:dimensionRef name="YDim:ascending" dimensionIndex="0" ref="ID_D1"/>
            <h4:dimensionRef name="XDim:ascending" dimensionIndex="1" ref="ID_D2"/>
            <h4:datum dataType="float32" byteOrder="bigEndian" floatingPointFormat="IEEE"/>
            <h4:arrayData fastestVaryingDimensionIndex="1" compressionType="deflate"
deflate_level="5">
              <h4:byteStreamSet>
                <h4:byteStream offset="1019457" nBytes="53248"/>
                <h4:byteStream offset="28545805" nBytes="8205"/>
              </h4:byteStreamSet>
            </h4:arrayData>
        </h4:Array>
```

Figure 2: Part of Sample HDF4 Map File in XML

and their corresponding XML elements are described in Section 3.3.

3.2 Usage

The writer is a command line tool with many options. The options are listed in the alphabetical order.

-c --continue-with-unmapped-objects

Continue to generate an incomplete map by ignoring all unmapped objects. By default, the writer checks for unmapped objects and reports errors without generating a map.

-e --put-file-attributes-at-the-end

Put all file attributes at the end of map file.

-f --filename-only

Include only the filename, not the complete directory path, in the map. By default, the writer inserts the real absolute path and filename in a file system, not symbolic link, in the map when symbolic links are used. If symbolic path or filename is desired in the map, the map file must be manually edited.

-h --help

Print the summary of usage, and exit.

-i TAG-LIST **--ignore-tags=**TAG-LIST

Ignore unmapped objects with tags in TAG-LIST and continue to generate an incomplete map file. The TAG-LIST is a sequence of tag numbers separated by comma. While the -c option ignores *all* unmapped objects, this option ignores unmapped objects with certain tags only.

-l LOGFILE **--error-log-file**=LOGFILE

Redirect all UNIX *stderr* messages to the LOGFILE. If LOGFILE exists, it will append error messages.

-m --merge-sds-file-attributes

Merge all SDS file attributes into a single attribute if they have a similar name and end with different suffixes (e.g., StructMetadata.0, StrcutMetadata.1, ..., StructMetadata.*n*).

-n --**no-checksum**

Do not compute an MD5 checksum. In addition, do not include any checksum information.

-o --**odl-parser**

Parse ODL metadata string and write its contents as Group and Attribute in map.

-s CHECKSUM --**set-checksum**=CHECKSUM

Set the checksum value to be CHECKSUM. The writer will insert the CHECKSUM instead of having the writer compute an MD5 checksum.

-t --trim-nulls-from-end

Trim NULL characters from the end of the merged SDS file attributes. For example, a string "END\\0\\0\\0" will be shortened as "END".

This option has no effect unless the -m (or --merge-sds-file-attributes) option is used.

-u --**uuid-md5**

Compute md5 and uuid on each byte stream and add them as attributes. XML validation will fail if this option is used because schema doesn't have them.

-V --version

Print version information.

-z[1-9] **--compress-output-file**[=1-9]

Compress the generated map file using gzip using the optional level. This option requires an output file name which will be compressed. Regulate the speed of compression using the specified digit #, where -z1 indicates the fastest compression method (less compression) and -z9 indicates the slowest compression method (best compression). The default compression level is 6 (that is, biased towards high compression at expense of speed).

3.3 Mapped Objects

Table 1 explains how HDF4 objects are mapped into XML elements.

HDF4 Object	XML Element
Scientific Data Set (SDS)	Array
Vgroup	Group
Vdata	Table
Palette	Palette
8-bit Raster Image	Raster
SDS Dimension	Dimension

Table 1: Mapping between HDF4 Objects and XML Elements

HDF4 Annotation objects can be attached to any HDF4 objects in Table 1 and they need special attention. Table 2 explains how HDF4 annotation objects are mapped into XML elements. Both Label and Description annotation objects are mapped. Annotations on Vdata, Dimension, and Raster Images are not supported yet.

HDF4 Annotation Object	XML Element
File, File SDS, File GR	FileAttribute
Vgroup	GroupAttribute
SDS	ArrayAttribute
Palette	PaletteAttribute

Table 2: Mapping between HDF4 Annotations and XML Elements

Please see section **5.1 Unmapped Objects** for objects that are not mapped by the writer.

4. Testing the Writer

The writer comes with a collection of test data and sample map files.

4.1 Description

The tests address these two questions

1. How do we know that all objects are properly mapped in the map file from the original HDF4 file?
2. How do we ensure that all data can be retrieved by following the instruction written on the map?

These questions are quite challenging to answer, but testing can boost our confidence.

For map testing, you will need the following tools.

- xmllint
- hread

4.2 Validating Map with Schema

The first phase of testing uses **xmllint** to determine whether or not the generated XML map is syntactically valid, and is consistent with the map schema. Once you have generated a map, you can run the tool as follows:

%xmllint –schema HDF4map.xsd –noout *test.xml*

If the command output says "*test.*xml validates", then the file is valid. Otherwise, the tool will report the first line in which it doesn't conform to the schema or XML syntax.

4.3 Reading Data with Map

The second phase of testing determines whether the data can be retrieved successfully with a sample reader. If you have generated a valid map file with the writer, you can run the reader tool as follows:

%hread *test.xml test.hdf*

If the reader can parse the input map file (test.xml), read data from the input HDF file (test.hdf) using the offset and length information specified in all **Array** elements in the map, and verify the values of data from the input HDF file against the verification values in the map, it will exit without any message. Otherwise, the reader will report an error message similar to the following:

verification failed with 9 errors on SDStemplate array.

Please note that the sample reader is a prototype only and far from the perfect. It was designed to give you a rough idea on how to retrieve data. Nevertheless, it can be useful in examining whether the writer is working properly or not.

Although we provide the sample reader, every map file has instructions at the beginning of map in an XML comment block as shown in Figure 3. Thus, the map user should be able to figure out how to read data from the original HDF4 file using the map file instruction only without looking at the sample reader code.

```
<!--
This XML file provides access to data stored in a companion HDF4 file without requiring HDF4 software.


Abbreviations:
obj = HDF4 object
elem = XML element
attr = XML attribute
Elem = Attribute, Group, Table, Array, Dimension, Raster or Palette elem
Read bytes = Using information from byteStream elem, read the indicated number of bytes, starting at the indicated zero-based
 offset in the HDF4 file. If there are multiple byteStream elems for a given Elem, they will be subelems of a byteStreamSet
 elem; read and concatenate the bytes from the multiple byteStreams in the order they appear.
Process = Using information from the datum elem, process the bytes read on a datum-by-datum basis, applying byteOrder
 transformations if needed
Access = Interpret the processed bytes based on the dataType to obtain the raw data values. Calibrate raw data values if Elem
 has calibration Attribute elems.
 …
```

Figure 3: The First Part of Instruction in Map

5. Limitations

The writer has some limitations and the user should be aware of them to minimize surprises. Knowing the writer's limitations can also help the map user to understand what's really going on with the map.

5.1 Unmapped Objects

Table 3 summarizes the HDF4 objects that are not supported by the writer. By default, the writer will report unmapped objects and exit without any map being produced when it encounters them. Unlike error handling behaviors explained in the next section, the writer doesn't exit immediately when it encounters one of the unmapped objects; it collects unmapped objects and reports them together after examining all HDF4 objects in an HDF4 file.

Unmapped Object
Tag: 1, 11-16, 50-51, 60, 102-103, 107, 204, 304, 307-312, 400-401, 500-501, 602-603, 709, 781
Little-endian number type
External Files
SDS compression other than gzip
Raster Images other than 8-bit with RLE
Vdata with column storage order

Table 3: Unmapped Objects

The writer can still generate a map when an ignore option (-c or –i) is given. This means that the writer will skip printing some or all information about the unmapped objects in UNIX *stderr,* but it will provide information about them in the final map. Thus, the writer user will not be bothered by the error messages but the map user in the future will know that something is missing in the map file.

Figure 4 illustrates how a map will look when there are unmapped objects. A short warning message about unmapped objects will appear at the end of map file before the closing file tag **</h4:HDF4FileContents>** in the comment block.

If you find any missing object in the maps generated from NASA files, please report them, following the procedure described in section 5.3.

Although the writer can map most 8-bit raster images, the content of the map writer may not accurately describe the true relationship between groups and rasters that was intended when the file was created. This limitation is related to the way that the HDF4 library associates raster images with groups internally.

5.2 Error Handling

Any system call error will produce error messages and force the writer to quit immediately. For example, the writer reports the "Out of Memory" message whenever memory allocation fails and exits immediately without further processing.

When there's an error with HDF4 function calls, it also exists immediately with an error message that includes the HDF4 function name and where the error occurred. This kind of error is very rare but critical, and that's why the writer exits immediately. For example, if external files are used and can't be found in the location specified in the HDF4 file, **SDreadata()** will return FAIL and the writer will exit immediately. Please report these HDF4 API errors, together with the HDF4 file that caused them, using the procedure described in section 5.3.

When there are unmapped objects, they will appear as errors in UNIX *stderr* (or log file if –l option is specified). The errors in UNIX *stderr* can help map producers to decide whether they should continue to generate an incomplete map or not with an ignore option (-c or –i) as described in section 5.1.

5.3 Reporting bugs

There are many HDF4 files that NASA has produced and still produces today. In addition, HDF4 API has evolved over a long period time and it is complex. Although the writer developers tried their best to cover a diverse group of HDF4 files, it could not cover them all. Thus, it is important for us to know what errors occurred when you used this writer on your HDF4 data product. Please report any errors and unmapped objects back to us by contacting The HDF Group Help Desk (help@hdfgroup.org).

```
<!--
Warning: Some objects or features in the HDF4 file were not mapped since they fell outside the capabilities of the map writer. If recovery of these objects or features is important, the use of HDF4 libraries will be required.
-->
  </h4:HDF4FileContents>
</h4:HDF4map>
```

Figure 4: Map with Unmapped Objects