HDF4 File Content Map Schema Design Notes

Ruth Aydt (aydt@hdfgroup.org)

**Ruth Duerr (rduerr@nsidc.org), Mike Folk (mfolk@hdfgroup.org),
Hyo-Kyung Lee (hyoklee@hdfgroup.org), Christopher Lynnes (christopher.s.lynnes@nasa.gov),
Elena Pourmal (epourmal@hdfgroup.org), Binh-Minh Ribler (bmribler@hdfgroup.org)**

The HDF4 File Content Map Schema was developed as part of a NASA-funded project to improve long-term preservation of EOS data by independently mapping HDF4 data objects. This document contains a subset of the design notes that were discussed by the project team during the development of the schema. The notes are being distributed more widely as part of the project output as they may provide helpful context for others outside of the project team.

Table of Contents

1 Background 4

2 Audiences 4

2.1 Primary Audience 4

2.2 Other Audiences 5

2.2.1 Data Preservation System 5

2.2.2 Demonstration Reader Application 5

2.2.3 Users of XML Technologies 5

3 Requirements and Goals 6

4 Guidelines and Responsibilities 6

5 Definitions from PREMIS 6

5.1 Types of Metadata 7

5.1.1 Preservation Metadata 7

5.1.2 Structural Metadata 7

5.1.3 Technical Metadata 7

5.1.4 Descriptive Metadata 7

5.2 PREMIS Data Model and Data Dictionary 7

5.2.1 Entity 7

5.2.2 Intellectual Entity 7

5.2.3 Digital Object 7

5.2.4 Representation 8

5.2.5 File 8

5.2.6 Bitstream 8

5.2.7 Compound Object 8

5.2.8 Relationship 8

5.2.9 Structural Relationship 8

5.2.10 Root 8

5.3 Other Terms used by PREMIS 8

5.3.1 Format 8

5.3.2 Transformation 9

5.3.3 Migration 9

6 Description of HDF4 Mapping using Definitions from PREMIS 9

7 HDF4 Data Model and File Format 10

7.1 HDF4 Data Model 10

7.1.1 HDF4 Object 10

7.1.2 Low-Level Object 11

7.2 HDF4 File Format 11

7.2.1 Low-Level Object (LL-Object), Low-Level Descriptor (LL-Descriptor), Low-Level Element (LL-Element) 12

7.3 HDF4 Library 12

8 Description of HDF4 Mapping using Terms from HDF4 13

9 HDF4 Objects and HDF4 File Content Map Elements 14

9.1 HDF4 File Object 14

9.2 HDF4 Vgroup Object 14

9.3 HDF4 Vdata Object 14
9.4 HDF4 Scientific Data Set (SDS) Object 15
9.5 HDF4 General Raster Object 15
9.6 HDF4 Palette Object 16
9.7 HDF4 Attribute Object 16
9.8 HDF4 Annotation Object 16

10 Data Type, Storage Representation, Datum 17
10.1 Data Type 17
10.2 Storage Representation 17
10.3 Datum Description 18
10.3.1 Character Datum 18
10.3.2 Integer Datum 19
10.3.3 Floating Point Datum 19

11 Preservation Considerations and Unmapped HDF4 Features 20
11.1 Unsupported Compression Schemes for Scientific Data Sets 20
11.2 Unsupported Compression Schemes for General Raster Images 21
11.3 Unsupported External Files for Raw Data 21
11.4 Unsupported Raster and Palette Features 21
11.5 Unsupported Annotations 22
11.6 Unsupported Ordering of Raw Data 22
11.7 Unsupported LL-Objects 22

12 Recommended Comments for HDF4 File Content Maps 23
12.1 Comments Providing Explanation 24
12.1.1 Comment Block One: HDF4FileInformation 24
12.1.2 Comment Block Two: HDF4FileContents 24
12.1 Comments Providing Values for Verification 27
12.1.1 Table Element and Verification Comment 27
12.1.2 Array Element and Verification Comment 28
12.1.3 Dimension Element and Verification Comment 28
12.1.4 Raster Element and Verification Comment 29
12.1.5 Palette Element and Verification Comment 29

13 Additional Information 29

Acknowledgements 29

Revision History 29

1 Background

The *HDF4 File Content Map Schema* describes an XML Document that provides access to *content* originally stored in a companion binary HDF4 file without reliance on the HDF4 library. For our purposes, we define *content* as the data and metadata provided by the creator of the binary HDF4 file.

The HDF4 File Content Map Schema is composed of several XML schema documents written in the XML Schema Definition Language, XSDL.

An XML Document that conforms to the HDF4 File Content Map Schema is an *HDF4 File Content Map*.

The HDF4 File Content Maps and the companion binary HDF4 files will be part of NASA's data preservation system.

The word *Schema* will be used interchangeably with *HDF4 File Content Map Schema* throughout the remainder of this document.

2 Audiences

The audiences for the HDF4 File Content Maps influence the Schema design.

2.1 Primary Audience

The primary target audience for the HDF4 File Content Maps is a person twenty or more years in the future who is interested in the content originally stored in the companion binary HDF4 files.

The person will have access to the binary HDF4 files.

The person will not necessarily know anything about the HDF4 data model or file format, nor can they be expected to have access to any HDF4 documentation or software.

The person will have very basic knowledge of XML, or access to materials to obtain that knowledge.

The person may not have access to the HDF4 File Content Map Schema.

The person will have computers and digital storage media at their disposal, and have the knowledge and ability to access the binary HDF4 files on a byte-by-byte basis.

The person will be familiar with, or have access to documentation that describes, data representations and compression schemes that are in use today. These include:

Data Representation:

- signed and unsigned characters (8 bits)
- signed and unsigned integers (8, 16, and 32 bit)
- two's complement signed integer representation
- IEEE floating point representation (32 and 64 bit)
- big and little-endian byte order

Compression:

- DEFLATE (gzip)

- Run-Length Encoding (RLE)
- Adaptive Huffman (sometimes referred to as Skipping Huffman)[1]
- JPEG[1]
- Szip[1,2]

The person may write a program (a Reader Application) to read an HDF4 File Content Map and companion binary HDF4 file to access the content.

2.2 Other Audiences

2.2.1 Data Preservation System

Another audience for the HDF4 File Content Maps is a present-day data repository used to support the digital preservation process. The repository may extract preservation metadata about a binary HDF4 file from an HDF4 File Content Map.

2.2.2 Demonstration Reader Application

A Demonstration Reader Application will be developed as part of the Mapping Project to read an HDF4 File Content Map and the companion binary HDF4 file, and makes the contents available to the user.

The Demonstration Reader Application will not be built with the HDF4 library.

The Demonstration Reader Application may be built with an XML parser from an Open-Source project such Gnome (libxml2) or Apache Xerces (Xerces C++, Xerces Java, Xerces Perl).

The developer of the Demonstration Reader Application will have knowledge of XML.

The developer of the Demonstration Reader Application will not have access to the Schema.

The developer of the Demonstration Reader Application may not have knowledge of the HDF4 data model or file format.

2.2.3 Users of XML Technologies

Others might employ XPath, XQuery, an XSLT processor, an XML validating parser, and/or an XML DOM parser to work with the HDF4 File Content Maps.

This audience will have knowledge of XML and access to the Schema.

This audience will not access the binary HDF4 files and may not have knowledge of the HDF4 data model or file format.

[1] Adaptive Huffman, JPEG, and Szip compression are commented out of the Schema as they are not implemented in Vesion 1.0.0 of the HDF4 File Content Map Writer, *h4mapwriter*.

[2] Szip compression is covered by copyright and licensing terms that are more restrictive than those of the other compression schemes listed. See http://www.hdfgroup.org/doc_resource/SZIP/ for more information.

3 Requirements and Goals

The Schema must be designed so that the HDF4 File Content Maps it describes provide complete access to the content in the binary HDF4 files held by NASA's Earth Observing System.[3]

To the extent possible given the time constraints of the project, and without introducing excessive complexity, the Schema should be designed so that the HDF4 File Content Maps it describes provide access to all content originally stored in binary HDF4 files. This would include content stored using HDF4 objects, data representations, or compression schemes that are not found in NASA's EOS collection of binary HDF4 files.

The Schema will be designed to accommodate the needs of all audiences to the greatest extent possible. When a design choice must be made, the needs of the primary audience (§ 2.1) will take priority over the needs of the other audiences (§ 2.2).

4 Guidelines and Responsibilities

Existing standard schemas and terminology from the Digital Preservation and other communities will be leveraged in the design and documentation of the HDF4 File Content Map Schema.

Terminology and design hints from the book "Definitive XML Schema", by Priscilla Walmsley, will be employed in the design and documentation of the HDF4 File Content Map Schema.

XML and XSDL offer almost unlimited options for schema design. The HDF4 Mapping Project team, especially the NASA members, will be responsible for deciding which of several draft Schemas best meet the Requirements and Goals of the project.

The HDF4 data model and binary file format offer considerable flexibility and extensibility, and have evolved over the lifetime of HDF. The HDF4 Mapping Project team, especially The HDF Group members, will be responsible for ensuring that the Schema design describes HDF4 File Content Maps that provide access to content originally stored in binary HDF4 files, in accordance with the Requirements and Goals of the project.

5 Definitions from PREMIS

The *PREMIS Data Dictionary for Preservation Metadata - version 2.0,* available from http://www.loc.gov/standards/premis/v2/premis-2-0.pdf, defines several terms that are relevant to the HDF4 Mapping project. *Unless otherwise noted, the text here is copied verbatim or almost verbatim from the PREMIS Data Dictionary—in most cases from the Glossary. The text was collected in this document so that it could be more easily accessible to the project team members during design discussions.*

[3] The HDF4 File Content Map Writer being developed as part of the Mapping Project will detect content that cannot be mapped and report an error. If this happens, a decision will be made to expand the Schema and Writer to cover the content, or–if it is deemed unimportant–to leave it unmapped. Because of the flexibility of the HDF4 format and library APIs, it is possible that unanticipated content may be present in HDF4 binary files.

5.1 Types of Metadata

5.1.1 Preservation Metadata

Information a Preservation Repository uses to support the digital preservation process. Preservation Metadata spans Administrative, Technical, and Structural Metadata.

Note: Administrative Metadata is not relevant to the HDF4 Mapping project.

5.1.2 Structural Metadata

Describes the internal structure of digital resources and the relationships between their parts. It is used to enable navigation and presentation. (From NINCH Guide to Good Practice: www.nyu.edu/its/humanities/ninchguide/appendices/metadata.html.)

5.1.3 Technical Metadata

Information describing physical (as opposed to intellectual) attributes or properties of Digital Objects. Some Technical Metadata properties are Format specific (that is, they pertain to a Digital Object in a particular Format, for example, color space associated with a TIFF image), while others are Format independent (that is, they pertain to all Digital Objects regardless of Format, for example, size in bytes).

5.1.4 Descriptive Metadata

Metadata that serves the purposes of discovery (how one finds a resource), identification (how a resource can be distinguished from other, similar resources), and selection (how to determine that a resource fills a particular need, for example, for the DVD version of a video recording. (From Caplan, *Metadata Fundamentals for All Librarians,* ALA Editions, 2003*)*

Typically, Descriptive Metadata describes Intellectual Entities and is often domain-specific. (pg 23-24)

5.2 PREMIS Data Model and Data Dictionary

5.2.1 Entity

Abstraction for set of "things" (agents, events, etc.) described by the same properties. The PREMIS data model defines five types of Entities: Intellectual Entities, Digital Objects, Agents, Rights, and Events.

Note: Agents, Rights, and Events are not relevant to the HDF4 Mapping project.

5.2.2 Intellectual Entity

Coherent set of content that is described as a single intellectual unit for purposes of management and description. For example, a book, a map, a photograph, a serial. An Intellectual Entity can include other Intellectual Entities; for example, a Web Site can include a Web Page, and a Web Page can include a photograph. An Intellectual Entity can have one or more Representations.

5.2.3 Digital Object

Discrete unit of information in digital form. Note that the PREMIS definition of Digital Object differs from the definition commonly used in the digital library community, which holds a digital object to be

a combination of identifier, metadata, and data. The Digital Object Entity in the PREMIS Data Model is an abstraction defined only to cluster attributes (Semantic Units) and clarify relationships. A Digital Object cannot be modified. A Digital Object can be a Representation, File, Bitstream, or Filestream.

Note: Filestream is not relevant to the HDF4 Mapping project.

Note: PREMIS also refers to these as Objects. We will use the complete name, Digital Objects, in this document to distinguish them from HDF4 Objects.

5.2.4 Representation

Digital Object instantiating or embodying an Intellectual Entity. A Representation is the set of stored Files and Structural Metadata needed to provide a complete and reasonable rendition of the Intellectual Entity.

5.2.5 File

Named and ordered sequence of Bytes that is known by an operating system. A File can be zero or more Bytes, has access permissions, and has file system statistics such as size and last modification date. A File also has a Format.

5.2.6 Bitstream

Contiguous or non-contiguous data within a File that has meaningful common properties for preservation purposes. A Bitstream cannot be transformed into a standalone File without the addition of file structure (headers, etc.) and/or reformatting the Bitstream in order to comply with some particular Format.

5.2.7 Compound Object

Digital Object composed of multiple Files, for example, a Web Page composed of text and image files.

5.2.8 Relationship

Statement about an association between instances of Entities.

5.2.9 Structural Relationship

Relationship between parts of a Digital Object.

5.2.10 Root

The File that must be processed first in order to render a Representation correctly.

5.3 Other Terms used by PREMIS

5.3.1 Format

Specific, pre-established structure for the organization of a File, Bitstream, or Filestream.

5.3.2 Transformation

Process performed on a Digital Object that results in one or more new Digital Objects that are not bit-wise identical to the source Digital Object. Examples of Transformation include Migration and Normalization

5.3.3 Migration

Preservation strategy in which a Transformation creates a version of a Digital Object in a different Format, where the new Format is compatible with contemporary software and hardware environments. Ideally Migration is accomplished with as little loss of content, formatting and functionality as possible, but the amount of information loss will vary depending on the Formats and content types involved. Also called “format migration” and “forward migration”.

6 Description of HDF4 Mapping using Definitions from PREMIS

This section contains a high-level description of the Mapping process using definitions from the PREMIS Data Dictionary.

The Intellectual Entity to be preserved is the content stored in a binary HDF4 file.

A binary HDF4 file is a File Digital Object with the HDF4 file format.

A binary HDF4 file is a Representation of the Intellectual Entity.

A Mapping Transformation applied to a binary HDF4 file results in an HDF4 File Content Map.

An HDF4 File Content Map is a File Digital Object in the XML format that conforms to the HDF4 File Content Map Schema.

Some, but not all, of the content stored in a binary HDF4 file is migrated to an HDF4 File Content Map by the Mapping Transformation. For the content that is not migrated, the Mapping Transformation writes Preservation Metadata to the HDF4 File Content Map that allows a reader to access the content in the binary HDF4 file. The content that remains in the binary HDF4 file is in Bitstream Digital Objects.

An HDF4 File Content Map and a companion binary HDF4 file is a Representation of the Intellectual Entity. The HDF4 File Content Map is the Root of this Representation.

Figure 1 shows a diagram of the PREMIS view of the HDF4 Mapping Process.

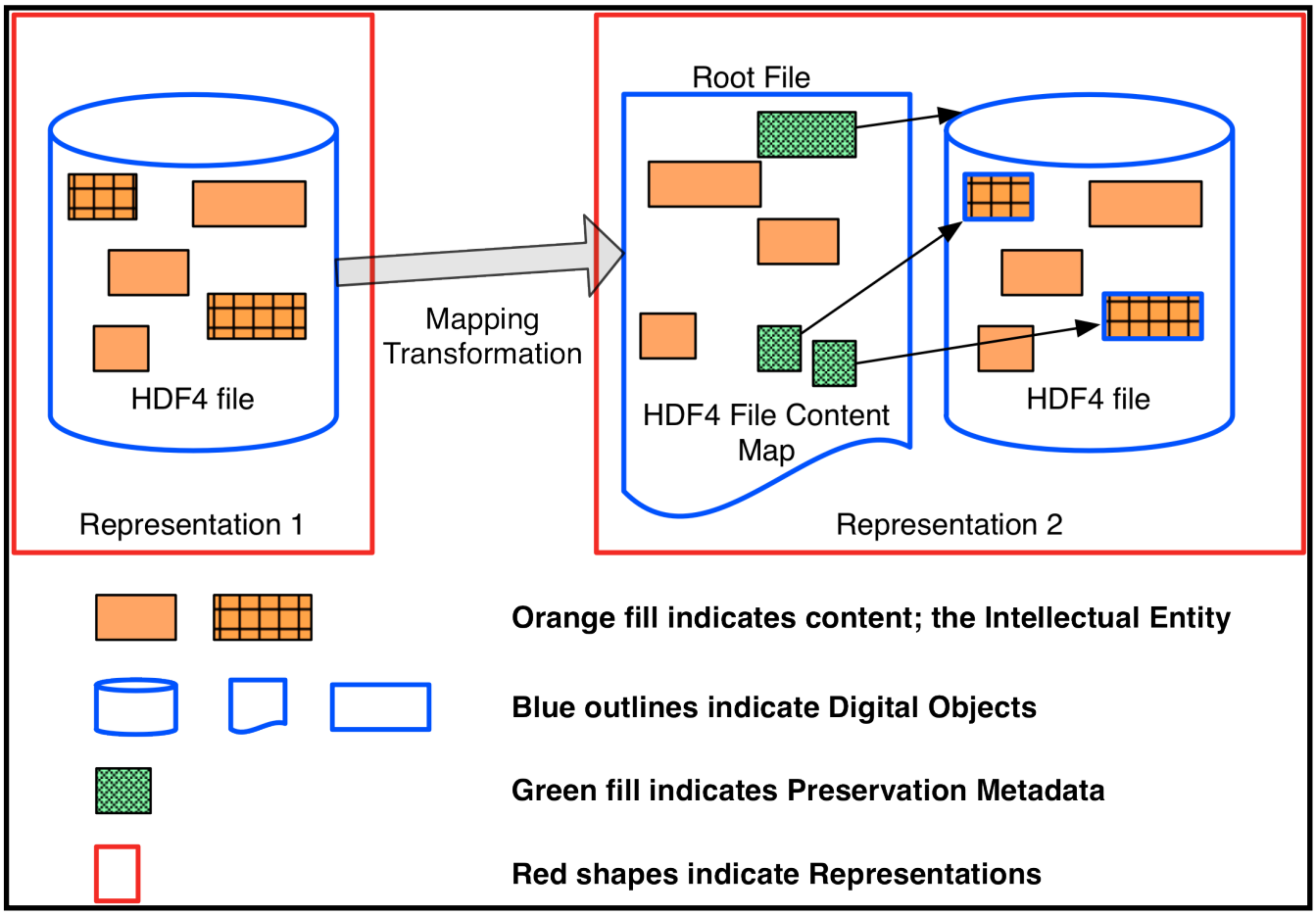

Figure 1: PREMIS view of HDF4 Mapping Process

If an HDF4 binary file refers to data stored in one or more external files, those files are also Digital Objects that would be included in both Representations.

7 HDF4 Data Model and File Format

At the beginning of this document, we defined *content* to be the data and metadata provided by the creator of a binary HDF4 file. In this section, we give a high-level overview of the HDF4 Data Model and File Format, and define terms we will use when discussing them and their roles in the HDF4 Mapping process. We strive to harmonize the terms used in this section with those defined in the PREMIS Data Dictionary.

7.1 HDF4 Data Model

The HDF4 Data Model is composed of two types of Entities: *HDF4 Objects* and *Low-Level Objects*.

7.1.1 HDF4 Object

An HDF4 *Object* is an abstraction in the HDF4 Data Model defined to store and organize user data and metadata (descriptive, technical, and structural).

HDF4 Objects are the primary *conceptual* units for representing user data and metadata in HDF4.

HDF4 Objects of the same type have the same properties (metadata), some of which may be optional.

An HDF4 Object may include, or refer to, other HDF4 Objects.

The single-file and multi-file APIs in the HDF4 library provide the interfaces to create, populate, and access HDF4 Objects.

The primary HDF4 Objects are: *Vgroup, Vdata, Scientific Data Set, General Raster Image, Palette, Attribute, and Annotation.*

An HDF4 Object can have associated metadata such as *name, rank,* and *data type* that may be Descriptive or Technical.

Metadata that is fairly small, such as *name, rank, and data type,* is considered part of the Object.

Metadata that is more extensive, such as a *Dimension*, is treated as a separate HDF4 Object related to one or more primary HDF4 Objects.

Structural metadata provided by the creator of the file is represented in the technical metadata associated with an HDF4 Object, or in the relationships between various HDF4 Objects,

The term *raw data* is used to refer to the creator-provided data values that are stored in HDF4 Scientific Data Set, General Raster Image, and Vdata Objects. Some metadata that is more extensive, such as that in Dimension, Attribute, Annotation, and Palette Objects, is also considered raw data.

The HDF4 Objects are: *File, Vgroup, Vdata, Scientific Data Set, Dimension, General Raster Image, Palette, Attribute, and Annotation.*

7.1.2 Low-Level Object

An HDF4 Low-Level Object is the basic building block used to organize and store data in an HDF4 file.

The HDF4 File Format defines these low-level blocks and their structures.

The Low-Level Objects are part of the HDF4 Data Model, as they are accessible via the Low-Level API and can be created and accessed directly by an application. This use is not typical.

More details on Low-Level Objects are included in the next section, which discusses the HDF4 File Format.

7.2 HDF4 File Format

The HDF4 File Format defines the low-level building blocks used to organize and store data in a binary HDF4 file. The internal layout of the low-level building blocks is also defined by the HDF4 File Format.

The standard HDF4 terms *Data Object, Data Descriptor,* and *Data Element* are not used for these low-level building blocks in this document. Instead, we adopt the terms *Low-Level Object, Low-Level Descriptor,* and *Low-Level Element.* The adopted terms are abbreviated *LL-Object, LL-Descriptor, and LL-Element.*

7.2.1 Low-Level Object (LL-Object), Low-Level Descriptor (LL-Descriptor), Low-Level Element (LL-Element)

The *Low-Level Object* is the basic structure for encapsulating data in an HDF4 file. The encapsulated data may be user data, user metadata, or HDF4 file format metadata. The HDF4 specification defines numerous LL-Object types.

The HDF4 Objects discussed in section 7.1.1 are instantiated in binary HDF4 files as LL-Objects.

Some HDF4 Objects are represented by a single LL-Object; most are represented by multiple LL-Objects.

Some HDF4 Objects (e.g., a Vdata and an Attribute) are represented by the same type of LL-Objects.

Some LL-Objects have the same name as HDF4 Objects (e.g., Vgroup). To distinguish between the two, we will preface the LL-Object names with LL- (e.g., LL-Vgroup)

An LL-Object is usually made up of a *Low-Level Descriptor* (12 bytes) and a *Low-Level Element* (variable length). For some LL-Objects that encapsulate HDF4 file format metadata, the LL-Descriptor component contains all of the data and there is no associated LL-Element component. (HDF4 User's Guide section 2.2.2)

A *tag* in the LL-Descriptor identifies the LL-Object type.

A *reference number* in the LL-Descriptor is assigned by the HDF4 library when a LL-Object is created.

The tag and reference number pair uniquely identifies the corresponding LL-Object in the HDF4 file. This pair is often called the *tag/ref.* (HDF4 User's Guide sections 2.2.2 and 2.2.2.1)

When the LL-Object has an LL-Element component, the offset and length of the LL-Element are stored in the LL-Descriptor. The tag, which identifies the LL-Object type, determines whether the LL-Element should exist. When an LL-Element is expected and none exists (the offset and length are 0), the LL-Object is *Incomplete.*

The LL-Element component of the LL-Object contains the bulk of the data, be it user data, user metadata, or HDF4 file format metadata. The LL-Object type determines how the data in the LL-Element is organized and interpreted.

Extended Tags are special versions of the defined tags that indicate the LL-Element component of the LL-Object is organized using an alternate physical storage layout that cannot be fully described by a simple offset and length. The HDF4 library uses data stored in the LL-Element referenced by the LL-Descriptor to locate and process the remainder of the data associated with the LL-Object. That data may be stored in linked non-contiguous blocks, in an external file, as chunked non-contiguous blocks, or in a compressed format. Extended tags are only available for the LL-Objects types that store the largest quantities of raw data (LL-COMPRESSED, LL-CHUNK, LL-Raster Image, LL-Scientific Data, LL-Vdata), and not all extended tags are available for all of the LL-Object types.

7.3 HDF4 Library

When an application uses the HDF4 Library the connection between the LL-Objects and the HDF Objects is typically hidden. For example, an Attribute is associated with a General Raster Image without the user knowing what underlying LL-Objects store the Image, the Attributes or the raw data values for either.

8 Description of HDF4 Mapping using Terms from HDF4

This section contains a high-level description of the Mapping process using definitions from the HDF4 Data Model and File Format.

The content to be preserved is stored in LL-Objects in a binary HDF4 file.

The creator of the HDF4 file thinks of the content in terms of HDF4 Objects.

A Mapping Transformation applied to a binary HDF4 file results in an HDF4 File Content Map with Elements that reflect the HDF4 Objects in the original binary HDF4 file.

The creator-supplied metadata in a binary HDF4 file is migrated to an HDF4 File Content Map by the Mapping Transformation. The Mapping Transformation writes Preservation Metadata to the HDF4 File Content Map that provides the information necessary to access the raw data in the LL-Elements of the binary HDF4 file. Preservation Metadata for the binary HDF4 file as a whole is also written to the HDF4 File Content Map.

Figure 2 shows a diagram of the HDF4 view of the HDF4 Mapping Process.

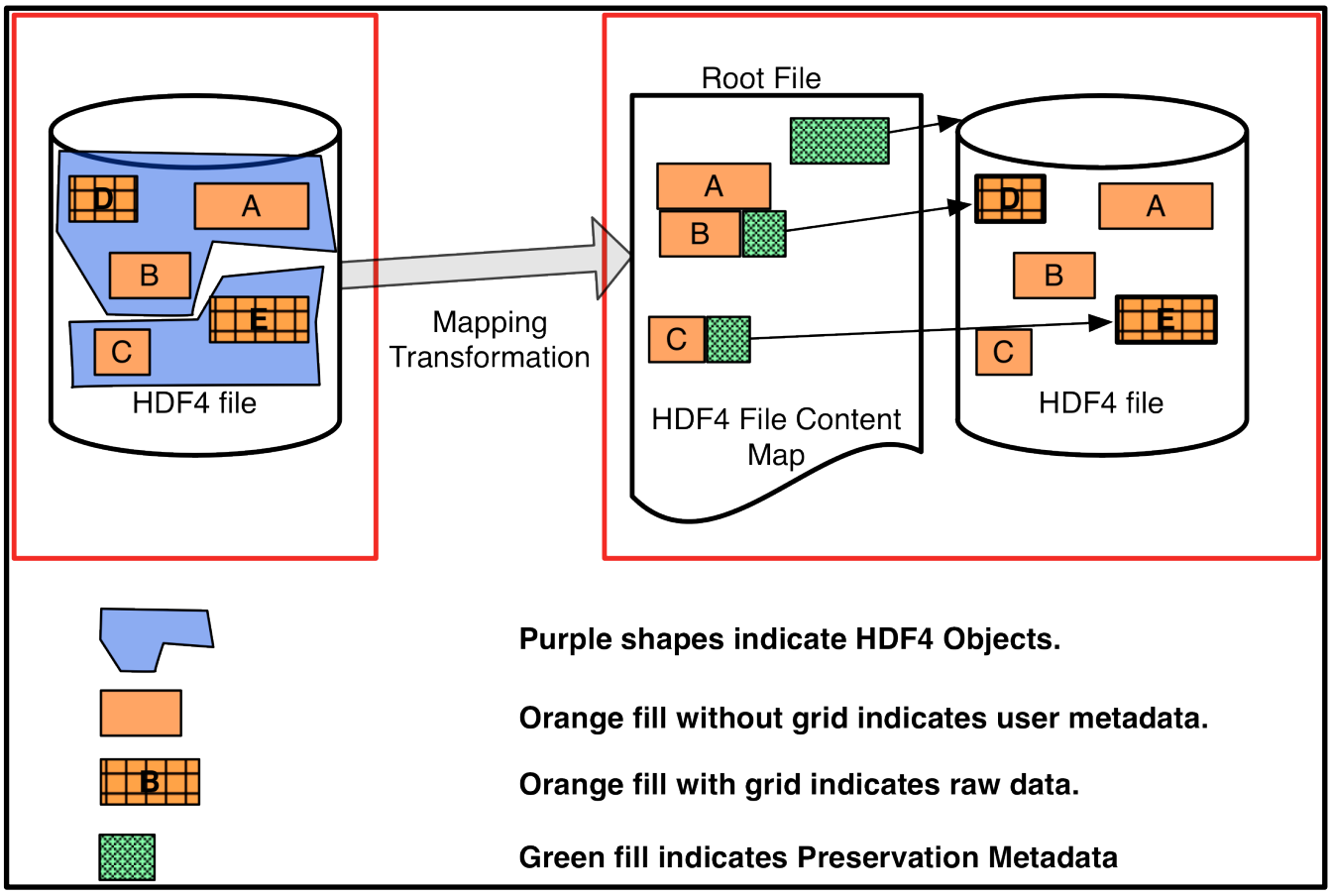

Figure 2: HDF4 view of HDF4 Mapping Process

If an HDF4 binary file refers to data stored in one or more external files, those files would also be included in the Mapping Process.

9 HDF4 Objects and HDF4 File Content Map Elements

This section gives a very high-level overview of the HDF4 Objects introduced in Section 7.1 and the elements that represent them in the HDF4 File Content Map Schema. Comments embedded in the Schema documents, available at http://www.hdfgroup.org/HDF4/XML/schema/HDF4map/1.0.0/, provide more details.

9.1 HDF4 File Object

The File Object is the entry point into the contents of an HDF4 File.

- Information about the HDF4 File, such as its size in bytes, is mapped to the *HDF4FileInformation* element.
- Information about the contents of the HDF4 File, such as other HDF4 Objects it contains, is mapped the *HDF4FileContents* element.

Mapping Note: The HDF4 File Library Version that is part of the information included in the file is not mapped, as it contains the version of the library used to create the file in some cases, and the version last used to update the file in others.

9.2 HDF4 Vgroup Object

A Vgroup Object is a grouping structure designed to associate related HDF4 Objects. (HDF4 User's Guide section 5.2)

- User-created Vgroups are mapped to *Group* elements.
- Group elements and their members form a graph of some Objects in the HDF4 file.
- The graph formed by the Groups may have cycles. Cycles are represented by *groupCycleRef* elements.

Mapping Note: Vgroup LL-Objects in the HDF4 file that were created by the HDF4 library for organization or "bookkeeping" are not mapped to Group elements as they do not represent information supplied by the creator of the HDF4 file.

9.3 HDF4 Vdata Object

A Vdata[4] Object is a customized table used to store data. The table is made up of rows (records) and named columns (fields). Every cell in the table has a [row,column] index. Every column has an associated data type and a number of entries per cell in the column (order). The values stored in the table are referred to as *raw data*. (HDF4 User's Guide section 4.2)

- User-created Vdatas are mapped to *Table* elements.
- HDF4 Vdata fields are mapped to *Column* elements.
- HDF4 Vdata number of records is represented as *nRows* Table attribute, number of fields as *nColumns* Table attribute, and field order as *nEntries* Column attribute.

[4] The name "Vdata" reflects the original use of the Object to store vertex and edge data of polygon sets.

- Preservation Metadata that allows access to, and intepretation of, the raw data in the HDF4 file is found in the *tableData* element.
- Interlace mode is mapped to the *storageOrder* attribute of tableData.

Mapping note: Vdata LL-Objects in the HDF4 file that were created by the HDF4 library for organization or to store Attribute or other data are not mapped to Table elements.

9.4 HDF4 Scientific Data Set (SDS) Object

A Scientific Data Set (SDS)[5] Object corresponds to a set of structures used to store a multi-dimensional array of data with a single data type. In addition to the primary structure that stores the array data, related structures store metadata (technical and descriptive) about the array. The values stored in the array are referred to as *raw data*. (HDF4 User's Guide section 4.2)

- HDF4 SDS objects are mapped to *Array* elements.
- SDS rank is mapped to *nDimensions* attribute.
- SDS dimensions are mapped to *dataDimensionSizes* element, which is absent if nDimensions is 0.
- If any SDS dimension has name, attribute, or scale it is mapped to a *Dimension* element that is referenced by *dimensionRef*.
- Preservation Metadata that allows access to, and intepretation of, the raw data in the HDF4 file is found in the *arrayData* element.
- Storage order is mapped to the *fastestVaryingDimensionIndex* attribute of arrayData. This attribute is absent if nDimensions is 0.
- Compression information is mapped to the *compressionType* attribute of arrayData.
- If raw data for the SDS is stored using chunks, and the array dimensions are not exact multiples of the chunk dimensions, the *allocatedDimensionSizes* attribute of Array gives the dimensions including ghost cells. See section 3.13 in the HDF4 User's Guide for a discussion of ghost areas.

9.5 HDF4 General Raster Object

The HDF4 General Raster Object encompasses several generations of Raster Images, including those written with the DFR8 API for 8-bit Raster Images, the DFR24 API for 24-bit Raster Images, and the most recent GR API.

- HDF4 8-bit Raster Images, 24-bit Raster Images, and General Raster (GR) Images are mapped to *Raster* elements in the HDF4 File Content Map.
- Number of rows (scan lines) in the image it mapped to the height attribute, number of columns to the width attribute, and image depth to the *nComponentsPerPixel* attribute.
- If a Raster has an associated Palette (color lookup table), it is referenced by *paletteRef*.

[5] The name "Scientific Data Set" reflects the type of data the Objects were originally designed to support. In fact, they can also be used to store non-scientific data. The array component of an SDS is conceptually equivalent to a variable in the netCDF data model.

- Preservation Metadata that allows access to, and intepretation of, the raw data in the HDF4 file is found in the *rasterData* element.
- Interlace mode is mapped to *dimensionStorageOrder* attribute of rasterData.
- Compression information is mapped to the *compressionType* attribute of rasterData.

9.6 HDF4 Palette Object

The HDF4 Palette Object, also know as a color lookup table (LUT), provides the means by which color is applied to an image.

- HDF4 palettes and color lookup tables (LUTs) are mapped to *Palette* elements.
- Preservation Metadata that allows access to, and intepretation of, the raw data in the HDF4 file is found in the *paletteData* element.
- Interlace mode is mapped to *dimensionStorageOrder* attribute of paletteData. This is an optional attributes that defaults to "entry,component" (Pixel interlace in HDF4 terminology).

9.7 HDF4 Attribute Object

The HDF4 Attribute Objects are used to associate textual or numeric metadata with HDF4 Files, VGroups, Vdata, Vdata Fields, SDSes, Dimensions, Rasters, and Palettes.

- HDF4 File Attributes, added through the SD or GR API, are mapped to *FileAttribute* elements.
    - Textual SD File Attributes with similar names may optionally be combined into a single element.
    - Textual File Attributes may optionally have trailing nulls trimmed.
- HDF4 Vgroup, Vdata, Vdata Field, SDS, Dimension, Raster, and Palette Attributes are respectively mapped to *GroupAttribute*, *TableAttribute*, *ColumnAttribute*, *ArrayAttribute*, *DimensionAttribute*, *RasterAttribute*, and *PaletteAttribute* elements.
- HDF4 has a number of pre-defined attributes with specific names that can be associated with SDS and Dimension objects. These are mapped to Attribute subelements of the associated Array or Dimension element in the HDF4 File Content Map.
- The Attribute metadata is fully represented inline the HDF4 File Content Map.
- Preservation Metadata that allows access to, and interpretation of, the Attribute metadata in its raw form in the binary HDF4 file is found in the *attributeData* element.

9.8 HDF4 Annotation Object

An HDF4 Annotation Object is used to associate textual descriptive metadata with an HDF4 File or LL-Object. The HDF4 API supports four distinct types of Annotation Objects (File Labels, File Descriptions, Object Labels, Object Descriptions), with slightly different characteristics.

- HDF4 File Label and File Description Annotations are mapped to *FileAttribute* elements.

- HDF4 Object Label and Object Description Annotations are mapped to *Attribute* elements based on the HDF4 Object the Annotation was associated with (via the LL-Object). See previous Section (9.7) for more on Attribute elements.
- HDF4 Annotation Objects do not have names; the names “File Label”, “File Description”, “Object Label” and “Object Description” are assigned to the Attribute elements that represent the Annotations in the HDF4 File Content Map.

10 Data Type, Storage Representation, Datum

HDF4 has the ability to store a variety of data types using different storage representations in a binary HDF4 file. The information about the data types and storage representations must be made available in the HDF4 File Content Map in order for the HDF4 file contents to be interpreted correctly. This section defines terminology and provides an overview of how the information is mapped.

10.1 Data Type

We use the term *Data Type* to refer to the user’s view of the data, in particular the data type and number of bits.

A Data Type includes:

a) data type (character, unsigned character, integer, unsigned integer, float)
b) number of bits in data type
c) for n-bit types, the relevant bits and their position in the larger integer, padding, and sign-extend information. (not implemented in Version 1.0.0 of the Schema)

In addition to standard data types with standard lengths, HDF4 allows the user to specify integer data types of non-standard length, referred to as *nbit types,* via the *SDsetnbitdataset* routine. The user is effectively saying that only certain bits of a standard-size integer are relevant. When the data is written to the HDF4 file, only those bits are saved, along with the positioning, padding, and sign-extend information. The NBIT compression scheme handles this downsizing. When the HDF4 file is read using the HDF4 library, the data is expanded into the standard-size integer. The information about the relevant bits and their position in the standard-size integer is part of the Data Type. HDF4 nbit types are not handled by Version 1.0.0 of the Schema. See also the NBIT table entry in Section 11.1.

10.2 Storage Representation

We adopt the term *Storage Representation* to describe how a given data value is stored in the HDF4 file. This includes byte order, floating point representation, and the number of bytes used to store the data when it is not the number expected based on the data type and number of bits in the Data Type. A non-standard number of bytes could occur, for example, when the data is written on a system where all numeric types are stored in 8-byte words. In this case, the number of bits in the Data Type is not equal to 8 times the number of bytes in the Storage Representation.

A Storage Representation includes:

a) byte order (big-endian, middle-endian, little-endian)

b) encoding for character types (BYTE, ASCII, EBCDIC)
c) floating point format (IEEE, Vax, Cray, Convex, Fujitsu)
d) number of bytes used to store a single data value of this type

Version 1.0.0 of the HDF4 File Content Map Schema supports all of the Storage Representation options, but the Map Writer only supports the most common (big-endian, BYTE and ASCII, IEEE, number of bytes used equals number of bytes expected).

10.3 Datum Description

The term *Datum Description* refers to the complete information about how a single value (a datum) is represented in HDF4. The Datum Description includes both the Data Type and the Storage Representation information.

While the HDF4 Data Model and File Format support a very wide range of possible Data Types and Storage Representations, only those the most common are fully supported by Version 1.0.0 of the Schema and Map Writer. Refer to the HDF4 source files "hdfi.h" and "hntdefs.h" for complete list of encodings.

Tables found in the following sections detail what datum are handled by Version 1.0.0 of the HDF4 File Content Map Schema and mapped byVerson 1.0.0 of the HDF4 Map Writer. See the Schema Document *HDF4map_datum_representation.xsd* for more information.

10.3.1 Character Datum

HDF4 defines both character and unsigned character types, and distinguishes between BYTE and ASCII representations when storing the type, with BYTE representation used only with raster data. For the mapping project, 8-bit character and unsigned character types with ASCII representation are mapped to *dataTypeT* "char8". 8-bit character and unsigned character types with BYTE representation are mapped to *dataTypeT* "byte8" and "ubyte8", respectively. The Storage Representation encoding is not expressed explicitly in the Content Map. The attribute *typeUsesNBytes* can be set if the number of bytes is other than 1.

Character Datum					
Data Type		**Storage Representation**		**Handled by V1.0.0**	
data type	**# of bits**	**# of bytes**	**encoding**	**Schema**	**Writer**
character	8	1	BYTE, ASCII	yes	yes
character	8	1	EBCDIC	yes	no
character	other than 8	1	BYTE, ASCII, EBCDIC	no	no
character	8	other than 1	BYTE, ASCII, EBCDIC	yes	no
unsigned character	8	1	BYTE	yes	yes
unsigned character	8	1	ASCII, EBCDIC	yes	no
unsigned character	other than 8	1	BYTE, ASCII, EBCDIC	no	no
unsigned character	8	other than 1	BYTE, ASCII, EBCDIC	yes	no

10.3.2 Integer Datum

Integer data types are mapped to *dataTypeT* values “int8”, “int16”, “int32”, “uint8”, “uint16”, and “uint32”. The attribute *typeUsesNBytes* can be set if the number of bytes is not the number of bits/8.

All integers are stored in twos-complement representation.

For byte order:

- BE = Big Endian, also known as MBO (Motorola Byte Order)
- LE = Little Endian, also known as IBO (Intel Byte Order)
- ME = Middle Endian, also known as VBO (Vax Byte Order)

The *byteOrder* attribute in the Content Map may be set to “bigEndian”, “littleEndian”, or “middleEndian” if the number of bytes is greater than one.

Little endian storage order was not seen in the NASA data and therefore not tested thoroughly. It can be mapped with the -c option to h4mapwriter, but there will be a warning in the Content Map file. For those cases in the table below where BE byte order is mapped, LE will probably be mapped correctly in spite of the warning.

Integer Datum						
Data Type			Storage Representation		Handled by V1.0.0	
data type	# of bits	n-bit type	# of bytes	byte order	Schema	Writer
integer	8, 16, 32	no	# bits / 8	BE	yes	yes
integer	8, 16, 32	no	any	LE, ME	yes	no
integer	8, 16, 32	yes	any	any	no	no
integer	other than 8, 16, or 32	yes or no	any	any	no	no
unsigned integer	8, 16, 32	no	# bits / 8	BE	yes	yes
unsigned integer	8, 16, 32	no	any	LE, ME	yes	no
unsigned integer	8, 16, 32	yes	any	any	no	no
unsigned integer	other than 8, 16, or 32	yes or no	any	any	no	no

10.3.3 Floating Point Datum

The HDF4 File Content Map Schema provides “float32” and “float64” *dataTypeT* values for floating point data types. The attributes *typeUsesNBytes* and *byteOrder* are the same as for integer datum. The type *floatingPointFormatT* is defined for the floating point formats, and had the value “IEEE” in all HDF4 files examined, although other values are defined in the Schema.

Floating Point Datum						
Data Type		Storage Representation			Handled by v1.0.0	
data type	# of bits	# of bytes	byte order	floating point format	Schema	Writer
float	32	4	BE	IEEE	yes	yes
float	32	4	LE	IEEE	yes	no

float	32	other than 4	BE, LE	IEEE	yes	no
float	32	any	ME	VAX	yes	no
float	32	any	BE	Cray, Convex, Fujitsu	yes	no
float (double)	64	8	BE	IEEE	yes	yes
float (double)	64	8	LE	IEEE	yes	no
float (double)	64	other than 8	BE, LE	IEEE	yes	no
float (double)	64	any	ME	VAX	yes	no
float (double)	64	any	BE	Cray, Convex, Fujitsu	yes	no
float	other than 32 or 64	any	BE, LE, ME	any	no	no

11 Preservation Considerations and Unmapped HDF4 Features

The purpose of the *HDF4 File Content Map Schema* is to describe an XML Document that provides access to *content* originally stored in a binary HDF4 file without reliance on the HDF4 library.

Some features of the HDF4 format are rarely used, and were not found in any of the NASA EOS files examined as part of the Mapping project. Due to budget constraints and project priorities, these features are not all fully supported by the HDF4 File Content Map Schema Version 1.0.0 or the Map Writer. In some cases, the Schema does handle the features, but the Map Writer does not. In other cases, the Schema itself would need extensions to support the features. The tables in Section 10 indicated data type and storage representation features in HDF4 that were not supported by the Schema or by the Map Writer. Other features that are not fully supported are noted in this section.

HDF4 binary files that rely on unsupported features will not be fully represented by the HDF4 File Content Map, and their content will not be fully preserved by the Mapping Transformation. The Map Writer was designed to issue warnings when these features are encountered in an HDF4 file.

The HDF4 File Content Map Schema documents also contain comments that more fully discuss the limitations noted here. Look for the key phrases “Schema Limitation” and “NOTE: h4mapwriter” in the Schema documents.

11.1 Unsupported Compression Schemes for Scientific Data Sets

HDF4 offers a range of compression schemes for raw data in Scientific Data Set Objects. The ones listed here have been commented out of the Schema as they are not implemented and tested in Version 1.0.0 of the Map Writer.

See the Schema Document *HDF4map_byteStream_representations.xsd* for more information.

Scheme	Comment
RLE	Byte-wise Run-Length-Encoding. Supported for General Raster Images, but not for Scientific Data Sets.
Adaptive Huffman	Also known as Skipping Huffman.
Szip	A licensed compression technology. There are 2 versions of szip and they have slightly

	different representations in the HDF4 binary files. That difference would need to be handled if this compression scheme is ever supported.
NBIT	Used by HDF4 to store data with non-standard bit length, specified by the user with a call to *SDsetnbitdataset*. nbit will require considerable effort and additional information from the HDF4 library to implement. It may also require decoding at a different point in the data read/process pipeline than the other compression types.

11.2 Unsupported Compression Schemes for General Raster Images

HDF4 offers a range of compression schemes for raw data in General Raster Image Objects. The ones listed here have been commented out of the Schema as they are not implemented and tested in Version 1.0.0 of the Map Writer.

See the Schema Document *HDF4map_byteStream_representations.xsd* for more information.

Scheme	Comment
DEFLATE	Also known as gzip compression. Supported for Scientific Data Sets, but not for General Raster Images.
Adaptive Huffman	Also known as Skipping Huffman.
Szip	A licensed compression technology. See table entry in section 11.1
JPEG	There are different versions of the JPEG algorithm, but it is not clear that the jpeg versions is kept in the HDF4 file, or that it is needed to uncompress the data. Would require further investigation if this were to be supported.
IMCOMP	This lossy image compression scheme was offered for 8-bit Raster Images, but is not supported by the General Raster interface. Availability of documentation describing it is uncertain, even if it were to be mapped.

11.3 Unsupported External Files for Raw Data

HDF4 allows raw data to be stored in external files for Vdata (Table), SDS (Array), and Raster (Raster) Objects. The Schema provides a mechanism for supporting external files for Tables and Arrays through the ExternalFile element, externalFileIDREF type, and the dataInExternalFile element. Raw data storage in external files was not included for Rasters in Version 1.0.0 of the Schema due to time constraints.

Version 1.0.0 of the Map Writer does not support data in external files for any type of HDF4 Objects.

11.4 Unsupported Raster and Palette Features

Version 1.0.0 of the Schema does handle raw data for Rasters that is stored non-contiguously in the HDF4 file.

Version 1.0.0 of the Schema and Map Writer handle only a subset of the Rasters and Palettes that are supported by HDF4.

- Only Rasters that have 8-bit datatypes, one component per pixel, and pixel interlace mode are supported.
- Rasters created with the HDF4 GR interface can't be in VGroups.
- Palettes must have 256 entries, 3 components per pixel (RGB), and use pixel interlace mode.

See comments in the schema documents HDF4map_rasters.xsd and HDF4map_palettes.xsd for more information.

11.5 Unsupported Annotations

Annotations are only supported by Version 1.0.0 of the Map Writer on the subset of LL-Objects that were annotated in the NASA files that were studied. If Annotations are encountered on other objects, a warning message is issued by the writer and they are not mapped.

11.6 Unsupported Ordering of Raw Data

Version 1.0.0 of the Map Writer does not support Vdata data stored with HDF4 NO_INTERLACE mode. This is supported in the Schema via *tableOrderT*="by column".

Version 1.0.0 of the Map Writer does not support SDS data stored in column-major storage order. This is supported in the Schema via *fastestVaryingDimensionIndex* equal to zero when the number of dimensions in the array is greater than one.

11.7 Unsupported LL-Objects

The HDF4 Documentation, and in some cases the Library, has LL-Objects that are not handled by the HDF4 Map Writer Version 1.0.0, nor in some cases by the schema. Some of these objects were designed to allow future expansion but never implemented. The HDF4 Map Writer checks for them in the HDF4 file that is being mapped, and raises an error if any are encountered.

TAG	Description	Notes
DFTAG_IMCOMP (12)	Image compression for 8-bit images	
DFTAG_TID (102)	Tag Identifier Annotation	
DFTAG_TD (103)	Tag Descriptor Annotation	
DFTAG_MT (107)	Machine Type	Only appears in source .h files. If encountered, make sure preservation metadata for bitstreams appropriately reflects the correct storage representation, taking this into account as well as the Number Type.
DFTAG_II8 (204)	IMCOMP compressed 8-bit image	
DFTAG_LD (307)	Palette Dimensions	
DFTAG_MD (308)	Matte Channel Dimensions	
DFTAG_MA (309)	Matte Channel Data	

DFTAG_CCN (310)	Color Correction	
DFTAG_CFM (311)	Color Format	
DFTAG_AR (312)	Aspect Ratio	
DFTAG_DRAW (400)	Composite Image	
DFTAG_RUN (401)	Run Program or Script	
DFTAG_XYP (500)	X-Y Position	
DFTAG_MT0 (501)	Machine Type Override	If encountered, see also DFTAG_MT # 107
DFTAG_T14 (602)	Textronix 4014 data	
DFTAG_T105 (603)	Textronix 4105 data	
DFTAG_SDT (709)	Transposition	See Table 9f in Spec and text after it in section 9.3.9. Says it is no longer written, but if encountered it is interpreted as originally intended. If we encounter it in mapping, need to make sure Schema makes visible in Preservation Metadata about bitstream.

12 Recommended Comments for HDF4 File Content Maps

In Section 2.1 we defined the primary audience the project is serving, and stated that they may not have information or documentation about HDF4 or the HDF4 File Content Map Schema. The project team felt that some direction should be given to this audience to help them understand how to correctly interpret the information in the Maps, and how to use the information to retrieve and process the necessary bytes from the binary HDF4 files. Because of the complexity of HDF4 and the variety of storage layouts that are possible for the raw data, the steps involved in the processing can be quite involved.

The decision was made to include two categories of comments in the HDF4 File Content Maps to serve this purpose. The comments are not required by the HDF4 File Content Map Schema, but should be included in every HDF4 File Content Map created for a NASA EOS HDF4 file. The HDF4 File Content Map Writer (h4mapwriter) that was developed for the project writes these comments to every Content Map it produces.

In developing the comments, the project team tried to strike a balance between file size and completeness. Some sacrifices were made to readability in order to save space. The expectation is that a person in the future may rely on the comments when they are first looking at a HDF4 File Content Map and developing a reader for the Map and companion HDF4 file, but would not refer to the comments for each and every file they process.

It is possible that HDF4 File Content Maps may also be used by other communities with different audiences, and they may decide the comments are not necessary and can be omitted in their Maps.

The two categories of comments are presented in the next sections.

12.1 Comments Providing Explanation

The first type of comments serve as explanation for the elements in the HDF4 File Content Map and the information they provide. The information in these comment blocks is always the same, regardless of the contents of the HDF4 File Content Map where they appear.

12.1.1 Comment Block One: HDF4FileInformation

The first comment block precedes the *HDF4FileInformation* element and briefly describes the purpose of the file overall as well as the element.

```
<!--

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

The HDF4FileInformation element provides information about the companion HDF4 file.

-->
```

12.1.2 Comment Block Two: HDF4FileContents

The second comment block precedes the *HDF4FileContents* element and is quite extensive. It describes the elements and attributes that can occur in the file, and gives instructions regarding how to use the information they contain to access and process data in the companion HDF4 file to retrieve the raw data values.

```
<!--

The HDF4FileContents element maps the contents of the companion HDF4 file.
Some contents are available directly from this element.
Other contents can be located in the HDF4 file and decoded using information in this element.

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.

HDF4FileContents:
-uses elems to represent objs in the HDF4 file
-uses nested elems and elem references to express relationships of objs to each other
-uses byteStream elems to provide maps to bytes in the HDF4 file that hold raw data for objs
-contains selected raw data values so the reader can verify binary data has been handled properly

Representations:
-Attribute elem represents Attribute or Annotation obj
 -contains user metadata
-Group elem represents Vgroup obj
 -used to associate related objs
 -can form directed graphs
-Table elem represents Vdata obj
 -organizes data into rows (records) and columns (fields)
 -datatype specified per column
 -number of entries per cell for a given column is constant for all rows and may be >1; all entries in a cell have
  same datatype
```

```
-Array elem represents SDS obj
 -organizes data into multi-dimensional array
 -all cells have same datatype
 -references Dimension elem for every dimension that has descriptive metadata
-Dimension elem represents Dimension obj
 -contains descriptive metadata (name, attribute, scale value/coordinate variable) about a dimension for one or more
  SDS objs
-Raster elem represents RIS8 or GR obj
 -two-dimensional [row,column] array of component values representing pixels
 -row 0 corresponds to top of image; column 0 corresponds to left of image
 -one component value per pixel; all values have single-byte datatype
 -references Palette elem to correlate pixel values to colors
-Palette elem represents Palette or Color Lookup Table obj
 -provides colors for images
 -each entry has 3 color components corresponding to Red, Green, Blue

Bytes in HDF4 file (general):
-Raw data are stored in binary format in the HDF4 file. The steps to convert bytes into meaningful raw data values
 depend on the elem and the HDF4 features used when obj was written
-byteStream elems, together with other attrs and elems, provide the information required to convert the binary data
-A datum elem defines the atomic unit of raw data for the obj it applies to
-Each elem in this file that has raw data in the HDF4 file associated with it contains a datum subelem that describes
 the data type, size, and format
 -The number at the end of the dataType is the number of bits per datum; divide by 8 to get number of bytes to process
  together
 -Integers are twos-complement and floating point is IEEE 754-1985

Conversion Instructions:
==Attribute
Raw data stored as single stream of bytes in HDF4 file
-Read bytes
 -If byteStreamSet elem for a given FileAttribute, multiple File Attribute objs were combined in the map file
  Each byteStream elem can be processed separately to reproduce the contents of the individual objs if desired
  (HDF4 limited the size of Attribute objs, so metadata was often split across multiple objs)
-Process
-Access
Note: stringValue (dataType=char8) and numericValues (other dataTypes) let reader access user metadata without
converting binary data; these have been pre-processed to make them more human-readable.  They can also be used to
verify conversion

==Table
-If the Table does not have raw data content associated with it there will not be a tableData subelem
 Only follow these instructions if tableData subelem

Raw data
 -stored as one or more streams of bytes in the HDF4 file or as single stream of bytes in another external file
 -stored by row or by column
-Read bytes
 -If dataInExternalFile elem, get bytes from file in ExternalFile elem
-Prepare to process
 -If >1 row or >1 column, storageOrder indicates if data stored by row or by column
 -If a column has >1 entry, the entries for a given cell are always stored together, regardless of the storageOrder
-Process
 -Follow storage order to determine "current" column and its datum element in this step
-Access
 -Follow storage order to determine dataType
Note: comment block "row(s) for verification" can be used to verify conversion

==Array
-If the Array does not have raw data content associated with it there will not be an arrayData subelem
 Only follow these instructions if arrayData subelem

Raw data
 -may be stored as a complete array (as a whole) or as multi-dimensional sub-arrays (as chunks), indicated by chunks
  subelem
 -may be stored in a compressed format, indicated by compressionType attr
 -are stored as one or more streams of bytes in the HDF4 file or as single stream of bytes in another external file
 -are stored with either rightmost or leftmost array index varying fastest

==Array stored as a whole
```

```
-Read bytes
 -If Array contains all fill values, there is a fillValues element and no bytes are read; use value in that element
  as raw data value for all cells in Array then skip to -Access step
 -If dataInExternalFile elem, get bytes from file in ExternalFile elem
-If data was compressed, uncompress using indicated algorithm
-Prepare to process
 -fastestVaryingDimensionIndex attr indicates storage order when nDimensions >= 1
-Process
 -Follow storage order when processing
-Access
Note: comment block "value(s) for verification" can be used to verify conversion

==Array stored as chunks
With chunked storage, the data for each chunk is processed individually and then positioned in the array
A chunk (sub-array) has the same number of dimensions as the complete array, and the dimension sizes for all chunks
  are the same
Special Notes:
-When the dimension sizes for the complete array are not exact multiples of the dimension sizes for the chunks,
 extra "ghost cells" are stored. Data in ghost cells is not meaningful, but must be read and processed to get proper
 alignment of the actual array data values
-When all data values in a given chunk are equal to the fill value, no bytes are written to the HDF4 file for that
 chunk. The data values for the chunk must all be set to the fill value.

-Read bytes for a single chunk
 -If chunk contains all fill values, there is a fillValues elem instead of byteStream elem for the chunk and no bytes
  are read; use value in fillValues elem as raw data value for all cells in chunk then skip to -Position data... step
-If data was compressed, uncompress using indicated algorithm
-Prepare to process
 -fastestVaryingDimensionIndex attr indicates storage order
 -since data stored chunk-by-chunk, storage order applies on a per-chunk basis
  e.g., for 3x5 array dims, 2x3 chunk dims, and fastestVaryingDimensionIndex=1, data values will be assigned to chunk
  indices [0,0], [0,1], [0,2], [1,0], [1,1], [1,2] as it is processed in the next step
-Process
 -Follow storage order for chunk when processing
 -Interpret the processed bytes based on the storage order and dataType to obtain the raw data values for the cells
  in the current chunk
-Position data values for chunk within allocated array
 -If there are ghost cells, the elem allocatedDimensionSizes indicates the size of the allocated array. If this elem
  is not present, dataDimensionSizes (the dimensions for the actual array) indicates the size of the allocated array
 -Use chunkPositionInArray offsets to position the data values for the current chunk within the allocated array
  e.g., for 3x5 array, 2x4 chunks, 4x8 allocated array, and chunkPositionInArray of [2,4] for the data just read
   allocated array[2,4] = chunk value[0,0] (only cell that isn't a ghost cell in this chunk)
   allocated array[2,5] = chunk value[0,1]
   allocated array[2,6] = chunk value[0,2]
   ...
   allocated array[3,7] = chunk value[1,3]
-Repeat steps for each chunk in the array
-Access
 -Ghost cells do not contain meaningful data values
Note: comment block "value(s) for verification" can be used to verify conversion

==Dimension
A Dimension elem will be present if a dimension has one or more of these: (1) a name (2) an attribute (3) values
If there are no values, the Dimension elem does not have raw data content associated with it and will not contain a
dimensionData subelem. Only follow these instructions if dimensionData subelem

Raw data stored as one or more streams of bytes in the HDF4 file
-Read bytes
-Process
-Access
Note: comment block "value(s) for verification" can be used to verify conversion

==Raster
-If the Raster does not have raw data content associated with it there will not be an rasterData subelem
 Only follow these instructions if rasterData subelem

Raw data stored as single stream of bytes in HDF4 file
-Read bytes
 -If Raster contains all fill values, there is a fillValues element and no bytes are read; use value in that element
  as raw data value for all cells in Raster then skip to -Access step
```

```
-If data was compressed, uncompress using algorithm indicated by compressionType attr
-Prepare to process
 -dimensionStorageOrder attr indicates order of values in file; last dimension always varies the fastest
-Process
 -Follow storage order when processing
-Access
Note: comment block "value(s) for verification; [row,column]" can be used to verify conversion

==Palette
Raw data stored as single stream of bytes in HDF4 file
-Read bytes
-Prepare to process
 -Storage order is entry[0][red]; entry[0][green]; entry[0][blue]; entry[1][red]...
-Process
 -Follow storage order when processing
-Access
Note: comment block "value(s) for verification; csv format" can be used to verify conversion

Calibration:
Calibration Attribute elems (calibrated_nt, scale_factor, scale_factor_err, add_offset, add_offset_err) provide
 calibration information for raw data values of an Elem. To compute original data values apply this formula:
  original_data_value = scale_factor * (raw_data_value - add_offset)
scale_factor_err and offset_error give potential errors due to scaling and offset
calibrated_nt is encoded datatype of original data values: 3=uchar8 4=char8 5=float32 6=float64 7=float128
 20=int8 21=uint8 22=int16 23=uint16 24=int32 25=uint32 26=int64 27=uint64 28=int128 30=uint128 42=char16 43=uchar16
Note: Some files may use the calibration attributes in a manner different than the standard way described.
Consult relevant data product specifications for your particular HDF4 files.

Compression algorithms:
-deflate: also known as gzip or zlib; see IETF RFC1951
-rle: byte-wise run length encoding
 Each run is preceded by a pseudo-count byte
 Low seven bits of the byte indicate the adjusted number of bytes (n)
 If high bit is 1, next byte should be replicated n+3 times
 If high bit is 0, next n+1 bytes should be included in whole

-->
```

12.1 Comments Providing Values for Verification

The second type of comments include raw data values for select cells in a Table (Vdata), Array (SDS), Dimension, Raster, or Palette. These values are included in the comments so they can be compared to the values a reader of the Content Map and binary HDF4 file retrieves at some point in the future, using the instructions in section 12.1.2 . If the values do not match (within floating point precision error), then the reader knows they have not processed the data correctly.

The information in these comment blocks varies depending on the contents in the HDF4 Map file. The format of the comment blocks vary slightly depending on the type and dimensions of the object, but were designed to be machine-parsable. The comment block follows the *objectData* element, where “object” can be “table”, “array”, “dimension”, “raster”, or “palette”.

Values for verification comments are shown for the various types of element, along with the elements.

12.1.1 Table Element and Verification Comment

```
<h4:Table name="strip" path="/Strips/Strip" class="identification" nRows="1" nColumns="5" id="ID_T10">
    <h4:Column name="strip id" nEntries="12" id="ID_C22">
        <h4:datum dataType="char8"/>
    </h4:Column>
    <h4:Column name="time" nEntries="24" id="ID_C23">
        <h4:datum dataType="char8"/>
```

```
    </h4:Column>
    <h4:Column name="node type" nEntries="10" id="ID_C24">
        <h4:datum dataType="char8"/>
    </h4:Column>
    <h4:Column name="node value" nEntries="1" id="ID_C25">
        <h4:datum dataType="float32" byteOrder="bigEndian" floatingPointFormat="IEEE"/>
    </h4:Column>
    <h4:Column name="event count" nEntries="1" id="ID_C26">
        <h4:datum dataType="int32" byteOrder="bigEndian"/>
    </h4:Column>
    <h4:tableData>
        <h4:byteStream offset="33765" nBytes="54"/>
    </h4:tableData>
    <!-- row(s) for verification; csv format
        strip[0]="F 1 2   1 9 9 5 0 8 0 1","1 9 9 5 - 0 8 - 0 1 T 0 2 : 1 0 : 1 4 . 0 0 0  ","A S C E N D I N G
",288.980011,42    -->
</h4:Table>
```

12.1.2 Array Element and Verification Comment

```
<h4:Array name="solzen" path="/L2_Support_atmospheric&amp;surface_product/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="12214592" nBytes="4"/>
        </h4:attributeData>
        <h4:numericValues>-9999.000000</h4:numericValues>
    </h4:ArrayAttribute>
    <h4:dataDimensionSizes>45 30</h4:dataDimensionSizes>
    <h4:dimensionRef name="GeoTrack:L2_Support_atmospheric&amp;surface_product" dimensionIndex="0" ref="ID_D1"/>
    <h4:dimensionRef name="GeoXTrack:L2_Support_atmospheric&amp;surface_product" dimensionIndex="1" ref="ID_D2"/>
    <h4:datum dataType="float32" byteOrder="bigEndian" floatingPointFormat="IEEE"/>
    <h4:arrayData fastestVaryingDimensionIndex="1" compressionType="deflate" deflate_level="1">
        <h4:byteStream offset="36946" nBytes="4384"/>
    </h4:arrayData>
    <!-- value(s) for verification
        solzen[0,0]=154.012939
        solzen[44,0]=147.946030
        solzen[0,29]=169.183838
        solzen[44,29]=158.630493
        solzen[2,6]=158.270035
        solzen[11,12]=160.616913
        solzen[18,16]=161.228180
        solzen[13,2]=155.559036
        solzen[21,3]=155.322067
    -->
</h4:Array>
```

12.1.3 Dimension Element and Verification Comment

```
        <h4:Dimension name="Y_Axis" id="ID_D2">
            <h4:datum dataType="float64" byteOrder="bigEndian" floatingPointFormat="IEEE"/>
            <h4:dimensionData>
                <h4:byteStream offset="14007" nBytes="128"/>
            </h4:dimensionData>
            <!-- value(s) for verification
                Y_Axis[0]=0.000000
                Y_Axis[15]=1.500000
                Y_Axis[12]=1.200000
                Y_Axis[8]=0.800000
                Y_Axis[7]=0.700000
                Y_Axis[3]=0.300000
                Y_Axis[5]=0.500000
            -->
```

12.1.4 Raster Element and Verification Comment

```
<h4:Raster name="" height="1024" width="2048" id="ID_R2">
    <h4:paletteRef ref="ID_P2"/>
    <h4:datum dataType="ubyte8"/>
    <h4:rasterData dimensionStorageOrder="row,column">
        <h4:byteStream offset="2098254" nBytes="2097152"/>
    </h4:rasterData>
    <!-- value(s) for verification; [row,column]
        [0,0]=2
        [0,2047]=2
        [1023,0]=255
        [1023,2047]=255
        [513,683]=211
    -->
</h4:Raster>
```

12.1.5 Palette Element and Verification Comment

```
<h4:Palette nEntries="256" nComponentsPerEntry="3" id="ID_P2">
    <h4:datum dataType="uint8"/>
    <h4:paletteData>
        <h4:byteStream offset="4195406" nBytes="768"/>
    </h4:paletteData>
    <!-- value(s) for verification; csv format
        entry[0]="255,255,255"
        entry[11]="100,100,100"
        entry[129]="255,255,0"
        entry[255]="0,0,0"
    -->
</h4:Palette>
```

13 Additional Information

Additional information about the HDF4 Independent Mapping Project, including the HDF4 File Content Map Schema, is available at: http://www.hdfgroup.org/projects/h4map/

Acknowledgements

This work was supported by a Cooperative Agreement and Contract with the National Aeronautics and Space Administration (NASA) under NASA grant NNX06AC83A and under the Earth Observing System Data and Information Systems (EOSDIS) Evolution and Development (EED) Program under prime Contract number NNG10HP02.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NASA.

Revision History

July 18, 2011:	Version 1 published. Corresponds to version 1.0.0 of the schema.
October 31, 2011:	Version 2 includes changes in Sections 9.4 and 12.1.2 to state that the fastestVaryingDimensionIndex attribute of arrayData is absent if nDimensions is 0. Corresponds to version 1.0.1 of the schema. Also updated Acknowledgements.

February 23, 2012	Version 3 includes changes in Section 10.3.1 noting that character BYTE representation only used with raster data.

.