S-100 – Part 10c

HDF5 Data Model and File Format

Contents

Page intentionally left blank

10c-1 Scope

The Hierachical Data Format 5 (HDF5) HDF has been developed by the HDFgroup as a file format for the transfer of data that is used for imagery and gridded data. This Part specifies an interchange format to facilitate the moving of files containing data records between computer systems. It defines a specific structure which can be used to transmit files containing data type and data structures specific to S-100.

This Part specifies constraints and conventions for HDF5 constructs that exclude HDF5 features not required by S-100 HDF5 datasets and specify rules for S-100 HDF5 data formats. Its scope is limited to the data format and does not include the application schema, nor does it include guidelines for how to develop product specifications or naming rules for features and attributes.

Commented [rmm1]: Sections 10c-1 through 10c-5 are unchanged except for the clarification in 10c-1 and typo and style fixes. All the content beginning from Section 10c-6 is new.

Formatted: Left

10c-2 Introduction

HDF5 uses an open source format. It allows users such as the IHO to collaborate with The HDF Group regarding functionality requirements and permits users' experience and knowledge to be incorporated into the HDF product when appropriate.

HDF5 is particularly good at dealing with data where complexity and scalability are important. Data of virtually any type or size can be stored in HDF5, including complex data structures and data types. HDF5 is portable, running on most operating systems and machines. HDF5 is scalable - it works well in high end computing environments, and can accommodate data objects of almost any size or multiplicity. It also can store large amounts of data efficiently - it has built-in compression. HDF5 is widely used in government, academia, and industry.

10c-3 Conformance

The S-100 HDF5 data format conforms to release 1.8.8 of HDF5.

10c-4 Normative references

The HDF Group, November 2011, HDF5 User's Guide Release 1.8.8.

The HDF Group, November 2011, HDF5 Reference Manual 1.8.8.

10c-5 HDF5 Specification

HDF5 implements a model for managing and storing data. The model includes an abstract data model and an abstract storage model (the data format), and libraries to implement the abstract model and to map the storage model to different storage mechanisms. The HDF5 library provides a programming interface to a concrete implementation of the abstract models. The library also implements a model of data transfer, i.e., efficient movement of data from one stored representation to another stored representation. The figure below illustrates the relationships between the models and implementations.

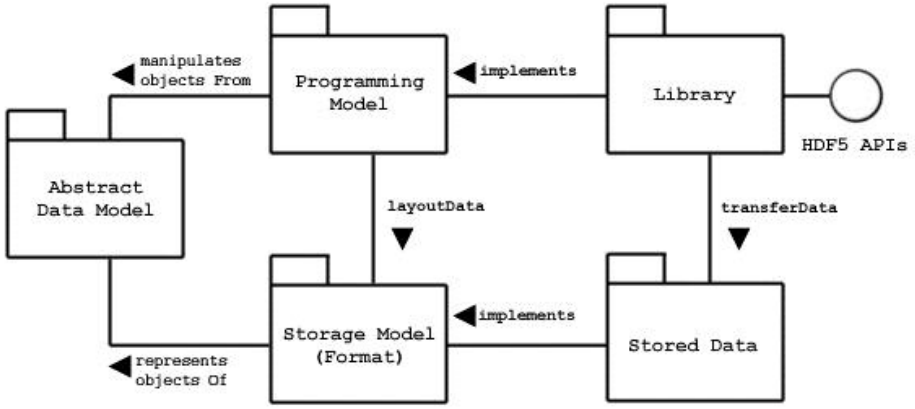

Figure 10c-1 - Abstract Data Model

The *Abstract Data Model* is a conceptual model of data, data types, and data organization. The abstract data model is independent of storage medium or programming environment. The *Storage Model* is a standard representation for the objects of the abstract data model. The *HDF5 File Format Specification* defines the storage model.

The *Programming Model* is a model of the computing environment and includes platforms from small single systems to large multiprocessors and clusters. The programming model manipulates (instantiates, populates, and retrieves) objects from the abstract data model.

The *Library* is the concrete implementation of the programming model. The Library exports the HDF5 APIs as its interface. In addition to implementing the objects of the abstract data model, the Library manages data transfers from one stored form to another. Data transfer examples include reading from disk to memory and writing from memory to disk.

Stored Data is the concrete implementation of the storage model. The storage model is mapped to several storage mechanisms including single disk files, multiple files (family of files), and memory representations.

The HDF5 Library is a C module that implements the programming model and abstract data model. The HDF5 Library calls the operating system or other storage management software (e.g., the MPI/IO Library) to store and retrieve persistent data. The HDF5 Library may also link to other software such as filters for compression. The HDF5 Library is linked to an application program which may be written in C, C++, Fortran, or Java. The application program implements problem specific algorithms and data structures and calls the HDF5 Library to store and retrieve data.

The HDF5 Library implements the objects of the HDF5 abstract data model. Some of these objects include groups, datasets, and attributes. A S-100 product specification maps the S-100 data structures to a hierarchy of HDF5 objects. Each S-100m product specification will create a mapping best suited to its purposes.

The objects of the HDF5 abstract data model are mapped to the objects of the HDF5 storage model, and stored in a storage medium. The stored objects include header blocks, free lists, data blocks, B-trees, and other objects. Each group or dataset is stored as one or more header and data blocks.

10c-5.1 Abstract Data Model

The abstract data model (ADM) defines concepts for defining and describing complex data stored in files. The ADM is a very general model which is designed to conceptually cover many specific models. Many different kinds of data can be mapped to objects of the ADM, and therefore stored and retrieved using HDF5. The ADM is not, however, a model of any particular problem or application domain. Users need to map their data to the concepts of the ADM.

The key concepts include:

- *File* - a contiguous string of bytes in a computer store (memory, disk, etc.), and the bytes represent zero or more objects of the model;
- *Group* - a collection of objects (including groups);
- *Dataset* - a multidimensional array of data elements with attributes and other metadata;
- *Dataspace* - a description of the dimensions of a multidimensional array;
- *Datatype* - a description of a specific class of data element including its storage layout as a pattern of bits;
- *Attribute* - a named data value associated with a group, dataset, or named datatype;
- *Property List* - a collection of parameters (some permanent and some transient) controlling options in the library;
- *Link* - the way objects are connected.

These key concepts are described in more detail below.

10c-5.1.1 File

Formatted: Font: Bold

Abstractly, an HDF5 file is a container for an organized collection of objects. The objects are groups, datasets, and other objects as defined below. The objects are organized as a rooted, directed graph. Every HDF5 file has at least one object, the root group. See the figure below. All objects are members of the root group or descendents of the root group.

HDF5 objects have a unique identity *within a single HDF5 file* and can be accessed only by its names within the hierarchy of the file. HDF5 objects in different files do not necessarily have unique identities, and it is not possible to access a permanent HDF5 object except through a file.

When the file is created, the *file creation properties* specify settings for the file. The file creation properties include version information and parameters of global data structures. When the file is opened, the *file access properties* specify settings for the current access to the file. File access properties include parameters for storage drivers and parameters for caching and garbage collection. The file creation properties are set permanently for the life of the file, and the file access properties can be changed by closing and reopening the file.

An HDF5 file can be “mounted” as part of another HDF5 file. This is analogous to Unix file system mounts. The root of the mounted file is attached to a group in the mounting file, and all the contents can be accessed as if the mounted file were part of the mounting file.

10c-5.1.2 Group

Formatted: Indent: Left: 0 cm, First line: 0 cm

An HDF5 group is analogous to a file system directory. Abstractly, a group contains zero or more objects, and every object must be a member of at least one group. The root group is a special case; it may not be a member of any group.

Group membership is actually implemented via link objects. See the figure below. A link object is owned by a group and points to a named object. Each link has a name, and each link points to exactly one object. Each named object has at least one and possibly many links to it.

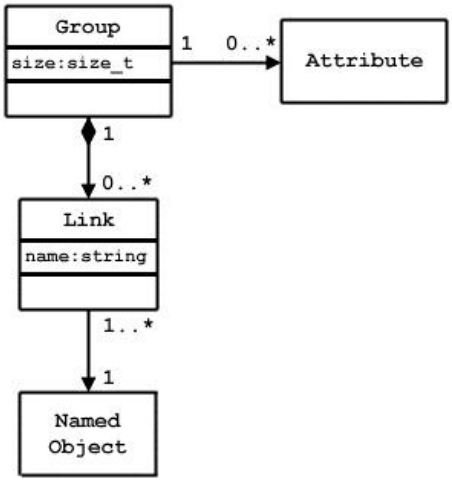

Figure 10c-2 - Group membership via link objects

There are three classes of named objects: group, dataset, and named datatype. See the figure below. Each of these objects is the member of at least one group, and this means there is at least one link to it.

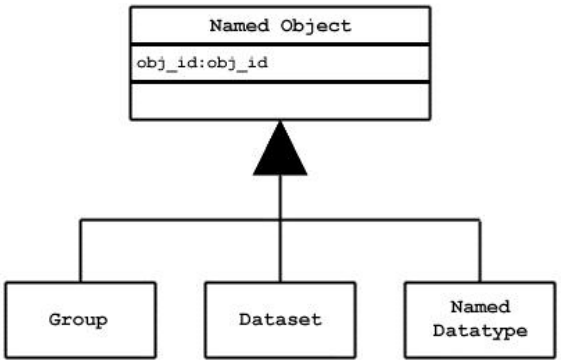

Figure 10c-3 - Classes of named objects

10c-5.1.3 Dataset

Formatted: Indent: Left: 0 cm, First line: 0 cm

An HDF5 dataset is a multidimensional array of data elements. See the figure below. The shape of the array (number of dimensions, size of each dimension) is described by the dataspace object.

A data element is a single unit of data which may be a number, a character, an array of numbers or characters, or a record of heterogeneous data elements. A data element is a set of bits. The layout of the bits is described by the datatype.

The dataspace and datatype are set when the dataset is created, and they cannot be changed for the life of the dataset. The dataset creation properties are set when the dataset is created. The dataset creation properties include the fill value and storage properties such as chunking and compression. These properties cannot be changed after the dataset is created.

The dataset object manages the storage and access to the data. While the data is conceptually a contiguous rectangular array, it is physically stored and transferred in different ways depending on the

storage properties and the storage mechanism used. The actual storage may be a set of compressed chunks, and the access may be through different storage mechanisms and caches. The dataset maps between the conceptual array of elements and the actual stored data.

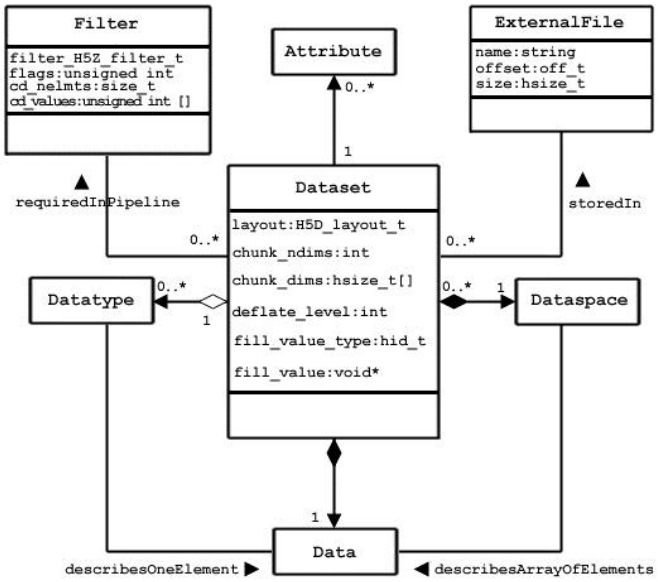

Figure 10c-4 - The dataset

10c-5.1.4 Dataspace

The HDF5 dataspace describes the layout of the elements of a multidimensional array. Conceptually, the array is a hyper-rectangle with one to 32 dimensions. HDF5 dataspaces can be extendable. Therefore, each dimension has a current size and a maximum size, and the maximum may be unlimited. The dataspace describes this hyper-rectangle: it is a list of dimensions with the current and maximum (or unlimited) sizes.

10c-5.1.5 DataType

The HDF5 datatype object describes the layout of a single data element. A data element is a single element of the array; it may be a single number, a character, an array of numbers or carriers, or other data. The datatype object describes the storage layout of this data.

Data types are categorized into 11 classes of datatype. Each class is interpreted according to a set of rules and has a specific set of properties to describe its storage. For instance, floating point numbers have exponent position and sizes which are interpreted according to appropriate standards for number representation. Thus, the datatype class tells what the element means, and the datatype describes how it is stored.

The figure below shows the classification of datatypes. Atomic datatypes are indivisible. Each may be a single object; a number, a string, or some other objects. Composite datatypes are composed of multiple elements of atomic datatypes. In addition to the standard types, users can define additional datatypes such as a 24-bit integer or a 16-bit float.

A dataset or attribute has a single datatype object associated with it. See the Dataset Figure above. The datatype object may be used in the definition of several objects, but by default, a copy of the datatype object will be private to the dataset.

Optionally, a datatype object can be stored in the HDF5 file. The datatype is linked into a group, and therefore given a name. A *named datatype* can be opened and used in any way that a datatype object can be used.

Formatted: Indent: First line: 0 cm

Formatted: Indent: First line: 0 cm

Not all the HDF5 datatypes have exactg equivalents in the S-100 basic and derived datatypes defined in clause 1-4.5.2 (Table 1-2). The correspondences between HDF5 and S-100 datatypes are given in Table 10c-2 later in thie Part.

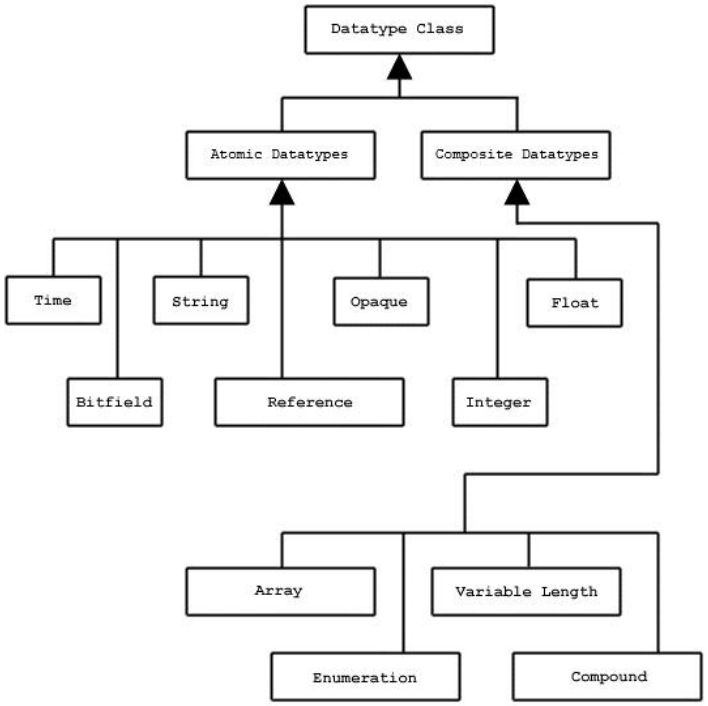

Figure 10c-5 - Datatype classifications

10c-5.1.6 Attribute

Formatted: Indent: First line: 0 cm

Any HDF5 named data object (group, dataset, or named datatype) may have zero or more user defined attributes. Attributes are used to document the object. The attributes of an object are stored with the object.

An HDF5 attribute has a name and data. The data portion is similar in structure to a dataset: a dataspace defines the layout of an array of data elements, and a datatype defines the storage layout and interpretation of the elements. See the figure below.

Attributes of data objects are in principle equivalent to thematic attributes but this edition of the HDF5 profile does not provide for vector feature or information type data in HDF5 files iand therefore does not make use of vector object attributes. HDF5 attributes of groups, datasets, or named datatypes play the role of metadata.

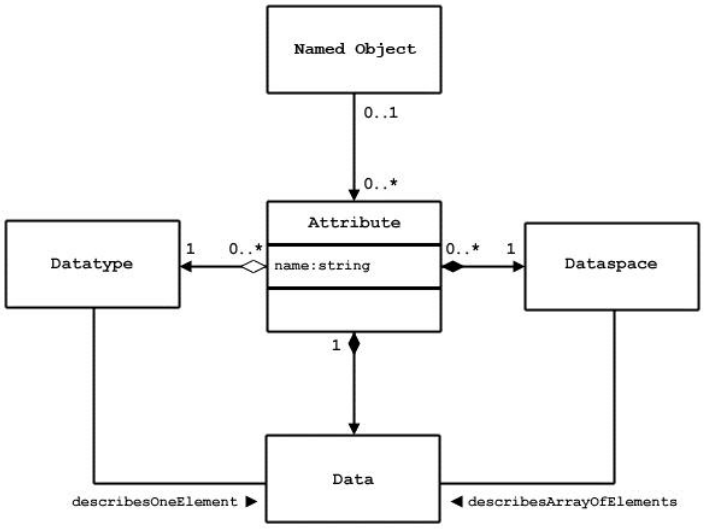

Figure 10c-6 - Attribute data elements

In fact, an attribute is very similar to a dataset with the following limitations:

- An attribute can only be accessed via the object;
- Attribute names are significant only within the object;
- An attribute should be a small object;
- The data of an attribute must be read or written in a single access (partial reading or writing is not allowed);
- Attributes do not have attributes.

Note that the value of an attribute can be an *object reference*. A shared attribute or an attribute that is a large array can be implemented as a reference to a dataset.

The name, dataspace, and datatype of an attribute are specified when it is created and cannot be changed over the life of the attribute. An attribute can be opened by name, by index, or by iterating through all the attributes of the object.

10c-5.1.7 Property List

Formatted: Indent: First line: 0 cm

HDF5 has a generic property list object. Each list is a collection of *name-value* pairs. Each class of property list has a specific set of properties. Each property has an implicit name, a datatype, and a value. A property list object is created and used in ways similar to the other objects of the HDF5 library.

Property Lists are attached to the object in the library, they can be used by any part of the library. Some properties are permanent (e.g., the chunking strategy for a dataset), others are transient (for example buffer sizes for data transfer). A common use of a Property List is to pass parameters from the calling program to a VFL driver or a module of the pipeline.

Property lists are conceptually similar to attributes. Property lists are information relevant to the behavior of the library while attributes are relevant to the user's data and application. Since the Property ~~n~~List couples the data specification to an implementation use of HDF5 property lists in S-100 Product Specifications is discouraged.

10c-5.2 HDF5 Library and Programming Model

Formatted: Indent: Left: 0 cm, First line: 0 cm

The HDF5 Library implements the HDF5 abstract data model and storage model. Two major objectives of the HDF5 products are to provide tools that can be used on as many computational platforms as

possible (portability), and to provide a reasonably object-oriented data model and programming interface.

~~To be as portable as possible, the HDF5 Library is implemented in portable C. C is not an object-oriented language, but the library uses several mechanisms and conventions to implement an object model.~~

~~One mechanism the HDF5 library uses is to implement the objects as data structures. To refer to an object, the HDF5 library implements its own pointers. These pointers are called identifiers. An identifier is then used to invoke operations on a specific instance of an object. For example, when a group is opened, the API returns a group identifier. This identifier is a reference to that specific group and will be used to invoke future operations on that group. The identifier is valid only within the context it is created and remains valid until it is closed or the file is closed. This mechanism is essentially the same as the mechanism that C++ or other object-oriented languages use to refer to objects except that the syntax is C.~~

~~Similarly, object-oriented languages collect all the methods for an object in a single name space. An example is the methods of a C++ class. The C language does not have any such mechanism, but the HDF5 Library simulates this through its API naming convention. API function names begin with a common prefix that is related to the class of objects that the function operates on. The table below lists the HDF5 objects and the standard prefixes used by the corresponding HDF5 APIs. For example, functions that operate on datatype objects all have names beginning with H5T.~~

Commented [E2]: Compared with the other encodings and other parts of S-100, this information is out of place. Suggest to consider remove and include only a reference to the HDF working group website for information on libraries.

~~Prefix~~	~~Operates on~~
~~H5A~~	~~Attributes~~
~~H5D~~	~~Datasets~~
~~H5E~~	~~Error reports~~
~~H5F~~	~~Files~~
~~H5G~~	~~Groups~~
~~H5I~~	~~Identifiers~~
~~H5L~~	~~Links~~
~~H5O~~	~~Objects~~
~~H5P~~	~~Property lists~~
~~H5R~~	~~References~~
~~H5S~~	~~Dataspaces~~
~~H5T~~	~~Datatypes~~
~~H5Z~~	~~Filters~~

~~**Table 10c-11 - The HDF5 API naming scheme**~~

Refer the HDF5 User's Guide Release 1.8.8 and the HDF5 Reference Manual 1.8.8 for more details on the HDF5 model implementation. S-100 Product Specifications must specify the HDF5 groups, datasaetsets and attributes in context of the S-100 General Feature Model.

10c-6 S-100 profile of HDF5

The S-100 profile of HDF5 restricts the HDF5 datatypes and constructs which can be used in S-100 HDF5 datasets, describes correspondences between S-100 and HDF5 datatypes and other constructs, and defines rules for how S-100 HDF5 datasets must be structured.

The S-100 HDF5 profile must apply to the kinds of information listed below – noting that the types are not all mutually exclusive, though most individual product specifications will use only a subset of possible combinations:

- data for one or more individual, fixed stations,
- regularly-gridded data,
- irregularly-gridded data,
- grids with variable cell sizes,
- ungeorectified gridded data,
- TIN data,

- moving platform (e.g., surface drifter) data,
- either static data or time series data (for any of the other kinds), with fixed or variable intervals,
- tiled and untiled coverages,
- multiple feature classes in the same datafile,
- multiple types of coverages in the same datafile.

The restrictions, correspondences, and rules are described in the following sections.

10c-7 Data types

Predefined HDF5 data types include Integer, Float, String, and Enumeration but not Boolean, S100_Codelist, S100_TruncatedDate. The classes Date, DateTime, and Time are mapped to HDF5 strings due to potential problems with portability across different processor architectures of HDF5 Time formats. In S-100 HDF5 data products, S-100 data types defined in Part 3 are ~~replaced~~ mapped to ~~by~~ equivalent HDF5 data types. These equivalences are summarized in Table 10c-2 below. HDF5 datatype classes not mentioned in this table shall not be used.

S-100 Attribute Value Types	HDF5 Datatype Class	Constraint on HDF5 datatype
real	Float	64-bit floating point
integer	Integer	1, 2, or 4-byte signed and unsigned integers
text (CharacterString in S-100 metadata)	String	variable-length string
enumeration	Enumeration	Numeric codes must be 1 or 2-byte unsigned integers, range [1, 2^8 – 1] or [1, 2^{16} - 1].
date	(Character) String, length=8	Date format according to Table 1-2 (Part 1), i.e., complete representation, basic format, as specified by ISO 8601.
time	(Character) Variable-length string, 6-7 characters	Time format according to Table 1-2 (Part 1), i.e., complete representation, basic format as specified by ISO 8601. UTC indicated by "Z" suffix; local time by absence of suffix.
dateTime	(Character) (variable length string)	Date-time format as specified by ISO 8601. EXAMPLE: 19850412T101530Z
boolean	(Integer)	1-byte unsigned, Values: 1 (TRUE); 0 (FALSE)
S100_Codelist	Compound (Enumeration, variable-length string)	Exactly one of the components is allowed; the other must be the numeric value 0 or the empty (0-length) string according to its data type.
URI, URL, URN	String (variable-length)	Format specified in RFC 3986 (URI, URL) or RFC 2141 (URN)
S100_TruncatedDate	String, length=8	Format as in Part 1 Table 1-12
value record (Part 8)	Compound	Datatypes of components according to attribute types in application schema. Product specifications may split the components of the "value record" across HDF5 datasets, i.e., arrays.

Table 10c-~~1~~2 – Equivalences between S-100 and HDF5 datatypes

Commented [rmm3]: The restrictions on the sizes of real and integer types are needed only if compatibility with the ISO 8211 format must be maintained. Part 10a currently allows 1/2/4 byte signed and unsigned integers and 8-byte floating point. However, the actual necessity of maintaining compatibility is doubtful since ISO 8211 and HDF5 are for different types of data, so unless the necessity of compatibility can be justified these restrictions will be removed.

Commented [E4R3]: I cannot think of any reason to maintain compatability with 8211 at this point. If there is a need to convert data from HDF5 to 8211, I suggest this is a bespoke operation that is out of scope for the standard.

Commented [rmm5]: Table 1-2 needs to be amended to correct and clarify usage of suffixes and zone offsets.

10c-8 Naming conventions

Names of HDF5 elements (datasets, objects, etc.) that encode data elements in the Application Schema (i.e., feature classes, attributes, roles, enumerations, codelists, etc.) must conform to the names in the Application Schema. Other sections in this Part indicate where the names from the Application Schema (or equivalently, the Feature Catalogue) are used.

Elements in embedded ("carrier") metadata and positioning information may have names that are unique to the HDF5 format (the differences being intended to simplify the abstractions in ISO 19123 and S-100 Parts 4, 4b, and 8, and relate to prior work). 'Latitude' and 'Longitude' must be used for geographic coordinate axes when they are appropriate, in preference to 'X' and 'Y', which should be used only when

latitude/longitude are inappropriate. The correpondences between the carrier metadata elements in this profile and Part 4-4c and Part 8 are specified later in this document.

Names in non-embedded metadata and catalogue files in exchange sets are treated as for vector product product specifications – i.e., they must conform to the standard S-100 metadata and exchange catalogue schemas.

An HDF5 group which corresponds to an element named elsewhere in S-100 or in the product specification must be given the same name as that element, using the camel-case code for the element if specified. For example, if a time series product specifies names for data collections at time points, those names should be used as the group names if the collection is encoded as a group. (In that event, specification developers must take care to specify collection names which conform to the allowed HDF5 syntax.)

Numeric suffixes preceded by the underscore character (i.e., the suffix '_NNN') may be added to distinguish groups which would otherwise have the same names (for example, data groups at different time points).

~~Groups that do not correspond to named elements in either S-100 or the product specification may be given any name in the Data Format section of the product specification, except that t~~The following group names are reserved for the uses specified:

~~Group_XY~~	~~2D positioning information as discrete coordinates. Includes compressed or compact encodings. Does not include positioning which can be completely specified by grid or coverage parameters alone.~~
~~Group_XYZ~~	~~3D positioning information as discrete coordinates. Includes compressed or compact encodings. Does not include positioning which can be completely specified by grid or coverage parameters alone.~~
~~Group_P~~	~~Other forms of discrete position information, e.g., spatial coordinates plus a time axis, or higher-dimensionality positioning. Includes compressed or compact encodings. Does not include positioning which can be completely specified by grid or coverage parameters.~~
Positioning	Discrete positioning information of all kinds and dimensions. The type of positioning data is indicated by a group attribute or attributes. Includes compressed or compact encodings. Does not include positioning which can be completely specified by grid or coverage parameters alone (such parameters are encoded in attributes attached to the root group).
Group_F	Feature specification information. E.g., feature and attribute names, codes, types, multiplicities, roles, etc. Also includes format metadata specific to the HDF5 format, like chunk sizes.
Group_IDX	Indexes, if encoded in an HDF5 group. Includes indexes to sparse arrays.
Group_TL	Tiling information, if encoded in a group.
Group_nnn	Data for one member of a series, e.g., at a time point in a time series, or for different stations. "n" means any digit from 0 to 9. Numbering must use 3 digits, 000-999.

Table 10c-~~23~~ – Reserved group names

Formatted: Strikethrough

Formatted: Strikethrough

Formatted: Strikethrough

Commented [rmm6]: Alternative to the preceding 3 groups. I would prefer a single group „Positioning" with an attribute (or attributes) that indicate the type of positioning information, because positioning information will be 2D, 3D (XYZ or XYT), perhaps 4D (XYZT).
The group name isn't important as long as it is standardized.

~~10c-8~~10c-9 Structure of data product

10c-9.1 General structure

An S-100 HDF5 file is structured to consist of Groups, each of which may contain other Groups, Attributes and Datasets. Groups are containers for different types of information (meaning data values, position information, metadata, or ancillary information). Datasets are designed to hold large amounts of numerical data and may be used to hold the coverage data values. Attributes are designed to hold single-valued information which apply to Groups or Datasets and may be used to hold certain types of metadata.

The order of groups within the root group follows the following sequence:

1) Feature information group.
2) Feature container groups – each acts as a container for the positioning, tile, indexes, and data groups pertaining to a single feature class. Its attributes encode any feature-class-level metadata.
    a. Tiling information group (conditional, only if values are stored as tiles).
    b. Indexes group (conditional, only if indexes to data are required).
    c. Positioning group (conditional, only if postions are not computable from metadata).
    d. Data values group(s).

~~Given the nature of coverage information this edition of the HDF5 profile assumes that there is only one instance of a feature class within each HDF5 datafile.~~ Data products which need to provide more than one instance of a feature class may provide a ~~distinct HDF5 datafile for~~ named group for each instance under the feature container group. Names of instances should be derived from identification attributes for the instances.

10c-9.2 Metadata

10c-9.2.1 Discovery metadata

Discovery metadata is encoded in the usual way in an external discovery metadata file, as specified in Parts 4a (Metadata) and 4b (Metadata for Imagery and Gridded Metadata).

10c-9.2.2 Carrier (embedded) metadata

Carrier metadata is metadata that is encoded within the HDF5 file. It is divided into general, type, and instance metadata, depending on whether it pertains to the HDF5 file as a whole, describes the structure and attributes of data object classes, or provides parameters needed to read instances of data object classes. Metadata is encoded in the following places:

- General metadata, defined as general parameters that apply to the file as a whole. General metadata consists of parameters that apply to all information in the data file, such as dates of issue, datum information, and overall spatial extent (bounding box). This includes the essential general elements for processing and cell location (the rest of the essential information is encoded with the feature instance). This metadata is encoded as attributes of the root group.
- Type metadata, defined as specific characteristics which describes data object classes in the file (e.g., pertains to specific features and attributes) and which will therefore be different for each feature ~~or information~~ class. This metadata is used for feature and attribute specification information (corresponding to entries in the feature catalogue). This ~~information~~type information is analogous to the feature catalogue described in Part 5, but may contain only extracts from the feature catalogue as well as add format-specific paramters relevant only to HDF5 encodings. The Type Metadata is encoded as content (HDF5 datasets) in the feature information group. The feature information group (Group_F) is also the future intended container for information from the exchange set catalogue or about support files, if it is necessary to include that within the HDF5 file and it is not applicable to the file as a whole.
- Instance metadata, defined as parameters ~~that are defined~~ for each feature class in the application schema. This includes parameters that are needed to read the information in the data product even if external metadata files are unavailable, including coverage-specific spatial parameters (extent, grid parameters). This metadata may include parameters that have significance only in the context of the specific coverage spatial type(s) permitted for the feature class in the application schema. This metadata is encoded as attributes of each feature container group.

Formatted: Strikethrough

10c-9.2.3 Extended metadata

Extended metadata elements defined in the product specification are encoded as either or both of:

- Additional attributes of the root of feature container group, depending on whether they are considered necessary for processing and pertain to the datafile as a whole or to feature instances.
- Extended metadata in the external XML files encoding the discovery metadata or feature catalogue, if they are considered discovery metadata.

Data products may also define vector feature metadata, e.g., quality meta-features with vector geometry. Vector features are not encoded within the HDF5 file but in a separate file conforming to Part 10a or Part

10b. If vector meta-features are present, a reference to the separate file must be included in carrier metadata by naming the file in the metaFeatures attribute (see section 10c-9.4).

10c-9.3 Generalized dimensions

For non-regularly gridded data only, there is an initial Group with positioning information. The nature of the positioning information depends on the data type.

- For fixed stations, ungeorectified grid, and moving platform, the positioning information is stored in one-dimensional arrays of size numPOS.
- For irregular grids, the positioning information is stored as regular grids with the missing cells populated with the "unknown" fill value. [More appropriate solution TBD – lists of cells that are populated with data, with some form of compaction? Linear scale arrays? A tree index to populated cells?].
- For variable cell sizes, the positioning information is stored as [TBD – lists of cells populated with data, along with cell extents? Linear scale arrays? A tree index to populated cells?] The format assumes that the varying cells are aligned with the grid and that cell sizes are multiples of unit cell size in each dimension.
- For TIN data, the positioning information is stored as one-dimensional arrays of size numPOS encoding the vertex locations plus a Triangles array encoding references to the vertices of the triangle and references to adjacent triangles.

Data Groups are separate groups containing the data values, which (for 2-d data) are stored in two-dimensional arrays of size numROWS by numCOLS. The total number of data Groups is numGRP. The meaning of numGRP for each type of spatial representation is specified in Table 10c-~~8.~~4 The format allows for time series data for all representations.

For 3-dimensional data, the vertical dimension is added.

The variables that determine the array sizes (numROWS, numCOLS. numPOS, and numGRP) are different, depending upon which coding format is used. Their descriptions are given in Table 10c-~~8.~~4

Coding Format	Data Type	numPOS	numCOL	numROW	numZ (3-d only)	numGRP
1	Fixed Stations	numberOfStations	numberOfTimes	1	1	numberOfStations
2	Regular Grid	(not used)	numPointsLongitudinal	numPointsLatitudinal	numPointsVertical	numberOfTimes
3	Ungeorectified Grid	numberOfNodes	numberOfNodes	1	1	numberOfTimes
4	Moving Platform	numberOfTimes	numberOfTimes	1	1	1
5	Irregular Grid	TBD	TBD	TBD	TBD	numberOfTimes
6	Variable cell size	TBD	TBD	TBD	TBD	numberOfTimes
7	TIN	numberOfNodes	numberOfNodes	1	1	numberOfTimes

Table 10c-34 – Array dimensions for different types of coverages

The name of each data Group begins with the characters 'Group_nnn', where n is numbered from 1 to numGRP. A maximum of 999 data groups are allowed. The length of the data group name is 9.

For all data types, the product structure in HDF5 includes (a) a metadata block, which is followed by (b) the feature information group, then (c) one or more data container groups, each of which contains tiling, indexing, positioning and data groups as described in section 10c-9.1. The tiling, indexing, and positioning groups are conditionally required depending on the type of data, indicated by an HDF5 attribute that specifies the coding format.

The following sections describe the content and attributes of each group.

10c-9.4 Root group

The root group acts as a container for the other groups. The carrier metadata (Table 10c-4)6) is contained as attributes in the root group. The carrier metadata consists of the data and parameters (a) needed to read and interpret the information in the product even if external metadata files are unavailable, and, mostly, (b) are not included elsewhere in the metadata.

Group	HDF5 Category	Name		Data Type	Data Space / Remarks
/ *(root)*	Attributes	(Carrier metadata attributes)		Integer, Float, Enumeration , or String	(none) Described in Table 10c-9.4
	Group	Group_F			Feature information group (see section 10c-9.6)
	Group(s)	(featureCode)			Feature container group – one group for each teature in the data product. The name is the feature code, which is given in Group_F. See Section 10c-9.6 for structure and attributes
	HDF5 Category	**Name**			
	Group (optional)	Group_TL			Tiling information, only if product uses tiles. See section 10c-9.7
	Group (optional)	Group_IDX			Spatial index information, only if product uses spatial indexes See section 10c-9.8
	Group	Positioning			Positioning information – 2D or 3D. Not required for dataEncodingFormat = 2 (Regular grid). See section 10c-9.9
	Group(s)	Group_NNN			Static data – only 1 values group Time series data – 000 to 999 groups See section 10c-9.10

Table 10c-45 - Root group

The attributes of the root group are listed in Table 10c-6. The root group contains only a subset of the elements of minimum metadata specified in Parts 4a and 4b. The external XML metadata file is required to contain all the mandatory metadata elements.

No.	Name	Camel Case	Mult.	Data Type	Remarks and/or Units
1	Product specification number and version	productSpecification	1	String	This must be encoded as 'S-NNN.X.X.X', with Xs representing the version number. "NNN" and "X" do not imply length restrictions. Corrresponds to combination of S100_ProductSpecification name and number fields.
2	Date-Time of data product issue	dateTimeOfIssue	0..1	String (DateTime format)	Must be consistent with issueDate in discovery metadata. Products must encode either this or issueDate.
3	Issue date	issueDate	0..1	String (Date format)	Must be consistent with issueDate in discovery metadata. Products must encode either this or dateTimeOfIssue.
4	Horizontal datum	horizontalDatumReference	1	String	EPSG

Commented [rmm7]: Q: How is S-101 doing this? Harmonize with other product specifications when a common format is defined.

Commented [E8R7]: S-101 follows S-100 on this point. Or do you mean in the 8211 encoding? In B1.5.2 it is "INT.IHO.S-101.1.0", but I am not sure there has been a change to the 8211 encoding of S-101 since the release of S-100 Ed 3.0.0

Commented [rmm9]: Q. for S-111 team – can this be replaced with timeOfIssue since there is a separate issueDate attribute anyway?

5	Horizontal datum number	horizontalDatumValue	1	Integer	4326 (for WGS84)
6	Bounding box	boundingBox	1	Compound (Float X 4)	Components: westBoundLongitude eastboundLongitude southBoundLatitude northBoundLatitude Ref. dataCoverage.boundingBox > EX_GeographicBoundingBox
7	Geographic location of the resource (by description)	geographicIdentifier	0..1	String	EX_Extent > EX_GeographicDescription.geographicIdentifier > MD_Identifier.code
8	Metadata	metadata	1	String	MD_Metadata.fileIdentifier Name of XML metadata file. Ref. Part 8.
9	Vertical reference	depthTypeIndex	1	Enumeration	1: Layer average 2: Sea surface 3: Vertical datum (see verticalDatum) 4: Sea bottom
10	Vertical datum reference	verticalDatum	0..1	Enumeration	See S100_VerticalAndSoundingDatum Conditional, iff depthTypeIndex=3
11	Meta features	metaFeatures	0..1	String	Name of 8211 or GML file containing meta-features

Commented [E10]: Noting Note 4) below. It makes more sense to me to include a standard way of how to extend the carrier metadata if this is needed, and the scrapping this. Or at least making it optional.

Commented [RM11R10]: Suggest extending simply by adding the extension attributes.

Table 10c-~~5~~6 – Embedded metadata (carrier metadata) in root group

Notes:

1) The bounding box is the cell bounding box; the coverage data feature instances may or may not cover the entire bounding box. If there is only a single coverge feature, its extent may or may not be the same as the cell.
2) Except for dateTimeOfIssue, geographicIdentifier and depthTypeIndex, the other attributes correspond to metadata attributes in S100 DatasetDiscoveryMetadata (Part 4a) or the imagery/gridded/coverage data attributes in Part 8.
3) The attribute dateTimeOfIssue is added to accommodate some products' need to encode time of issue.
4) Vertical datum is only conditionally mandatory since it is not applicable to some types of depth referencing as used in some data products, e.g., S-111.

Product specifications which need additional metadata attributes may include them as additional attributes, defined in the product specification.

10c-9.5 Feature information group

The feature information group contains the specifications of feature classes and their attributes. The componenets of the feature information group are described in the table below. The "additional attribute characteristics" are information that characterizes attributes but is not common to all attributes (e.g. pattern constraints for string attributes, range or unit of measure information for numeric attributes).

Group	HDF5 Category	Name	Data Type or HDF Category	Data Space
/Group_F	Dataset	featureCode	String (variable length)	Array (1-d): i=0, F-1 Values = codes of feature classes (F is the number of feature classes in the application schema.)
	Dataset	numAttributes	Integer	Array (1-d): i=0, F-1 (Ni=number of feature attributes for feature i)

	Dataset	numAdditionalCharacteristics	Integer	Array (1-d): i=0, F-1 (Xi=number of additional metadata characteristics for feature i)
	Dataset(s)	attributes-f e.g., attributes-00, attributes-01, etc.	Attribute	Attribute name = featureCode Type = string value = featureCode[f]
			String (variable length)	Array (2-d): i=0, *Ni-1, j=0,5* Col [i,0]: camel case code of attribute as in feature catalogue Col [i,1]: long name as in feature catalogue Col [i,2]: units (uom.name from S-100 feature catalogue) Col [i,3]: fill value (integer or float value, string representation) Col [i,4]: chunk sizes (chunk sizes, string representation) Col [i, 5]: HDF5 data type, as returned by H5Tget_class() function Columns 0 and 5 encode the rangeType attribute of the coverage features in Part 8.
	Dataset(s)	additionalAttribute_Characteristics-f	Attribute	Attribute name = featureCode Type = string value = featureCode[f]
			String (variable length)	Array (1-d): i=0,Xf-1 Xf=number of additional characteristics for feature f. Identifier tags for any extended attribute characteristics the product specification requires. E.g., Standard name of units from CF conventions. If there are no additional characteristics (Xf=0), this dataset is not encoded at all.
	Dataset(s)	Additional_Characteristic_Values-f	Attribute	Attribute name = featureCode Type = string value = featureCode[f]
			String (variable length)	Array (2-d): i=0, Nf-*1, j=0.Xf* Nf=number of attributes for feature f. Xf=number of additional characteristics for feature f. If none, this dataset is not encoded. Col [i, 0]: camel case code of attribute as in feature catalogue. Col [i, 1..Xf]: characteristic, string representation. Sequence must correspond to Additional Attribute Characteristics array for feature f.

Table 10c-67 - Components of feature information group

EXAMPLE:

Dataset featureCode

index	Values
0	SurfaceCurrent

Table 10c-78 - Example of feature codes array for feature information group

Dataset numAttributes

Index	Values
0	2

Commented [E12]: What are these and how will they be used? Reading from below, it seems to be a means to, for example, encode one attribute as metres, and the next as centimetres? If this is the intent, it makes for complications in implementation with ECDIS and any other system that can use several types of data together. How will a standard implementation be made ready to recalculate values into a global value? I think more thought and explination is needed with these.
Or are these the method to encode complex attributes?

Commented [RM13R12]: Feature catalog model already allows encoding units of measure.

Table 10c-89 - Example of attribute count array for feature information group

Dataset numAdditionalCharacteristics

Index	Values
0	3

Table 10c-910 - Example of additional attribute characteristics array for feature information group

Dataset Attributes-00:
Attribute featureCode = SurfaceCurrent

(All the numeric values are string representations of the numeric value, e.g., "-9999.0" not the float value -9999.0. Applications are expected to parse the strings to obtain the numeric value. Inapplicable entries are represented by the empty (0-length) string.)

row/col index	0	1	2	3	4	5
0	surfaceCurrentSpeed	Surface current speed	knots	-9999.00	""	H5T_FLOAT
1	surfaceCurrentDirection	Surface current direction	degrees	-1.0	96,56	H5T_FLOAT

Table 10c-1011 - Example of attributes array for feature information group

Dataset Additional_Attribute_Characteristics-00
Attribute featureCode = SurfaceCurrents

index	
0	lower
1	upper
2	closure

Table 10c-1112 - Example of encoding attribute value range for feature information group

Dataset Additional_Characteristic_Values-00
Attribute featureCode = SurfaceCurrents

row/col index	0	1	2	3
0	surfaceCurrentSpeed	0.00	""	geSemiInterval
1	surfaceCurrentDirection	0.0	359.9	closedInterval

Table 10c-1213 - Example of encoding attribute value range for feature information group

10c-9.6 Feature container group

Group	HDF5 Category	Name	Data Type or HDF Category	Remarks / Data space
/(feature code)	attributes	See Table 10c-15	(see table)	Single-valued attributes as described in Table 10c-15

	Dataset (optional)	domainExtent	Compound (Float, Float)	Array (2-d): i=0, P-1 Values = <latitude, longitude> coordinates of polygon vertices (P is the number of polygons in the feature coverage extent.) Present if and only if the bounding box is not encoded in the attributes domainExtent: EX_GeographicExtent.EX_BoundingPolygon
	Dataset (optional)	uncertainty	Compound (String, Float)	Array (1-d): i = 0, (up to) numAttributes Code and uncertainty of data values. E.g., ("surfaceCurrentSpeed", 0.1) numAttributes is encoded in Group_F
/ (feat. code) /Group_TL (subgroup)				Tile information. Conditional, required if the product specification specifies tiling.
/ (feat. code) /Group_IDX (subgroup)				Spatial indexing method. Conditional, required if the product specification specifies spatial indexing. ~~(Described in product specifications.)~~
/ (feat. code) /~~Group_P~~Positioning (subgroup)				Positioning information. Coordinates of data values. Conditional, required if dataCodingFormat is not 2 (Regular grid)
/ (feat. code) /Group_nnn (subgroup)				

Table 10c-13~~14~~ - Structure of feature container group

Notes:

1) "uncertainty" is the uncertainty in data values, postion uncertainty (both horizontal and vertical) is encoded separately.

No.	Name	Camel Case	Mult.	Data Type	Remarks and/or Units
1	Data organization index, used to read the data (see Table 10c-4)	dataCodingFormat	1	Enumeration	1: Time series at fixed stations 2: Regularly-gridded arrays 3: Ungeorectified gridded arrays 4: Moving platform 5. Irregular grid 6. Variable cell size 7. TIN
2	Land mask value	gridLandMaskValue	1	Real	Fill value (e.g. -1.0 or -99.999). Also denotes a missing value.
6	Bounding box	boundingBox	0..1	Compound (Float X 4)	Components: westBoundLongitude eastboundLongitude southBoundLatitude northBoundLatitude Ref. domainExtent: EX_GeographicExtent > EX_GeographicBoundingBox
	Dimension	dimension	1	Integer	
29	Horizontal position uncertainty	horizontalPositionUncertainty	1	Real	-1.0 (unknown) or positive value (m)
30	Vertical position uncertainty	verticalUncertainty	1	Real	-1.0 (unknown/inapplicable) or positive value (m)

31	Time uncertainty	timeUncertainty	0..1	Real	-1.0 (unknown) or positive value (s) Only for time series data
dataCodingFormat = 1					
7	Valid Time of Earliest Value	dateTimeOfFirstRecord	1	Character	DateTime
8	Valid Time of Latest Value	dateTimeOfLastRecord	1	Character	DateTime
9	Time interval	timeRecordInterval	1	Integer	Seconds.
10	Number of time records	numberOfTimes	1	Integer	
13	Number of fixed stations	numberOfStations	1	Integer	
dataCodingFormat = 2					
17	Longitude of grid origin	gridOriginLongitude	1	Real	Arc Degrees (if dataCodingFormat=2)
18	Latitude of grid origin	gridOriginLatitude	1	Real	Arc Degrees (if dataCodingFormat=2)
19	Grid spacing, long.	gridSpacingLongitudinal	1	Real	Arc Degrees (if dataCodingFormat=2)
20	Grid spacing, lat.	gridSpacingLatitudinal	1	Real	Arc Degrees (if dataCodingFormat=2)
21	Number of points, long.	numPointsLongitudinal	1	Integer	iMax (if dataCodingFormat=2)
22	Number of points, lat.	numPointsLatitudinal	1	Integer	jMax (if dataCodingFormat=2)
23	First grid point num., long.	minGridPointLongitudinal	1	Integer	0 (if dataCodingFormat=2)
24	First grid point num., lat.	minGridPointLatitudinal	1	Integer	0 (if dataCodingFormat=2)
dataCodingFormat = 3					
25	Nodes in ungeorectified grid	numberOfNodes	1	Integer	Used if dataCodingFormat=3
Metadata specific to data product (all values of dataCodingFormat)					

Table 10c-1415 - Attributes of feature container group

10c-9.7 Tiling information group

This group encodes information about the tiling scheme used in the dataset. It is present if and only if the data is encoded in more than a single tile. Some tiling schemes are described in Part 8 (section 8-2). Note that tiling is not quite the same concept as "chunking," as the latter is defined in HDF5 and NetCDF – tiles divide the data into different datasets which are stored as distinct components of the HDF5 file, while chunking defines slices of a single storage component.

Group	HDF5 Category	Name	Data Type or HDF Category	Remarks / Data space
/Group_TL	Attribute	numTiles	Integer	Number of tiles value > 0
	Attribute	tilingScheme	Enumeration	1: Simple grid 2: Variable-density simple grid 3: Variable tile size 4: Polygonal tiles etc., etc. (specified in product specification)
	Dataset(s)	Attribute tileNumber	Integer	Sequence number of tile. Conditional - mandatory if more than one tile boundaries dataset is encoded.

		tiles-nnn (nnn: 001-999)	(Depends on scheme)	Data specifying the boundaries of tiles, e.g., bounding boxes, polygon vertices. Depends on the tiling scheme. (Specified in product specifications.)

Table 10c-~~15~~16 - Tiling information group

The details of tiling methods and the structure of tile datasets are left to product specifications in this edition of S-100.

10c-9.8 Indexes group

The indexes group encodes spatial indexing information, if used by the product specification. This group is encoded if and only if the product specification prescribes a spatial indexing method and requires explicit encoding of the spatial index.

Group	HDF5 Category	Name	Data Type or HDF Category	Remarks / Data space
/Group_IDX (subgroup)	Attribute	indexingMethod	Enumeration	Spatial indexing method. (Described in product specifications.)
	Dataset(s)	index-nnn (nnn: 000-999)	(Depends on indexing method)	Data encoding the spatial index. (Described in product specifications.)

Table 10c-~~16~~17 - Indexes group

The details of indexing methods and the structure of index datasets are left to product specifications in this edition of S-100.

10c-9.9 Positioning group

Commented [rmm14]: At this point of time (21 February) everything from this point on is WIP.

Depending of the data format, there can be an initial group of longitudes and latitudes, Group XY. This group contains no attributes but two datasets, X (longitude and Y (latitude). The number of values is *numPOS*. This group appears for values of *dataCodingFormat* of 1, 3, and 4 (Section 10c-9.3).

The traversal order for grids is specified by a carrier metadata attribute.

The dimensionality D of the data is given by the dimension metadata attribute in the feature container group.

Number of positions is computed as follows:

Type of data	numPOS computation
Time series, fixed station	
Point set	
2-D or 3-D regular grid	
Higher-dimension regular grid	
Irregular grid	
Variable-cell grid	

Positioning group, 2-D time series or fixed station data

Group	HDF5 Category	Name	Data Type	Data Space
/Group_XY	Dataset	Longitude	Float	Array (1-d): n=0, *numPOS-1*
	Dataset	Latitude	Float	Array (1-d): n=0, *numPOS-1*

Positioning group, 3-D time series or fixed station data

Group	HDF5 Category	Name	Data Type	Data Space
/Group_XYZ	Dataset	Longitude	Float	Array (1-d): n=0, *numPOS-1*
	Dataset	Latitude	Float	Array (1-d): n=0, *numPOS-1*
	Dataset	ZCoordinate	Float	Array (1-d): n=0, *numPOS-1*

Positioning group, other

Group	HDF5 Category	Name	Data Type	Data Space
/Group_P	Dataset	(axisNames[0])	Float	Array (1-d): n=0, *numPOS-1*
	Dataset	(axisNames[1])	Float	Array (1-d): n=0, *numPOS-1*
	Dataset	(axisNames[D])	Float	Array (1-d): n=0, *numPOS-1*

10c-9.10 Data values groups

The key idea at the core of the structure is this: the organization of the information is substantially the same for each of the various types of data, but the information itself will be interpreted differently.

~~The product format is designed to be flexible enough to apply for (a) time series data for one or more individual, fixed stations, (b) regularly-gridded data for multiple times, (c) ungeorectified gridded data for multiple times, and (d) moving platform (e.g., surface drifter) data with a constant time interval.~~ This approach contains, for each type, data in a similar format but which is interpreted differently. Since each type of data will be interpreted differently, the type of data must be identified by the variable dataCodingFormat, as shown in Table 10c-4.

Commented [E15]: I think this topic is important and the concept need to be worked out, either by this method or another. I am wondering why it need to be this flexible, as I think it will be harder to implement in systems which expect to utilize many data streams. Would not a standard field (and global) in metadata suffice?

For regularly gridded data, the value arrays are two dimensional, with dimensions numPointsLongitudinal and numPointsLatitudinal. By knowing the grid origin and the grid spacings, the position of every point in the grid can be computed by simple formulae.

However, for time series data, ungeorectified gridded data, and moving platform data (i.e., when dataCodingFormat is 1, 3 or 4), the location of each point must be specified individually. This is accomplished by the data in "Positioning", which gives the individual longitude and latitude for each location. For time series data, the longitude and latitude values are the positions of the stations; the number of stations is numberOfStations. For ungeorectified gridded data, the values are the positions of each point in the grid; the number of grid points is numberOfNodes. For moving platform data, values are the positions of the platform at each time; the number of platforms is numberOfStations.

Commented [E16]: Think this can benefit from a mapping table.

NOTE: If dataCodingFormat is 2, Group XY is not present.

The remaining Groups each contain a title, a date-time value, and the speed and direction arrays. The title can be used to identify each individual station with time-series data. For dataCodingFormat = 2 or 3, the date-time is for the entire grid. The data value arrays are two dimensional, with a number of columns (numCOL) and rows (numROW). For a time series, the speed and direction values will be for each time in the series. For a grid, the speed and direction values will be for each point in the grid.

The Groups are numbered 001, 002, etc., up to the maximum number of Groups, numGRP. For fixed station data, the number of Groups is the number of stations. For regular and ungeorectified grids, the number of Groups is the number of time records. For moving platform data, aside from Group XY, there is only one Group, corresponding to a single platform; additional platforms can be accommodated in additional data products.

10c-9.10.1 Time series data

Group	HDF5 Category	Name	Data Type	Data Space
/Group_001	Attribute	Title	String	

	Attribute	Date-Time	String	
	Dataset	Code of Attribute 1	(according to attribute type)	Array (2-d): i=0, *numCOL-1*, j=0,*numROW-1*
	Dataset	Code of Attribute 2		Array (2-d): i=0, *numCOL-1*, j=0,*numROW-1*
	Dataset	Code of Attribute N		Array (2-d): i=0, *numCOL-1*, j=0,*numROW-1*
/Group_002	Attribute	Title	String	
	Attribute	Date-Time	String	
	Dataset	Code of Attribute 1	(according to attribute type)	Array (2-d): i=0, *numCOL-1*, j=0,*numROW-1*
	Dataset	Code of Attribute 2		Array (2-d): i=0, *numCOL-1*, j=0,*numROW-1*
	Dataset	Code of Attribute N		Array (2-d): i=0, *numCOL-1*, j=0,*numROW-1*
/Group_999	Attribute	Title	String	
	Attribute	Date-Time	String	
	Dataset	Code of Attribute 1	(according to attribute type)	Array (2-d): i=0, *numCOL-1*, j=0,*numROW-1*
	Dataset:	Code of Attribute 1		Array (2-d): i=0, *numCOL-1*, j=0,*numROW-1*
	Dataset	Code of Attribute N		Array (2-d): i=0, *numCOL-1*, j=0,*numROW-1*

For each individual Group, there is one dataset for each attribute. Each dataset stores the values of the attribute as an array. The dimensions of the arrays depend on the spatial dimensions and data encoding format.

dimension	dataCodingFormat	Dimensions of values dataset array
2 or 3	1,3,4	1 X numCOL
2	2	numROW X numCOL
3	2	numROW X numCOL X numZ
2,3	5	

.

The number of individual Groups is given by the metadata variable, *numGRP*. The time interval between individual times is given by the metadata variable *timeRecordInterval*.

Values which represent different times are stored sequentially, from oldest to newest. The initial date value is contained in the Character format mimicking the DT format: *yyyymmddThhmmssZ*. By knowing the time interval (seconds) between each record, the time applicable to each value can be computed. In addition, the Groups, if they represent different times, are arranged sequentially, from oldest to newest.

~~10c-9~~10c-10 Support files

The HDF5 format does not encode support file information as feature attributes, i.e., application schema thematic attributes cannot be references to support files. The HDF5 “metadata” attribute of the root group is a reference to an external metadata file.

Mixed vector-coverage data products may continue to use support files in connection with vector feature classes and define vector feature or information classes with attributes that are references to support files, as usual.

Commented [E17]: Maybe a paragraph of how this fit with an exchange set would be useful to the reader.

~~10c-10~~10c-11 Prohibited HDF5 constructs

Constructs which cannot be processed using the standard libraries of the HDF5 release specified in this Part must not be used. This means specifically that HDF5 constructs which require the use of a library for a later release that specified in this Part must not be used.

Commented [E18]: Suggest to move this text up to the part that speaks about the HDF5 libraries.

~~10c-11~~10c-12 Constraints and validation

10c-12.1 Requirements

TBD

~~1. There is no mapping from Feature Catalog data definitions to most or all HDF-5 meta data elements. We do not want to code words from each specific product spec in order to interpret the data into S-100 constructs.~~

~~2. It should be possible to define an HDF-5 "profile" for S-100 products. HDF-5 is such a generic encoding standard that some profiling should be considered and imposed at the S-100 level.~~

~~3. There may be more fundamental issues at the spatial level when considering S-100 and HDF-5 grid definitions. At the very least, there should be clarification in part 10C regarding this, but an overall profile definition could also capture this.~~

~~4. There does not appear to be any constructs for information types within HDF-5 encoding to S-100~~

10c-12.2 Validation tests

Numerical attributes (i.e., float or integer data types) should have lower and upper bounds on values encoded in additional attribute characteristics, along with the closure type. The allowed list of closure types is listed in S-100 Edition 3.0.0 Part 1 Figure 1-4 (S100_IntervalType).