S-100 – Part 10c

HDF5 Data Model and File Format

Contents

10c-1 Scope ... 1
10c-2 Introduction ... 1
10c-3 Conformance ... 1
10c-4 Normative references ... 1
10c-5 HDF5 Specification ... 1
10c-5.1 Abstract Data Model ... 2
10c-5.1.1 File ... 3
10c-5.1.2 Group ... 3
10c-5.1.3 Dataset ... 4
10c-5.1.4 Dataspace ... 5
10c-5.1.5 DataType ... 5
10c-5.1.6 Attribute ... 6
10c-5.1.7 Property List ... 7
10c-5.2 HDF5 Library and Programming Model ... 7
10c-5.3 Prohibited HDF5 constructs ... 8
10c-6 S-100 profile of HDF5 ... 8
10c-7 Data types ... 8
10c-8 Naming conventions ... 9
10c-9 Structure of data product ... 10
10c-9.1 General structure ... 10
10c-9.2 Metadata ... 11
10c-9.2.1 Discovery metadata ... 11
10c-9.2.2 Carrier (embedded) metadata ... 11
10c-9.2.3 Extended metadata ... 12
10c-9.3 Generalized dimensions and storage of coordinates and data ... 12
10c-9.4 Root group ... 14
10c-9.5 Feature information group ... 17
10c-9.6 Feature container group and feature instance group ... 19
10c-9.7 Feature instance group ... 23
10c-9.8 Tiling information group ... 28
10c-9.9 Indexes group ... 29
10c-9.10 Positioning group ... 29
10c-9.10.1 Spatial representation strategy ... 30
10c-9.10.2 Data structures for storing position information for grid points ... 30
10c-9.11 Data values groups ... 31
10c-10 Support files ... 35
10c-11 Catalogue and metadata files ... 35
10c-12 Vector spatial objects, features, and information types ... 36
10c-13 Constraints and validation ... 36
10c-13.1 Validation tests ... 36
10c-14 Updates ... 36
10c-15 Summary of model ... 36
10c-16 Rules for product specification developers ... 37
10c-16.1 Defining the format for a product specification from this profile ... 37
10c-16.2 Miscellaneous rules ... 38
10c-16.3 Extensions of this profile ... 38
10c-16.4 Adding metadata ... 38
10c-17 Implementation guidance ... 38

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.

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.

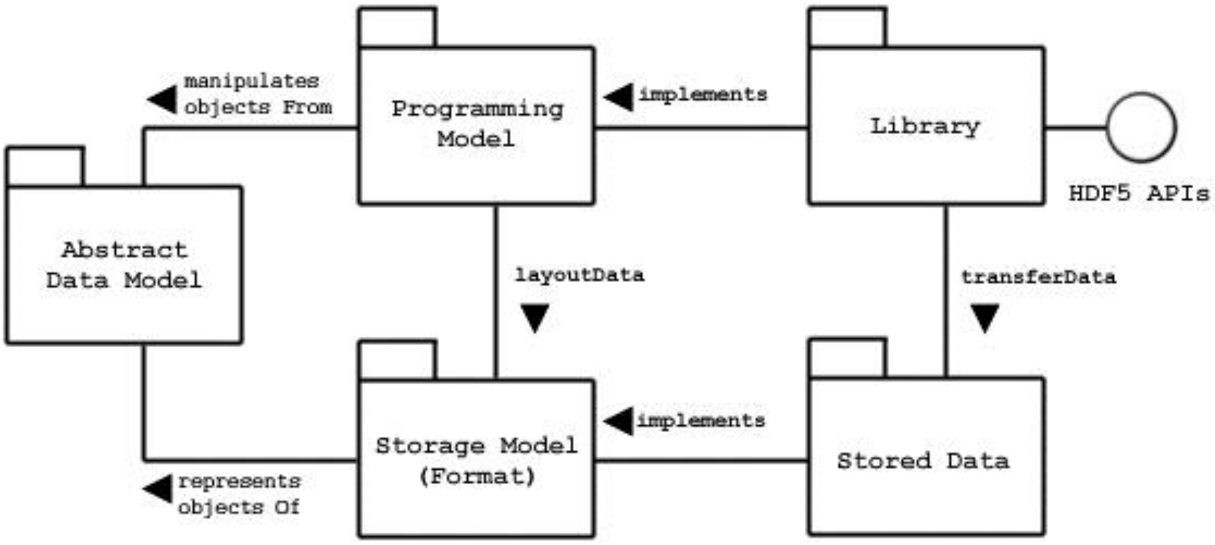

Figure 10c-1 - Abstract Data Model

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

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

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.

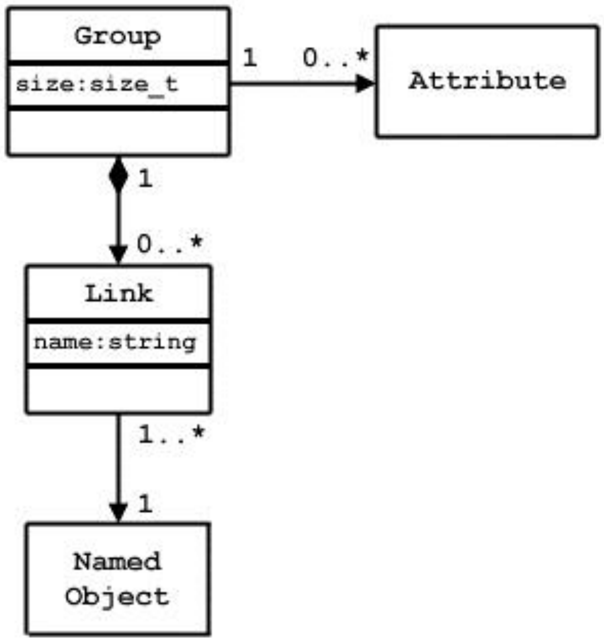

Figure 10c-2 - Group membership via link objects

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.

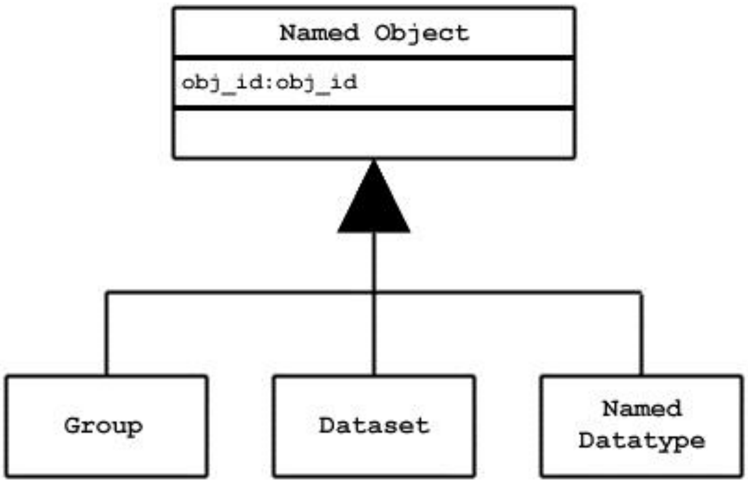

Figure 10c-3 - Classes of named objects

10c-5.1.3 Dataset

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.

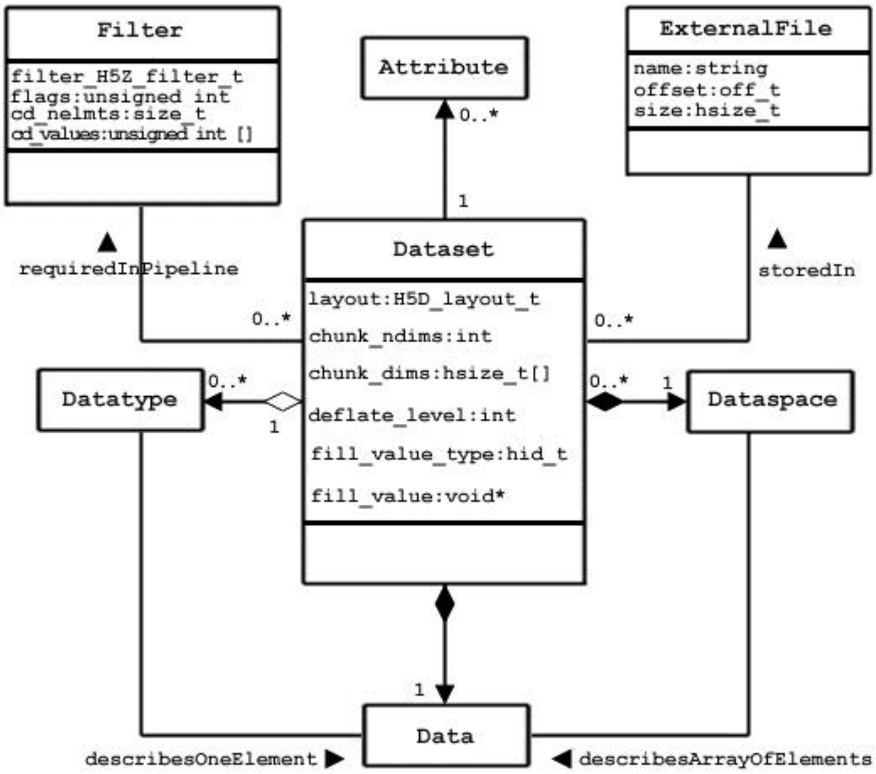

Figure 10c-4 - The dataset

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.

Not all the HDF5 datatypes have exact 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 this Part.

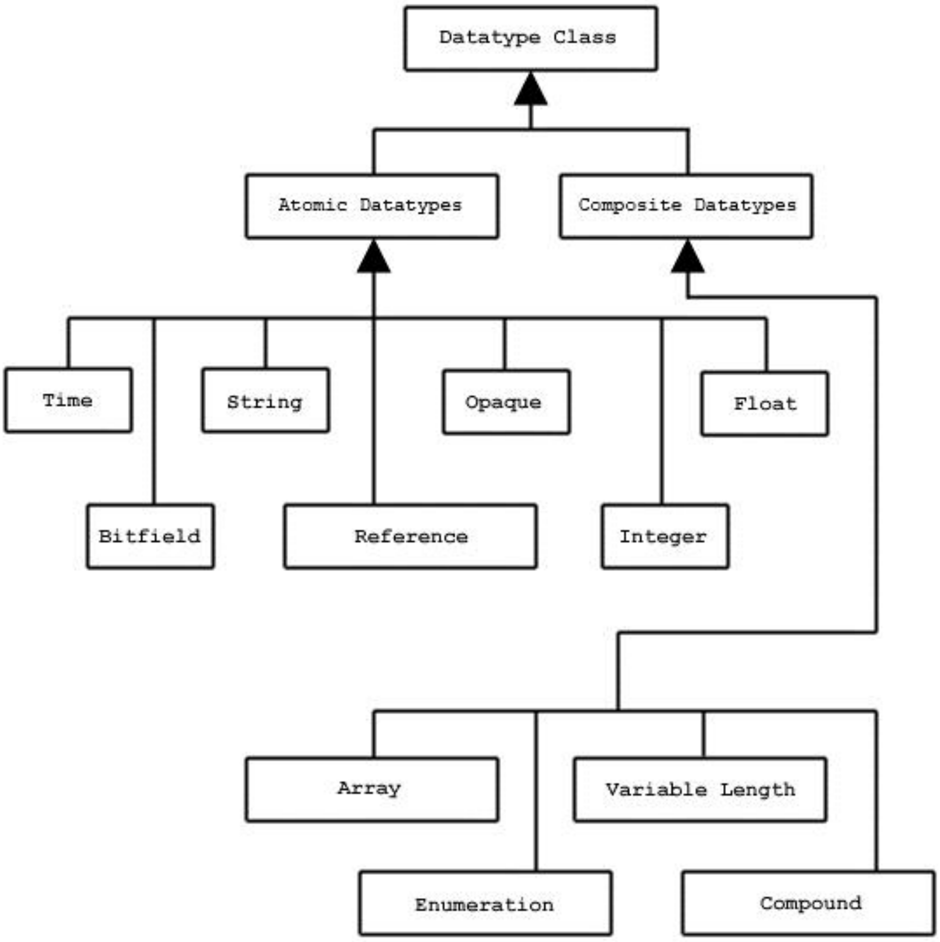

Figure 10c-5 - Datatype classifications

10c-5.1.6 Attribute

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.

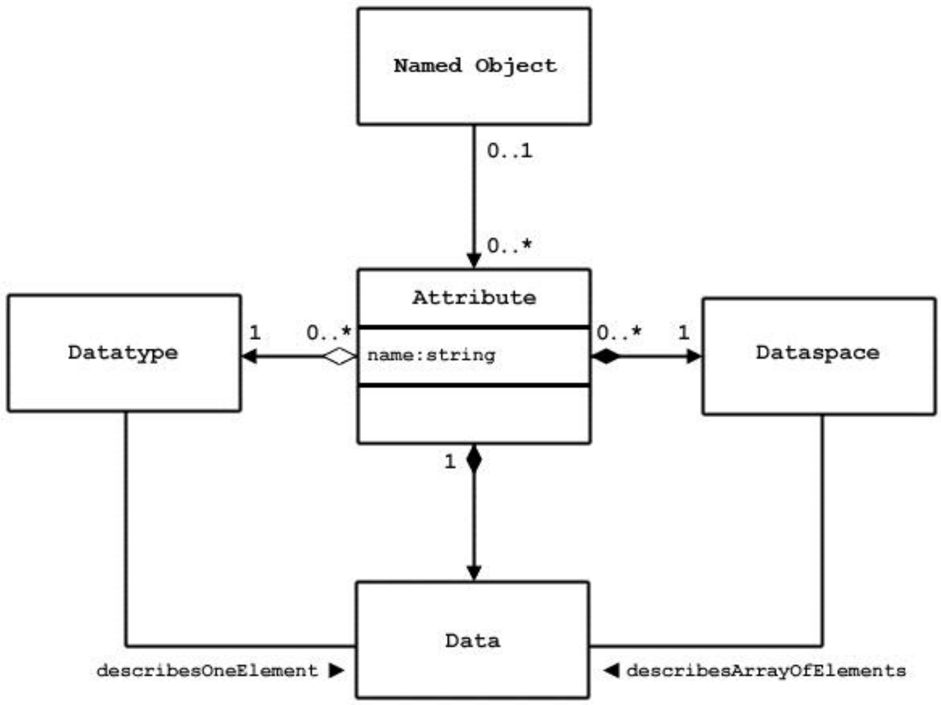

Figure 10c-6 - Attribute data elements

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

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 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

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.

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-5.3 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 than that specified in this Part must not be used.

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 (Part 8 clause 8-8.1.2),
- 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 mapped to equivalent HDF5 data types. These equivalences are summarized in Table 10c-1 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	32 or 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 must be according to value attribute types in the application schema. The “value record” corresponds to the value(s) record in Part 8 Figs. 8-21, 8-22, 8-23, 8-28, 8-29.

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

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 (since there is 1/1 mapping from the Application Schema to the Feature catalogue, this also amounts to requiring the same conformance to the Feature Catalogue). ‘Names’ used must be the camel case names. 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 which correspond to attributes in Parts 4a-4c must also conform to the corresponding camel case names in Part 4a-4c & 8.

Elements which do not have a direct correspondence 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 shorten fields which are deeply nested within the XML schemas).

The names ‘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 a schema element already named in S-100 or in the product specification must be given the same name as that element, using the camel-case code 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. (Product 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).

The following group names are reserved for the uses specified:

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). Specifications which require non-uniform positioning (e.g., second-order algebraic formulae) must be treated as ungeorectified grids.
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, 001-999.

Table 10c-2 – Reserved group names

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 following groups are contained within the root group. (The nesting levels in the list below correspond to the nesting levels in the HDF5 file).

1) Feature information group.
2) Feature container groups – each acts as a container for individual instances of a feature class. Its attributes encode any feature-class-level metadata.
    a) Feature instance groups – each acts as a container for the positioning positioning, tile, indexes, and data groups pertaining to a single feature instance. Its attribute encode any instance-level metadata
        i) Tiling information group (conditional, only if values are stored as tiles).
        ii) Indexes group (conditional, only if indexes to data are required).
        iii) Positioning group (conditional, only if postions are not computable from metadata).
        iv) Data values group(s). Only time series data will have more than one value group.

Note that the order in which groups and datasets are stored within the datafile may not be the same as the order in which they are created.

The basic structure of an S-100 HDF5 file is depicted in the figure below. ‘F’ is the number of feature classes defined in the product specification. It is not a requirement that every data file contain instances of all feature classes. There is one values group for each time point in the time series[1] (datasets which are not time series will have only a single values group in each feature instance group).

The FeatureContainer and Positioning groups are abstract classes because their attributes and content depend on the type of coverage.

A more detailed diagram is included later in this Part.

[1] Except for moving station data. The use of value groups for each coverage type is described later in this Part.

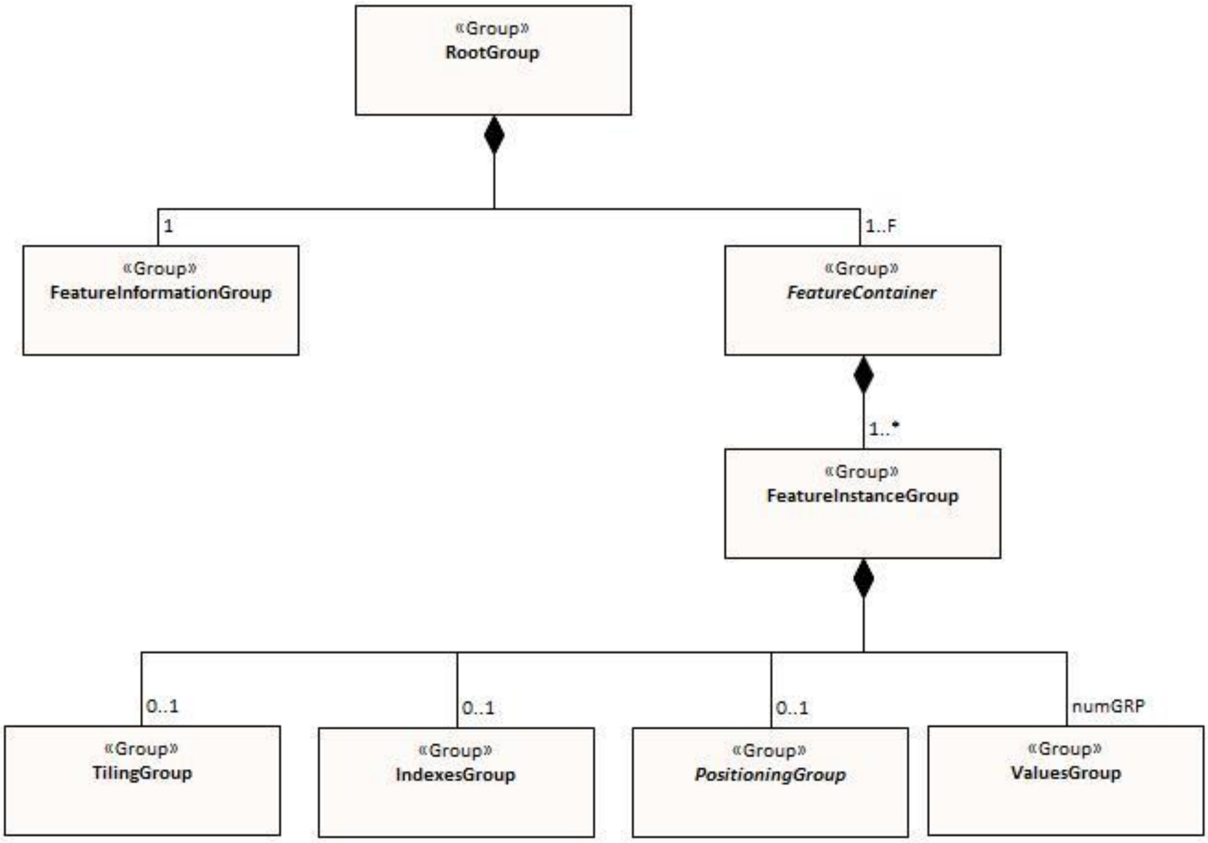

Figure 10c-7 - Basic structure of S-100 HDF5 file

10c-9.2 Metadata

Metadata is defined at different levels in the logical structure, so that metadata at the root group applies to all the features in the file, metadata at the feature container level applies to all instances of that feature class, and metadata at the instance level applies only to that particular feature instance.

10c-9.2.1 Discovery metadata

Full 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). See clause 10c-11 for naming conventions.

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 class. This metadata is used for feature and attribute specification information (corresponding to entries in the feature catalogue). This 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 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.

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. An example is provided later in this Part (Table 10c-7).
- 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 and storage of coordinates and data

This section provides an overview of the general approach to representing positioning information and storing data in S-100 HDF5 datasets. The basic approach is to minimize the variety of data structures used for storing data records. This profile stores data in one of two ways:
1) A multi-dimensional data array, of rank and dimensions corresponding exactly to the shape of the grid. This is used only for regular grids. In order to reduce space requirements, the coordinates of grid points are not explicitly stored because they can be computed from grid parameters.
2) One-dimensional arrays of data and grid coordinates, accompanied by meta-information describing the shape of the grid. This is also used for multipoint data (where there is no actual grid).

The key idea at the core of the structure is this: the organization of the data is logically the same for each of the various types of data, but the information itself will be interpreted differently depending on the type of spatial representation (which is indicated by an attribute).

For regularly-gridded data, the positioning information is not stored in the form of explicit coordinates because the grid metadata (extent and grid cell spacing information) suffices to specify the coordinates of each grid point. For example, for 2-D grids the value arrays are two dimensional, with dimensions specified by the attributes 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.

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 and moving platform data, the positioning information is stored in one-dimensional arrays of size numPOS of compound elements. The components of the compound element correspond to the coordinate axes, e.g., latitude, longitude, z-coordinate, time, etc. The sequence of points corresponds either to the positions of fixed stations or sequential positions of moving platforms, as appropriate.
- For ungeorectified grids, the positioning information is stored as one-dimensional arrays of size numPOS of compound elements (each as defined above).
- For irregular grids, the positioning information is stored as one-dimensional arrays of size numPOS of compound elements (as defined above). In addition, the tiling group may be populated with tiles whose spatial union exactly covers the grid. The sequence of coordinate elements must conform to the sequencingRule metadata attribute in the feature container group (10c-9.6). An optional tile index component (index into the tiles array – see 10c-9.7) may be

added to by a product specification for faster retrieval. If used, the tile index component must be named ‘tileIndex’ and be of ‘integer’ datatype.

- For grids with variable cell sizes, the positioning information is stored as one-dimensional arrays of size numPOS of compound elements (as defined earlier). The actual cell size is included as an attribute that describes the level of aggregation. 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 (using the same type of compound elements defned above) plus a Triangles array encoding references to the vertices of the triangle and references to adjacent triangles.

The storage of data and coordinate values is summarized in the table below. (‘D’ is the number of dimensions of the coverage.)

The datasets storing coordinates and values are designed so as to use uniform data storage structures across different coverage types as well as reduce the total data volume. These criteria resulted in storing the additional information needed by some coverage types separately (e.g., cell location and size information for irregular and variable cell size grids).

Coverage type	**Coordinate values**	**Data values**
Regular grid	Not explicitly stored Computable from metadata	D-dimensional array of value tuples
Irregular grid	Not explicitly stored Computable from metadata	1-d array of value tuples + information about location of cells
Variable cell size grid	Not explicitly stored Computable from metadata	1-d array of value tuples + information about cell size and location
Fixed stations, ungeorectified grid, moving platform	1-d array of coordinate tuples	1-d array of value tuples
TIN	1-d array of coordinate tuples + triangle information	1-d array of value tuples

Table 10c-3. Summary of storage strategies for coordinates and data values

Data Groups are separate groups containing the data values, which are stored in arrays corresponding to the positioning information. For coverage types where positioning information is not explicitly stored (N-dimensional regular grids), data is stored in N-dimensional arrays of rank corresponding to the grid dimensions (e.g. for 2-D data, 2-D arrays of size numROWS by numCOLS).

For time series data, multiple data groups are present. The total number of data Groups is numGRP. The meaning of numGRP for each type of spatial representation is specified in Table 10c-4. The format allows for time series data for all representations.

Positions in coordinate systems with more than 2 coordinate axes are encoded using correspondingly more dimensions. For example, for 3-dimensional data, the vertical dimension is used as a third dimension. This profile recommends limiting the number of dimensions to no more than four (space and time), but higher dimensionality may be used if required for the data product.

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

Coding Format	**Data Type**	**Positioning**	**Data Values**			**Times**
		numPOS	**numCOLS**	**numROWS**	**numZ (3-d only)**	**numGRP**
1	Fixed Stations	numberOfStations	numberOfStations	1	1	numberOfTimes

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	numberOfNodes	numberOfNodes	1	1	numberOfTimes
6	Variable cell size	numberOfNodes	numberOfNodes	1	1	numberOfTimes
7	TIN	numberOfNodes	numberOfNodes	1	1	numberOfTimes

Table 10c-4 – Array dimensions for different types of coverages

Note that numROWS, numCOLS, numZ, numPOS, and numGRP are not explicitly encoded in the HDF5 file. This specification uses them only to indicate array dimensions for implementation purposes. It is the number of stations, nodes, points, etc. that are encoded in as attributes of feature instances (section 10c-9.6.1).

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 logical product structure in HDF5 consists of (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 one or more feature instance groups, which in turn contain 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 physical layout of the file may not be the same as its logical data structure, however the HDF5 API allows implementers to access information using the logical data structure.

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-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.

<table>
<tr><th>Group</th><th>HDF5 Category</th><th colspan="2">Name</th><th>Data Type</th><th>Data Space / Remarks</th></tr>
<tr><td rowspan="7">/ (root)</td><td>Attributes</td><td colspan="2">(Carrier metadata attributes)</td><td>Integer, Float, Enumeration, or String</td><td>(none)
Described in Table 10c-6</td></tr>
<tr><td>Group</td><td colspan="2">Group_F</td><td></td><td>Feature information group (see section 10c-9.6)</td></tr>
<tr><td rowspan="5">Group(s)</td><td colspan="2">(featureCode)</td><td></td><td>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</td></tr>
<tr><td>HDF5 Category</td><td>Name</td><td></td><td></td></tr>
<tr><td>Group
(optional)</td><td>Group_TL</td><td></td><td>Tiling information, only if product uses tiles.
See section 10c-9.8</td></tr>
<tr><td>Group
(optional)</td><td>Group_IDX</td><td></td><td>Spatial index information, only if product uses spatial indexes
See section 10c-9.9</td></tr>
<tr><td>Group</td><td>Positioning</td><td></td><td>Positioning information – 2D or 3D.
Not required for dataEncodingFormat = 2 (Regular grid).
See section 10c-9.10</td></tr>
<tr><td></td><td></td><td>Group(s)</td><td>Group_NNN</td><td></td><td>Static data – only 1 values group
Time series data – 000 to 999 groups
See section 10c-9.11</td></tr>
</table>

Table 10c-5 - Root group

The common (core) metadata elements are specified as attributes of the root group, as 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 'INT.IHO.S-NNN.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	Time of data product issue	timeOfIssue	0..1	String (Time format)	Must be consistent with timeOfIssue in discovery metadata.

3	Issue date	issueDate	1	String (Date format)	Must be consistent with issueDate in discovery metadata.
4	Horizontal datum	horizontalDatumReference	1	String	EPSG
5	Horizontal datum number	horizontalDatumValue	1	Integer	4326 (for WGS84)
6	Bounding box	boundingBox	1	Compound (Float/Double 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 datum reference	verticalDatum	0..1	Enumeration	See S100_VerticalAndSoundingDatum Conditional, iff depthTypeIndex=3
10	Meta features	metaFeatures	0..1	String	Name of 8211 or GML file containing meta-features

Table 10c-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) The core attributes correspond to metadata attributes in S100_DatasetDiscoveryMetadata (Part 4a) or the imagery/gridded/coverage data attributes in Part 8. The correspondences are given in the Remarks column.
3) Vertical datum is optional 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. The additional attributes must be defined in the same way as Table 10c-6 – specifically, they must have a camel-case name beginning with a lower-case letter, multiplicity either 0..1 (optional) or 1 (mandatory) and be one of the allowed types listed in Table 10c-1. In addition, restrictions or additional conditions can be added for core carrier metadata attributes. The data types of common carrier metadata attributes cannot be changed, but the range of allowed values may be restricted or optional attributes made mandatory or conditionally mandatory.

EXAMPLE: The table below shows how S-111 might define an additional attribute (Vertical reference), introduce a conditional test for a core metadata attribute (Vertical datum reference), and make an optional metadata attribute mandatory (Time of data product issue).

No.	Name	Camel Case	Mult.	Data Type	Remarks and/or Units
Additional carrier metadata					
11	Vertical reference	depthTypeIndex	1	Enumeration	1: Layer average 2: Sea surface 3: Vertical datum (see verticalDatum) 4: Sea bottom
Additional restrictions or conditions on core carrier metadata					
2	Time of data product issue	timeOfIssue	1	String (Time format)	Mandatory in S-111
9	Vertical datum reference	verticalDatum	0..1	Enumeration	Required iff depthTypeIndex=3

Table 10c-7 – Example of extended metadata attribute and additional conditions on core metadata attributes

How the product specification describes core and extended metadata attributes is left to the specification writers, but specifications should distinguish core attributes from extended attributes as well as clearly indicating any additional restrictions or conditions on core attributes. The ISO format for specifying metadata extensions (Part 4a § 4a-5.6.5) may be used.

10c-9.5 Feature information group

The feature information group contains the specifications of feature classes and their attributes. The components of the feature information group are described in the table below.

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(s) (feature information datasets - one for each feature in the featureCode array)	<featureCode> e.g., SurfaceCurrent, WaterLevel	Attribute	Attribute name: chunking Type = string value = chunk dimensions (HDF5 chunk dimensions for data values of this feature, in string representation. See section 10c-5.3 and HDF5 documentation.)
			Array of Compound (String X 8)	Array (1-d): i=0, *NA_F-1 (NA_F = number of attributes of feature named by <featureCode>).* Components of the compound type: code: camel case code of attribute as in feature catalogue name: long name as in feature catalogue uom.name: units (uom>name from S-100 feature catalogue) fillValue: fill value (integer or float value, string representation) datatype: HDF5 data type, as returned by H5Tget_class() function lower: lower bound on value of attribute upper: upper bound on attribute value closure: type of closure The “code” and “datatype” components encode the rangeType attribute of the coverage features in Part 8. “lower”, “upper”, and “closure” encode any constraints on attribute values as encoded in the feature catalogue (see “S100_FC_SimpleAttribute>constraints” in Part 5 and S100_NumericRange in Part 1)

Table 10c-8 - Components of feature information group

All the numeric values in the feature description dataset are string representations of numeric values, 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 null values or the empty (0-length) string.

An entry in Group_F is required for every feature type that is used in the HDF5 data file. This means that:

- The **featureCode** array much include each feature type for which there is a feature instance somewhere in the current physical file.
- There must be a feature description dataset for each feature type named in the **featureCode** array.
- Each feature description dataset must list all the attributes of the feature type (both direct and inherited) as specified in the feature catalogue.

Note that the above requirements do not mandate entries in Group_F for feature types which are defined in the XML feature catalogue but for which there are no instances in the current data file.

The number of attributes for each feature type (NA_F in Table 10c-8) is not explicitly specified but can be determined using HDF5 API to determine the number of rows in each feature description dataset.

The figure below depicts Group_F for a hypothetical product with two feature types, *SurfaceCurrent* and *WaterLevel*. The two features are named (using the camel case codes from the feature catalogue) in the dataset **featureCode**. The feature description datasets **SurfaceCurrent** and **WaterLevel** describe the attributes of each feature type. The feature description datasets are given the same names as the values in the **featureCode** dataset, which are the camel case codes of the features from the XML feature catalogue. Each feature description dataset is an array of compound type elements, whose components are the 8 components specified in Table 10c-8. The chunk dimensions for the data itself are provides in the *chunking* attribute of each feature description dataset (shown in the two panels at the top right in the figure).

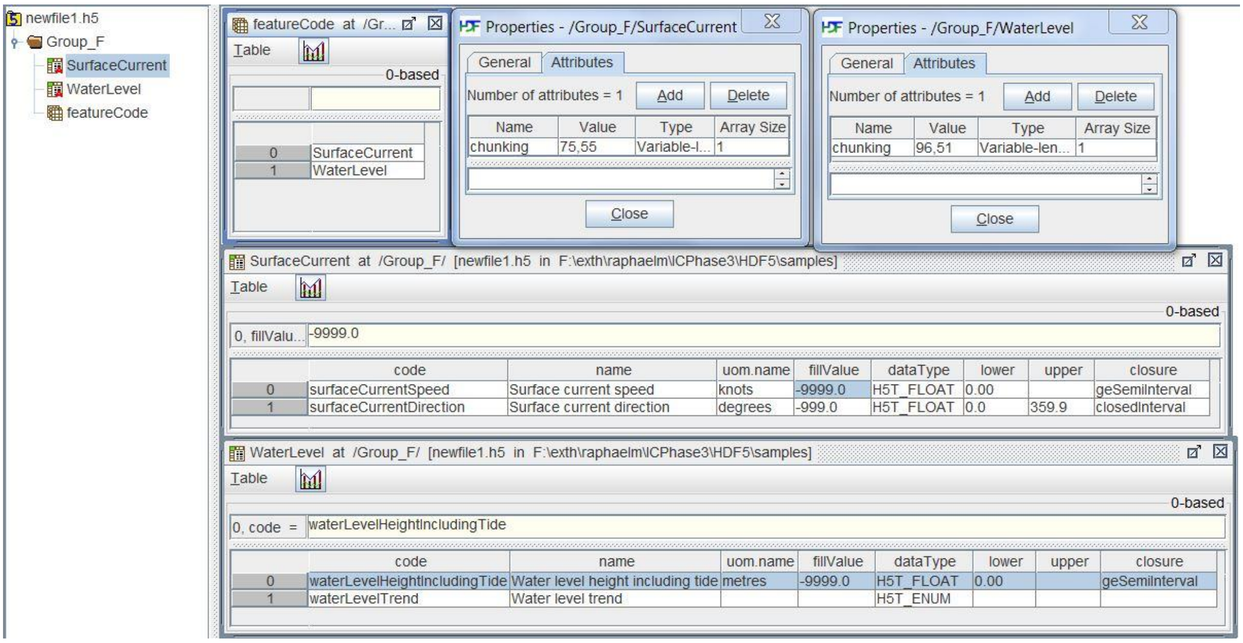

Figure 10c-8. Example of Group_F

10c-9.6 Feature container group and feature instance group

The feature container groups contain the coordinates and values for all instances of a single feature class. Each feature instance is allocated its own group within the feature container group. This organization allows class-wide attributes to be attached to the class as a whole and instance-specific attributes to be attached to the appropriate feature instance.

NOTE: The decision to make a distinct group for each feature instance is based on the fact that there will be multiple datasets for a single instance in some circumstances (e.g., index, TIN, etc.), and placing all the datasets directly under the container group is likely to add confusion to the data organization from the human perspective at least (though suffixes might suffice to distinguish different instances for programming purposes).

The structure of the Feature Container group is shown in Table 10c-9 below. This table also shows the feature instance group(s). The axis names are given in a dataset at the feature container level.

Metadata that is common to all instances of the feature class (such as dimensionality) is encoded at the feature container level and these metadata elements are listed in Table 10c-10. Metadata that is specific to feature instances (such as grid parameters) is encoded at the instance level and these elements are listed in Table 10c-12.

Product specifications may add product-specific metadata attributes at both the container and instance levels. The guidelines for additional metadata elements are the same as additional metadata elements in the root group (section 10c-9.4).

Group	HDF5 Category	Name	Data Type	Remarks / Data space
/(feature code)	attribute	See Table 10c-10	(see table)	Single-valued attributes as described in Table 10c-10
	Dataset	axisNames	String	Array (1-D): 0..D-1 where D is the value of the *dimension* attribute. axes should be in major-minor order, i.e., if storage is to be in row-major order the X/longitude axis should be first.
	Group	/(feature code).N		Container for each instance of a feature type. Numbered sequentially from 1 to *numInstances (Table 10c-10)*. Zero-padding with leading zeros must be used so that the 'N' suffixes are all the same length. To accommodate expansion, an extra zero is recommended.

Table 10c-9 - Structure of feature container groups

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
	Data organization index	dataCodingFormat	1	Enumeration	Indication of the type of coverage in instances of this feature. Used to read the data (see Table 10c-4). 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
	Dimension	dimension	1	Integer	The dimension of the feature instances.
	Common point rule	commonPointRule	1	Enumeration	The procedure used for evaluating the coverage at a position that falls on the boundary or in an area of overlap between geometric objects. Values from CV_CommonPointRule (ISO 19123)
	Horizontal position uncertainty	horizontalPositionUncertainty	1	Float	The uncertainty in horizontal coordinates. E.g., -1.0 (unknown) or positive value (m)
	Vertical position uncertainty	verticalUncertainty	1	Float	The uncertainty in vertical coordinate(s). E.g., -1.0 (unknown/inapplicable) or positive value (m)
	Time uncertainty	timeUncertainty	0..1	Float	Uncertainty in time values. E.g., -1.0 (unknown) or positive value (s) Only for time series data

	Number of feature instances	numInstances	1	Integer	Number of instances of the feature. (Records in the same time series or moving platform sequence are counted as a single instance, not as separate instances.)
	(additional common attributes)				(as specified in product specification)
dataCodingFormat = 1					
	(none)				
dataCodingFormat = 2					
	Sequencing rule	sequencingRule	1	Compound (Enumeration, String)	Method to be used to assign values from the sequence of values to the grid coordinates Components: type: Enumeration CV_SequenceType scanDirection: String <axisNames entry> (comma-separated). E.g., "latitude, longitude"
	Interpolation type	interpolationType	1	Enumeration	Interpolation method recommended for evaluation of the S100_GridCoverage Values: CV_InterpolationMethod (ISO 19123)
dataCodingFormat = 3					
	Interpolation type	interpolationType	1	Enumeration	Interpolation method recommended for evaluation of the S100_GridCoverage Values: CV_InterpolationMethod (ISO 19123)
dataCodingFormat = 4					
	(none)				
dataCodingFormat = 5					
	Sequencing rule	sequencingRule	1	Compound (Enumeration, String)	Method to be used to assign values from the sequence of values to the grid coordinates Components: type: Enumeration CV_SequenceType scanDirection: String <axisNames entry> (comma-separated). E.g., "latitude, longitude"
	Interpolation type	interpolationType	1	Enumeration	Interpolation method recommended for evaluation of the S100_GridCoverage Values: CV_InterpolationMethod (ISO 19123)
dataCodingFormat = 6					
	Sequencing rule	sequencingRule	1	Compound (Enumeration, String)	Method to be used to assign values from the sequence of values to the grid coordinates Components: type: Enumeration CV_SequenceType scanDirection: String <axisNames entry> (comma-separated). E.g., "latitude, longitude"
	Interpolation type	interpolationType	1	Enumeration	Interpolation method recommended for evaluation of the S100_GridCoverage Values: CV_InterpolationMethod (ISO 19123)

dataCodingFormat = 7					
	Interpolation type	interpolationType	1	Enumeration	Interpolation method recommended for evaluation of the S100_GridCoverage Values: CV_InterpolationMethod (ISO 19123)
(any dataCodingFormat value)					
	(additional attributes)				(as specified in product specification)

Table 10c-10 - Attributes of feature container groups

10c-9.7 Feature instance group

The feature instance groups are contained within the feature container groups. The structure of a feature instance group is defined in Table 10c-11. The attributes that are specific to each feature instance are defined in the table following (Table 10c-12) and consist of information that may vary for different instances in the same dataset, such as extent, location, time, and grid size.

Group	HDF5 Category	Name	Data Type	Remarks / Data space
/(feature code).N E.g. SurfaceCurrent.01	attributes	See Table 10c-12	(see table)	Single-valued attributes as described in Table 10c-12
	Dataset (optional)	domainExtent.polygon	Compound (Float, Float)	Spatial extent of the domain of the coverage. Array (1-d): i=0, P Components: <longitude, latitude> or <X, Y> (coordinates of bounding polygon vertices as a closed ring, i.e., the first and last elements will contain the same values.) Either this or the bounding box attribute must be populated. For irregular arrays, this dataset must specify the polygon indicating the area for which data are provided.
	Dataset (optional)	domainExtent.verticalElement	Compound (Integer X 2, Float X 2)	Array (1-d) of compound elements each providing a grid location and maximum, minimum vertical extents at the location. The components of the compound type are: gridX, gridY: Integer (grid point numbers along X/longitude and Y/latitude axes) minimumValue, maximumValue (Float): minimum and maximum Z values at the grid point specified by gridX and gridY Applicable only to 3-D grids. Either this dataset or the verticalExtent attribute (Table 10c-12) must be populated for 3-D grids.
	Dataset	extent	Compound (Integer X D)	1-D array, of compound elements, 2 rows. Row 0 gives the “low” values, row 1 the “high” values. The area of the grid for which data are provided. (Part 8 Fig. 8-23) Components of compound type are named according to the axis names in the axisNames dataset.

	Dataset (optional)	uncertainty	Compound (String, Float)	Array (1-d): i = 0, (up to) N_F Code and uncertainty of data values. E.g., ("surfaceCurrentSpeed", 0.1) numAttributes is encoded in Group_F
	Grid cell geometry (optional)	cellGeometry	Compound (String, Float X 2, Integer X 2)	Cell geometry. Array (1-d) of length the same as the *axisNames* array defined above (this means that if present, this dataset encodes all the axes including latitude, longitude, etc.). Conditional, required only for regular grids (dataEncodingFormat=2) using coordinate reference systems with axes other than (latitude, longitude, vertical), or with more than 3 dimensions. This array serves to extend the information encoded in the grid parameter attributes (origin, spacing, number of points, first point) defined in Table 10c-12 (Attributes of feature instance group) for data products which use higher-dimensional grids or non-standard coordinate axes. Components: axisName: string (an entry in the *axisNames* array defined above). gridOrigin: Float (the origin of the axis named in the axisName component) gridSpacing: Float (Cell spacing for the named axis) numPoints: Integer (the number of grid lines along the named axis) minGridPoint: Integer (first grid point for the named axis)
	Group (optional)	/Group_TL		Tile information. Conditional, required if the product specification specifies tiling.
	Group (optional)	/Group_IDX		Spatial indexing method. Conditional, required if the product specification specifies spatial indexing.
	Group (optional)	/Positioning		Positioning information. Coordinates of data values. Conditional, required if dataCodingFormat is not 2 (Regular grid)
	Group	/Group_nnn		Data Values group(s).

Table 10c-11 - Structure of feature instance groups

No.	Name	Camel Case	Mult.	Data Type	Remarks and/or Units
	Bounding box	boundingBox	0..1	Compound (Float X 4)	The geographic extent of the grid, as a bounding box. Components: westBoundLongitude eastboundLongitude southBoundLatitude northBoundLatitude Ref. domainExtent: EX_GeographicExtent > EX_GeographicBoundingBox Either this or the domainExtent dataset must be populated.
	Number of time records	numberOfTimes	0..1	Integer	The total number of time records. Time series data only.
	Time interval	timeRecordInterval	0..1	Integer	The interval between time records. Units: Seconds. Time series data only.
	Valid Time of Earliest Value	dateTimeOfFirstRecord	0..1	Character	The validity time of the earliest time record. Units: DateTime Time series data only
	Valid Time of Latest Value	dateTimeOfLastRecord	0..1	Character	The validity time of the latest time record. Units: DateTime Time series data only
	Vertical extent	verticalExtent	0..1	Compound (Float X 2)	Vertical extent of 3-D grids Components: minimumZ, maximumZ: Minimum and maximum values of the grid's spatial extent along the vertical direction
	Number of groups	numGRP	1	Integer	The number of data values groups contained in this instance group.
	(additional attributes specific to data product)	(as defined in product specification)			
dataCodingFormat = 1					
	Number of fixed stations	numberOfStations	1	Integer	The number of fixed stations
dataCodingFormat = 2					
	Longitude of grid origin	gridOriginLongitude	1	Float	The longitude of the grid origin. Unit: Arc Degrees
	Latitude of grid origin	gridOriginLatitude	1	Float	The longitude of the grid origin. Arc Degrees
	Vertical grid origin	gridOriginVertical	0..1	Float	The grid origin in the vertical dimension. Only for 3-D grids. Units specified by product specifications.

	Grid spacing, long.	gridSpacingLongitudinal	1	Float	Cell size in the X/longitude dimension. This is the X/longitudinal component of the offset vector (8-7.1.4). Units: Arc Degrees
	Grid spacing, lat.	gridSpacingLatitudinal	1	Float	Cell size in the Y/latitude dimension. This is the Y/latitudinal component of the offset vector (8-7.1.4). Units: Arc Degrees
	Grid spacing, Z	gridSpacingVertical	0..1	Float	Cell size in the vertical dimension. Only for 3-D grids. Units specified by product specifications.
	Number of points, long.	numPointsLongitudinal	1	Integer	Number of grid points in the X/longitude dimension. (iMax)
	Number of points, lat.	numPointsLatitudinal	1	Integer	Number of grid points in the Y/latitude dimension. (jMax)
	Number of points, vertical	numPointsVertical	0..1	Integer	Number of grid points in the vertical dimension. (kMax)
	First grid point num., long.	minGridPointLongitudinal	1	Integer	The first grid point in the X/longitude dimension. E.g.: 0
	First grid point num., lat.	minGridPointLatitudinal	1	Integer	The first grid point in the Y/latitude dimension. E.g.: 0
	First grid point number, vertical	minGridPointVertical	0..1	Integer	The first grid point in the vertical dimension. E.g.: 0
	Start sequence	startSequence	1	String	Grid coordinates of the grid point to which the first in the sequence of values is to be assigned. The choice of a valid point for the start sequence is determined by the sequencing rule. Format: n, n… (comma-separated list of grid points, one per dimension – e.g., 0,0)
dataCodingFormat = 3					
	Nodes in grid	numberOfNodes	1	Integer	The total number of grid points.
dataCodingFormat = 4					
	(none)				
dataCodingFormat = 5 or 6					
	Longitude of grid origin	gridOriginLongitude	1	Float	The longitude of the grid origin. Unit: Arc Degrees
	Latitude of grid origin	gridOriginLatitude	1	Float	The longitude of the grid origin. Arc Degrees
	Vertical grid origin	gridOriginVertical	0..1	Float	The grid origin in the vertical dimension. Only for 3-D grids. Units specified by product specifications.
	Grid spacing, long.	gridSpacingLongitudinal	1	Float	Cell size in the X/longitude dimension. This is the X/longitudinal component of the offset vector (8-7.1.4). Units: Arc Degrees For variable cell size grids this is the unit cell size (the size of the smallest cell in this dimension).
	Grid spacing, lat.	gridSpacingLatitudinal	1	Float	Cell size in the Y/latitude dimension. This is the Y/latitudinal component of the offset vector (8-7.1.4). Units: Arc Degrees For variable cell size grids this is the unit cell size.

	Grid spacing, Z	gridSpacingVertical	0..1	Float	Cell size in the vertical dimension. Only for 3-D grids. Units specified by product specifications. For variable cell size grids this is the unit cell size.
	Nodes in grid	numberOfNodes	1	Integer	The total number of grid points.
	First grid point num., long.	minGridPointLongitudinal	1	Integer	The first grid point in the X/longitude dimension. E.g.: 0
	First grid point num., lat.	minGridPointLatitudinal	1	Integer	The first grid point in the Y/latitude dimension. E.g.: 0
	First grid point number, vertical	minGridPointVertical	0..1	Integer	The first grid point in the vertical dimension. E.g.: 0
dataCodingFormat = 7					
	Nodes in grid	numberOfNodes	1	Integer	The total number of grid points.
	Triangles in grid	numberOfTriangles	1	Integer	The total number of triangles in the TIN.
(any dataCodingFormat value)					
	(additional attributes)				(as specified in product specification)

Table 10c-12 - Attributes of feature instance groups

NOTES:

1) The type-specific attributes for regular and variable cell size grids are the same except that the parameters giving the number of points in each dimension are replaced by the total number of nodes in the grid.
2) Attributes “Valid time of earliest value” and “Valid time of latest value” provide the *temporalElement* component of the domainExtent attribute in the grid model (Figures 8-21, 8-22, 8-28, 8-29).

The figure below depicts the structure of a hypothetical data file containing 3 instances of the **SurfaceCurrent** feature type.

- The vertical panel on the left shows the overall structure. The data product consists of 2 features (**SurfaceCurrent** and **WaterLevel**). Each is represented by a group just under the root group. The Feature Information group described earlier (section 10c-9.5) is also shown.
- The Feature Container group named **SurfaceCurrent** contains 3 instances of the **SurfaceCurrent** feature type (hypothetically, data for 3 separate places, each with a local coverage grid). Each instance contains subgroups (Group_001, etc.) for time series data.
- Locations are encoded in the geometryValues dataset in the **Positioning** group (panel at top right). The **axisNames** panel to its left names the components of the **geometryValues** (i.e., the coordinate axes).
- The **SurfaceCurrent** panel in the the middle shows the metadata attributes common to all instances, which are attached to the **SurfaceCurrent** feature container group.
- The two panels at the bottom show the instance-specific metadata for the feature instances **SurfaceCurrent.01** and **SurfaceCurrent.02**.

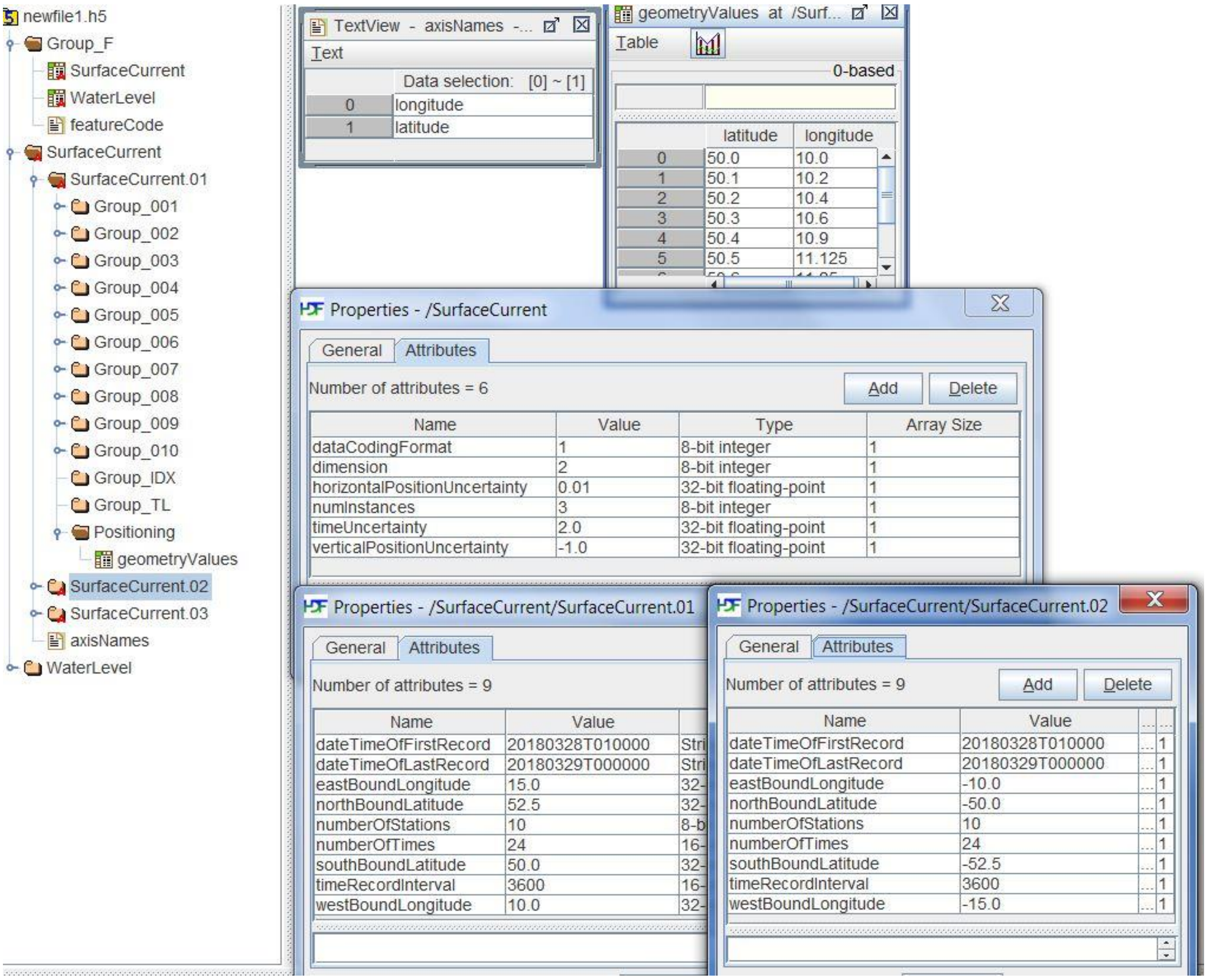

Figure 10c-9 - Illustrative example of dataset[2] structure.

10c-9.8 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). This edition of the HDF5 profile allows only two tilings: simple grid and variable density simple grid. In both cases, the extents of the tiles are specified in terms of their bounding boxes (Table 10c-12).

[2] The components of compound types among the Attributes are created as separate attributes because of software tool limitations.

The spatial union of tile surfaces must cover all the features in the dataset, but the converse is not a requirement. (Informally, this means that there may be parts of tiles that are not covered by the geometry of any feature in the dataset, but not vice versa – there cannot be parts of feature geometry that are not covered by at least one tile.)

Note that tiling is not quite the same concept as “chunking,” as the latter is defined in HDF5 and NetCDF – tiles are coordinate-based geographical partitions, while chunking defines slices of datasets for storage and retrieval performance optimization.

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 (product specification must pick one)
	Dataset	tiles	Array Compound (Float X 4, Integer)	Bounding boxes of tiles. Components: westBoundLongitude: Float eastboundLongitude: Float southBoundLatitude: Float northBoundLatitude: Float tileID: Integer (tile identifier)

Table 10c-13 - Tiling information group

The details of tiling methods are left to product specifications in this edition of S-100. This profile does not specify an ordering for the tiles, nor does it control the use or non-use of hierarchical tiling schemes. Part 8 (8-7.1) requires that any tiling scheme used must be completely described as part of the product specification for a particular data product. This includes the dimensions, location and data density of tiles as well as a tile identification mechanism (tileID).

10c-9.9 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	Attribute	indexingMethod	Enumeration	Spatial indexing method. (Described in product specifications.)
	Dataset(s)	spatialIndex	(Depends on indexing method)	Data encoding the spatial index. (Described in product specifications.)

Table 10c-14 - 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-9.10 Positioning group

Depending of the data format, there can be a positioning group, Positioning. This group contains no attributes, it contains a coordinates dataset, which is an array of compound type with components named the same as the *axisNames* dataset in the Feature Container group. This group is used for values of *dataCodingFormat* of 1, 3, 4, 7 (Section 10c-9.3). It is not used for *dataCodingFormat* = 2 (regular grids), 5 (irregular grid), or 6 (variable cell size grid).

The traversal order for grids of different types is specified by the carrier metadata attribute *sequencingRule* in the feature container group. Traversal order is not used for fixed station, moving platform, or TIN data (*dataCodingFormat* = 1, 4, or 7).

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

10c-9.10.1 Spatial representation strategy

For regularly gridded data (dataCodingFormat = 2), the number of grid points in each dimension, grid spacing, and grid origin are encoded in metadata attributes. (E.g., for 2-D grids, the metadata attributes *numPointsLongitudinal* and *numPointsLatitudinal* encode the points along the longitude and latitude axes.) Given these parameters and the indexes of a point in the grid, the position of the point can be computed by simple formulae.

For fixed station time series data, ungeorectified gridded data, moving platform data, and triangulated irregular networks (i.e., when dataCodingFormat is 1, 3, 4, or 7), the location of each point must be specified individually. This is accomplished in a dataset in the "Positioning" group, which gives the individual location coordinates (e.g., longitude and latitude) for each location. For fixed station 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*.

For irregular grid and variable cell size coverages (dataCodingFormat 5 and 6), the storage format uses the same metadata as for regular grids plus datasets indicating which cells are populated or aggregated respectively. The latter datasets encode the locations of cells in terms of grid point or cell address in grid coordinates – i.e., the indexes in the grid, or the Morton code – not the geographic (latitude/longitude) coordinates. The sequencing and axis order needed for interpretation of the grid coordinates as geographic coordinates are given by the *sequencingRule* and *scanDirection* attributes respectively. By combining this information with the grid parameters provided in metadata, the position of populated cells/points can be computed with slightly more complex formulae than for regularly gridded data.

The table below summarizes the strategies for storage of coordinate information.

Type of coverage	**dataCoding Format**	**Structure of coordinates dataset**
Fixed Stations	1	1-dimensional Array, length = numberOfStations
Regular Grid	2	not used
Ungeorectified Grid	3	1-dimensional Array, length = numberOfNodes
Moving Platform	4	1-dimensional Array, length = numberOfTimes
Irregular Grid	5	not used
Variable cell size	6	not used
TIN	7	1-dimensional Array, length = numberOfNodes

Table 10c-15 – Positioning dataset types and dimensions for different coverage types

NOTE: Multiple moving platforms can be encoded as different feature instances.

10c-9.10.2 Data structures for storing position information for grid points

The number of positions is computed as specified in Table 10c-4 in Section 10c-9.3.

Group	**HDF5 Category**	**Name**	**Data Type**	**Data Space**

/Positioning	Dataset	geometryValues	Compound (Float X D)	Array (1-dimensional) of size dependent on dataEncodingFormat, see Table 10c-15. Components of compound type are named according to the axis names (e.g., 'latitude', 'longitude', 'Z', etc.) The dimension D and the component names are specified in the feature container group *dimension* attribute and *axisNames* dataset respectively (Tables 10c-10 and 10c-9).
	Dataset	triangles (optional)	Array (Integer)	Array (2-d): dimensions numberOfTriangles X 3 Each row encodes a triangle as the indexes of 3 coordinates in the *geometryValues* dataset Required only for dataEncodingFormat = 7 (TIN)
	Dataset	adjacency (optional)	Array (Integer)	Array (2-d): dimensions numberOfTriangles X 3 Each row encodes the triangles adjacent to any given triangle by specifying their indexes in the triangles dataset. adjacency[i][0] = triangle adjacent to the edge specified by triangles[i][0] & triangles[i][1] adjacency[i][1] = triangle adjacent to edge triangles[i][1] & triangles[i][2] adjacency[i][2] = triangle adjacent to edge triangles[i][2] & triangles[i][0] Elements for edges without adjacent triangles are filled with the value -1. Applicable only for dataEncodingFormat = 7 (TIN), but optional even for TIN.

Table 10c-16 - Positioning group

10c-9.11 Data values groups

The structure of data values content is analogous to that of positioning content, except that regular grid data values (dataEncodingFormat = 2) are stored as a D-dimensional array corresponding to the axis order in the *axisNames* dataset in the Feature Container group (major index precedes minor index). The dimensionality D is encoded in the *dimension* attribute of the Feature Container group.

EXAMPLE: For two-dimensional regularly gridded data, the value arrays are two dimensional, with dimensions numPointsLongitudinal and numPointsLatitudinal.

For fixed station time series data, ungeorectified gridded data, moving platform data, and triangulated irregular networks (i.e., when dataCodingFormat is 1, 3, 4, or 7), the data values are stored as 1-dimensional datasets of length given by the numberOfNodes or numberOfStations metadata attribute of the feature instance group (Table 10c-12) depending on the dataEncodingFormat.

For irregular grid coverages (dataCodingFormat=5), the storage of data values is the same as for ungeorectified grids etc. (i.e., a 1-dimensional array of value records, length = numberOfNodes) but the value group includes a dataset that specifies the grid point or cell address associated to each entry in the values array. This second dataset uses grid coordinates - i.e., the indexes in the grid, or the Morton code – not the geographic (latitude/longitude) coordinates. The sequencing and axis order needed for interpretation of the grid coordinates as geographic coordinates are given by the *sequencingRule* and *scanDirection* attributes respectively.

For variable cell size coverages (dataCodingFormat=6) the storage of data values is the same as for irregular grid coverages but the values groups contains the grid address dataset used by irregular grids as well as a dataset indicating which cells are aggregated into larger cells.

The various datasets and their components are described in the following table.

Type of coverage	dataCoding Format	Structure of values and auxiliary datasets	Dataset components
Fixed Stations	1	values: 1-dimensional Array, length = numberOfStations	Compound, one component for each attribute specified in the corresponding feature information dataset in the Feature Information group (Table 10c-8). Component name: attribute code as specified in the feature information dataset Component type: Any appropriate HDF5 datatype consistent with the attribute datatype specified in the Feature Information dataset
Regular Grid	2	values: D-dimensional array, dimensions specified by: 2-D: numPointsLatitudinal X numPointsLongitudinal 3-D: numPointsLatitudinal X numPointsLongitudinal X numPointsVertical If *cellGeometry* is present in feature instance group: product of all cellGeometry[i].numPoints values.	As for fixed stations
Ungeorectified Grid	3	values: 1-dimensional Array, length = numberOfNodes	As for fixed stations
Moving Platform	4	values: 1-dimensional Array, length = numberOfTimes	As for fixed stations
Irregular Grid	5	values: 1-dimensional Array, length = numberOfNodes	As for fixed stations. Ordered according to the sequence rule specified by the *sequencingRule* and *scanDirection* attributes of the Feature Container group (Table 10c-10).
		gridIndex: 1-dimensional Array, length = numberOfNodes (dataset attribute codeSize: Integer - gives the length of the bitfield)	Element type: bitfield (length determined by grid dimensions) Order of element corresponds to the values array. Each element contains the code of the cell (grid point) according to the sequence rule specified by the *sequencingRule* and *scanDirection* attributes. E.g., the Morton code of the cell
Variable cell size	6	values: 1-dimensional Array, length = numberOfNodes	As for fixed stations
		gridIndex: 1-dimensional Array, length = numberOfNodes (dataset attribute codeSize: Integer - gives the length of the bitfield)	(As for the *gridIndex* Array for irregular grids) For cells that aggregate multiple unit cells, use the first cell (grid point) encountered in the sequencing order. E.g., the Morton code of the cell
		cellScale: 1-dimensional Array, length = numberOfNodes	Element type: Compound. Order of elements corresponds to the values array. Components of the compound type are named according to the axis names in the axisNames dataset in the Feature Container group.

Type of coverage	dataCoding Format	Structure of values and auxiliary datasets	Dataset components
			Each component is of type Integer and gives the number of cells aggregated along the named axis.
TIN	7	1-dimensional Array, length = numberOfNodes	(As for fixed stations)

Table 10c-17 - Values dataset type and size for different data encoding formats

NOTES:
1) 64-bit unsigned integers for gridIndex arrays allow 4-D grids with a maximum of 2^{16} - 1 (65,535) points/cells in each dimension.
2) The *gridIndex* datasets have an integer attribute named *codeSize* that gives the length (in bits) of the bitfield that contains the index. This depends on the type of code and the number of dimensions. For example, a 2-D grid with 8 points in each dimension needs 6-bit Morton codes.
3) The size of the bitfield is calculated by multiplying the number of bits needed to accommodate the largest dimension by the number of dimensions (D). To reduce complexity each dimension is allocated the same number of bits in the bitfield. For example, a 200 X 1000 array is given a 20-bit bitfield, calculated as:
   codeSize = 2 × maximum (ceiling($\log_2$ 200), ceiling($\log_2$ 1000)) = 20.

The figure that follows depicts *gridIndex* and *cellScale* arrays for an irregular grid (left) and variable cell size array (right). Both use Morton codes and 2-D grids of (nominally) 4×4 cells in each dimension. Note that in the figure it is the cells rather than grid points that are assigned codes. The panels on the left describe an irregular grid with 11 populated cells. The panels on the right describe a variable cell size grid with two aggregate cells, each aggregating 2×2 unit cells.

The grids themselves are depicted below the panels, with the Morton codes shown in the respective cells[3]. The example on the right also indicates the scaling of each cell in parentheses (it is assumed that the scaling is the same in all dimensions, i.e., cells 0100 and 1000 each aggregate 2×2 regions of the grid).

For the irregular grid example, the missing cells are not shown in the grid. For the variable cell size example, the greyed cells are aggregated with cells 0100 or 1000.

For variable cell size grids, this profile specifies the size of aggregated cells in terms of the number of unit cells they cover in each direction, instead of applying the same zoom factor in each dimension as depicted in the example at the bottom right of the figure. This is for the better accommodation of rectangular and odd-shaped aggregations. Odd-shaped regions must be split into multiple rectangular aggregations. (Using rectangular aggregations has an associated extra storage cost.)

Further optimizations may be addressed in future editions of this profile.

[3] The two grid depictions at the bottom of the figure are from “Elevation Surface Model Standardized Profile” (DGIWG 116-1) Ed. 1.0.1, Defence Geospatial Information Working Group (10 June 2014).

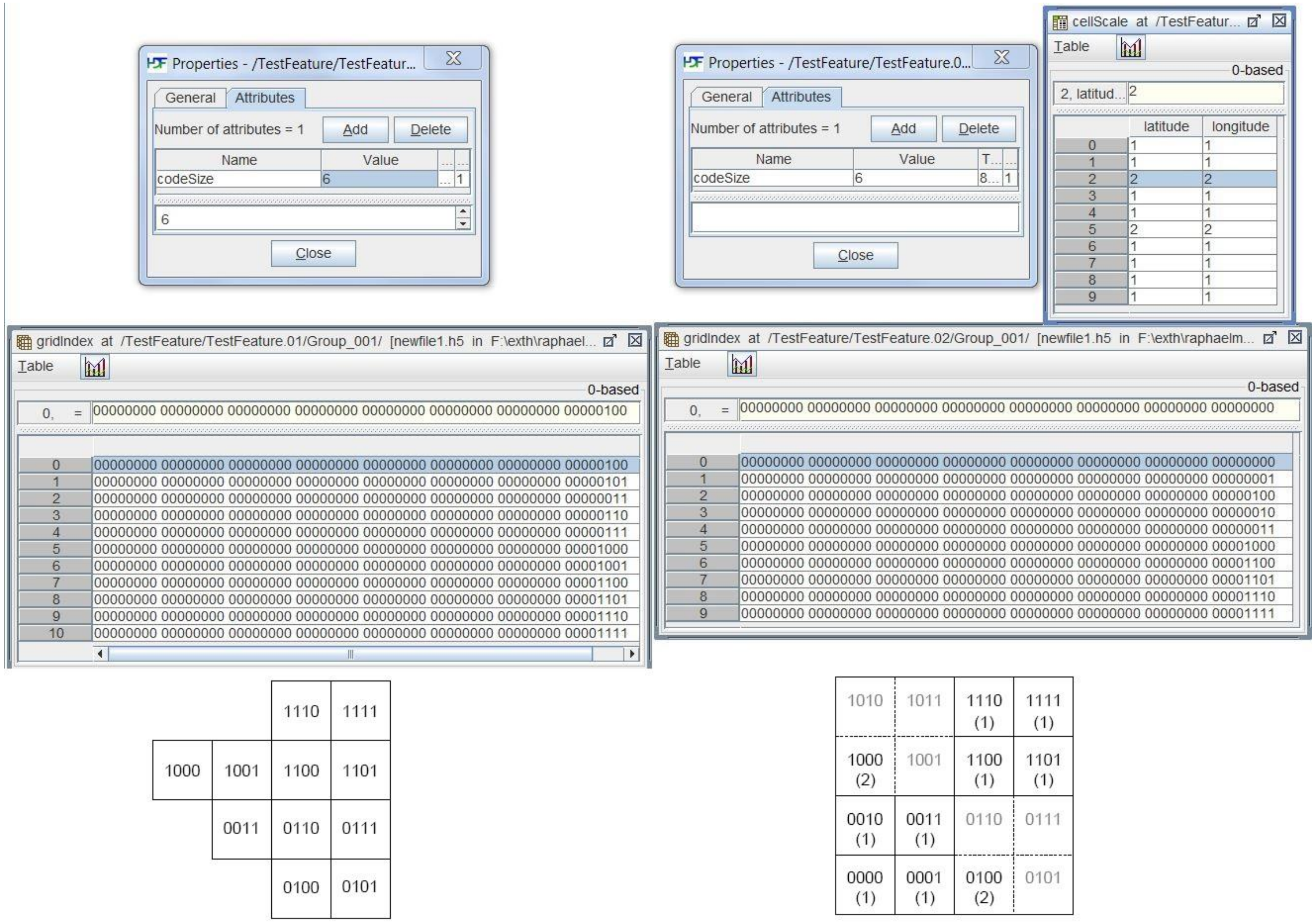

Figure 10c-10 - Illustrative examples of grid index array for irregular grids (left) and grid index and cell scale arrays for variable cell size grids (right).

The structure of the data values groups can now be described. Each group is structured as depicted in the table below.

Group	HDF5 Category	Name	Data Type	Data Space
/Group_NNN	Attribute	timePoint (optional)	String (date-time format)	Time point for time series data. For other types of data, it can be used to indicate the time for the whole grid.
	Dataset	values	Compound	1-dimensional Array of Compound type, described in Table 10c-17.
	Dataset	gridIndex	Bitfield	Required for dataEncodingFormat = 5 or 6. Described in Table 10c-17.
	Dataset	cellScale	Compound	Required for dataEncodingFormat = 6. Described in Table 10c-17.

Table 10c-18 - Structure of value group

Time series data for all except the moving platforms format (dataEncodingFormat = 4) are encoded in successive groups contained within the instance group.

The sub-Groups each contain a date-time value, and the value record arrays. For dataCodingFormat = 2, 3, 5, or 6, the date-time is for the entire grid. The data value arrays are two dimensional, with a number of columns (numCOLS) and rows (numROWS). For a time series, the data 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 all coverage types except moving platforms, the number of Groups is the number of time records. For moving platform data, there is only one Group, corresponding to a single platform; additional platforms can be accommodated in additional feature instances.

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-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. This means that references to pictures or text files, etc., are not permitted in coverage features.

Also, feature and information associations from coverage to vector features are not permitted.

The HDF5 “metadata” attribute of the root group is a reference to an external metadata file. The reference must be a string of the form:

fileRef:<fileName>

where <fileName> is the base name of the ISO 8211 or GML file. The extension part of the file name is not used.

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.

10c-11 Catalogue and metadata files

Exchange set catalogues and metadata files must conform to the standard XML schemas for catalogues and metadata defined for this edition of S-100 and the relevant ISO standards. The files must be named as follows:

CATALOG.NNN.XML (or .xml) Exchange catalogue XML file. NNN is the product specification number.

MD_<HDF5 data file base name>_.XML (or .xml) ISO metadata

10c-12 Vector spatial objects, features, and information types

In some circumstances it may be necessary to use vector spatial objects, such as area of influence polygons. This edition of the profile does not encode vector spatial objects directly in the HDF5 data file. Instead, the spatial objects should be defined in an external file (either GML or ISO 8211 format) and a reference to the spatial object encoded. The reference must be a string of the form:

extObjRef:<fileName>:<recordIdentifier>

where <fileName> is the base name of the ISO 8211 or GML file, and <recordIdentifier> is the record identifier of the vector object record within that file. The extension part of the file name is not used. The record identifier is the gml:id for GML datasets, or the record identification number (RCID) for ISO 8211 datasets. The file must be present in the same exchange set.

This method can be used to reference polygons, etc., defined in external files in GML or 8211 format data files in the same exchange set. It can also be used to reference feature or information type instances in the GML or ISO 8211 file.

EXAMPLES:

USSFC00001:S093546 references the object with gml:id S093456 in the GML data file USSFC00001.GML (GML).

USSFC00001:93546 references the object with record identifier 93456 in the ISO 8211 data file USSFC0000.000 (ISO 8211).

10c-13 Constraints and validation

10c-13.1 Validation tests

Validation tests must be defined in the product specification, and include checks that:

HDF5 file structure conforms to this profile.

Mandatory attributes in the groups are present according to the encoded value of dataCodingFormat.

Group, dataset, and attribute names conform to this profile.

Lengths of postioning and value records arrays are consistent.

Components of compound types are named as required by the specification.

10c-14 Updates

Updates to HDF5 datafiles are recommended to follow the same structure as the base HDF5 datafile. Updates may include only the HDF5 datasets which are being updated. The specific datasets being updated are included in their entirety in the update datafile.

This clause implies that datasets may be updated in part as well as replaced completely by updated data, but product specifications are not required to permit partial updates. They may defne update creation and management processes which are more suitable for their particular domains and applications. However, if updates to parts of datasets are allowed, the rule in the previous paragraph must be followed.

10c-15 Summary of model

The basic structure of the HDF5 profile (Figure 10c-7) can now be presented as a more detailed conceptual model using the group and dataset specifications in the previous sections. The conceptual model of HDF5 file contents is shown in the following figure. This figure shows the group structure and the datasets which contain spatial representations and data values. (Metadata attributes and datasets containing metadata are not included for the sake of simplicity.) The *MatchingOrders* association indicates that the sequences of elements in the associated datasets are interdependent.

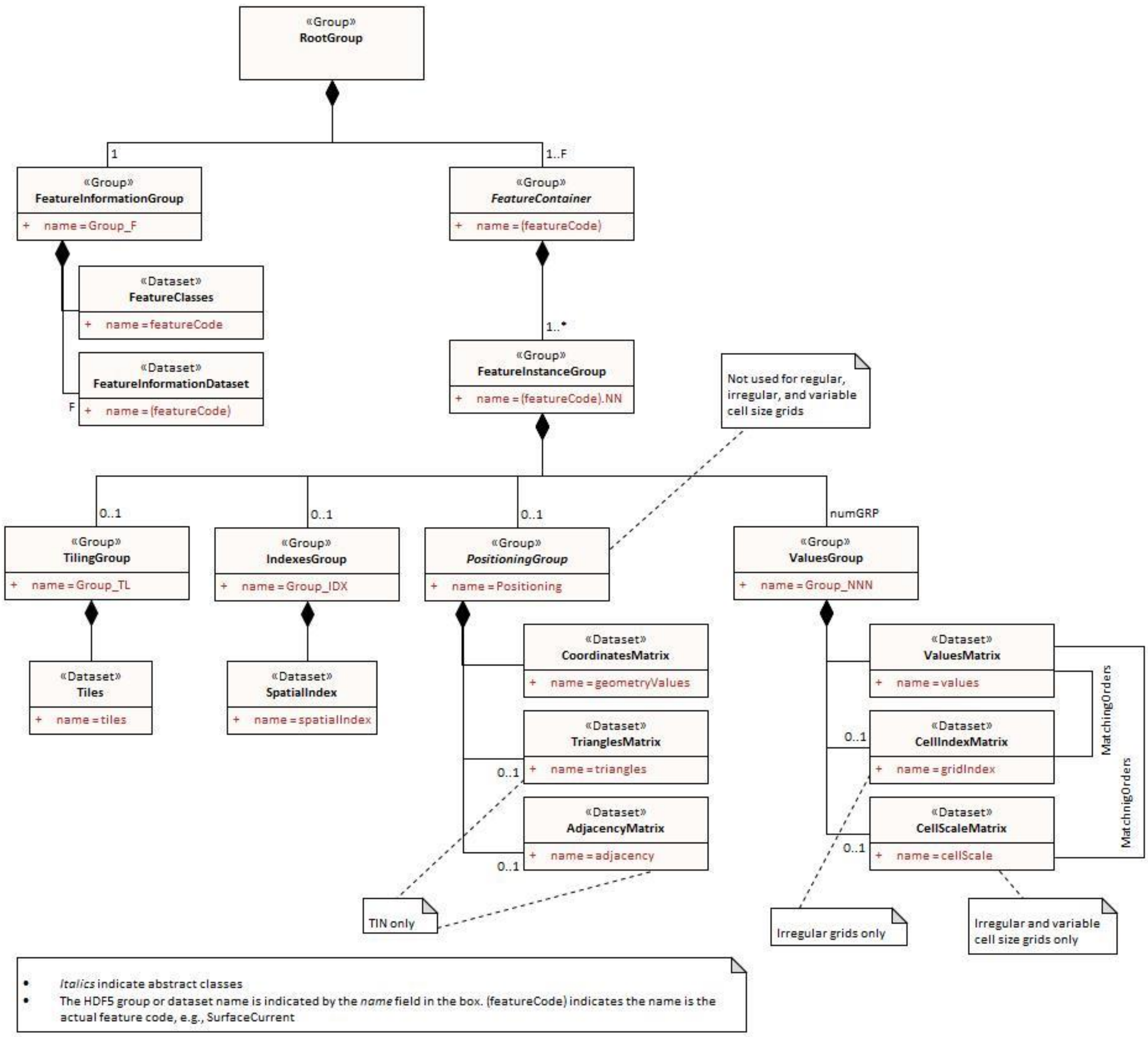

Figure 10c-11 - Conceptual model of content

10c-16 Rules for product specification developers

10c-16.1 Defining the format for a product specification from this profile

Most product specifications will need only a subset of this profile. However, all product specifications must include the mandatory elements of this profile.

The logical structure of the datafile must conform to the logical structure depicted in Figure 10c-11 and specified in the preceding sections.

The 'Data Format' section of the product specification must indicate what part of the profile is used (e.g., which values *dataCodingFormat* can take, which groups and datasets are used, whether the spatial representation is 2-dimensional, 3-dimensional, etc.)

UML diagrams derived from the conceptual structure depictions in this Part are recommended but not mandatory. Documentation tables specifying product-specific constraints or limitations on metadata and content must be provided unless the corresponding table in this profile applies without modification.

Specifications which require grids with non-uniform spacing must be treated as ungeorectified grids and have the coordinates of each position explicitly encoded.

This profile does not prevent a feature class from having different coverage types of coverage, but repeating spatial attributes for the same instance is not possible in this profile. This means that a feature instance cannot have two grids, whether or not they are the same coverage type. If product specifications appear to need multiple coverages for the same instance, consider combining the two into a single coverage object or using two feature instances.

Feature and information associations are not fully implemented in this profile. However, it is possible to link coverage objects to vector feature or information objects in accompanying GML or ISO 8211 datasets using the object reference methods described in section 10c-12. References to vector objects, such as influence polygons must be encoded using the same method.

10c-16.2 Miscellaneous rules

The use of variable length strings as components of compound types is discouraged due to reported performance problems.

In theory, the use of tiles can interact with HDF5 chunking to affect performance. Product specifications for which performance is a significant consideration may need to consider possible interaction effects and investigate their magnitude and consequences.

10c-16.3 Extensions of this profile

Product specifications may extend the format in this profile by defining additional data structures or extending the data structures defined in this profile, but all extensions must retain the core specifications of this profile so that **implementations must be able to ingest and portray data without processing the additional data structures**. **The product specification must be written so that use of these extra data structures for processing or portrayal is optional.**

Such additions should be placed in the appropriate location in the HDF5 data file, e.g., spatial indexes in the Group_IDX group.

Some examples of permissible and impermissible extensions are are given below.

- Permissible extensions:
  - Quadtree index, added as a dataset in the indexes group.
  - Extension of the value record structure that retain the core format described in this profile (i.e., the 1-d array structure and the specified components).
  - Linear scale arrays indicating the grid points on each axis where the cell size changes, as an adjunct to variable cell size arrays.
  - Product-specific metadata as attributes of any of the groups specified in this profile.
  - Product-specific metadata as additional datasets in any of the groups specified in this profile.
  - Additional groups, provided these are not used as substitutes for one of the mandatory groups in this profile.
- Impermissible extensions:
  - Changes to the rank of an array dataset type, e.g., using a 2-d array in place of a 1-d array.
  - Changes to the rules for naming of a component of a compound data type defined in this profile.

10c-16.4 Adding metadata

While section 10c-16.3 permits adding metadata, defining product-specific metadata means that implementation must – if they are to do anything with the additional metadata other than merely display it – include product-specific coding in applications. Given that the S-100 ecosystem includes multiple data products which would ideally all be processable (including portrayal) by an S-100 application, this Part recommends against adding product-specific metadata that has any effects on processing or portrayal. If such additions are considered essential they should be proposed as an extension to the S-100 standard itself using the maintenance mechanism described in the S-100 standard and related documents. Display-only metadata (i.e., where the application is only expected to display the content of the added attribute) may be added but is discouraged.

10c-17 Implementation guidance

The HDF5 C API includes interfaces for determining the types of compound type components. Ths suggests that the size of a datatype can be checked to mitigate possible conversion issues.

The HDF5 C API also defines iterators for iterating over attributes or items in a group. These iterators can be used to discover profile datasets, groups, or attributes from datasets, groups, and attributes defined only in individual product specifications.

The order in which objects are retrieved may not be the same as the creation order. Implementers should allow for this or investigate the availability of order-preserving functions in the HDF5 API.

Linkage between the XML feature catalogue and objects in the HDF5 file is preserved by using the codes for features, and attributes