Format Specification Document

Description of Bathymetric Attributed Grid Object (BAG) Version 1.5.1

Document Version
RELEASE 1.5.1

by
Open Navigation Surface Working Group
(ONSWG)

Format Specification Document	**Document Title:** Description of Bathymetric Attributed Grid Object (BAG)		
Document History	**Version**	**Comments**	**Date**
	Draft 1.0	Initial draft.	2004-01-23
	Draft 1.1	Reviewed for release of BAG API Version 1.0.0.	2006-03-31
	Release 1.0	Released along with BAG API Version 1.0.0.	**2006-04-07**
	CR 1.1	Expanded section 5.3 to include details of XML metadata library	**2009-07-10**
	CR 1.1	Modified the appropriate sections to include the description of the optional datasets within the BAG API.	**2009-07-20**
	Release 1.5	Updates to document to include changes since 1.1	**2012-05-18**
	Release 1.5.1	Updates to document to include changes in 1.5.1	**2013-07-12**
Status	Candidate Release 1.5.2 to be released summer 2013		

Number of pages:	53
Number of annexes:	0
The Open Navigation Surface Working Group	
National Ocean Service, National Oceanic and Atmospheric Administration, Silver Spring MD, USA.	Richard T. Brennan, Barry Gallagher, Jack Riley,
Center for Coastal and Ocean Mapping Joint Hydrographic Center, University of New Hampshire, Durham NH, USA.	Brian Calder, James D. Case,
SAIC, Marine Survey and Engineering Solutions Division, Newport RI, USA.	Shannon Byrne, Webb McDonald,
Naval Oceanographic Office, Stennis Space Center MS, USA.	David Fabre, R. Wade Ladner,
CARIS Ltd., New Brunswick, Canada.	Bill Lamey,
Seven Cs GmbH & Co. KG, Hamburg, Germany.	Friedhelm Moggert,
IVS3D Ltd., New Brunswick, Canada.	Mark Paton.

Authors:	Brian Calder, Shannon Byrne, Webb McDonald, Wade Ladner, Bill Lamey, Friedhelm Moggert, Elizabeth Warner Michelle Russell

Contents

CONTENTS 4

1. INTRODUCTION 6

1.1. MOTIVATION 6
1.2. NOMENCLATURE 6
1.3. PROPERTIES OF THE BAG 2
1.4. STATUS OF THE PROJECT 2
1.5. HISTORY 3
1.6. IDENTIFICATION 3

2. OVERVIEW OF THE BAG STRUCTURE 4

2.1. TOP-LEVEL REQUIREMENTS 4
2.2. TECHNOLOGY BASE 4
2.3. BAG FILE LAYOUT 5
2.4. METADATA 7
2.4.1. REQUIREMENTS FOR METADATA 7
2.4.2. STANDARD OBJECT MODEL 8
2.4.3. COMMON ENCODING 10
2.4.4. INTERNAL PARSING AND VALIDATION 10
2.5. ELEVATION GRID 10
2.6. UNCERTAINTY GRID 11
2.7. TRACKING LIST 11
2.8. EXTENSIONS 11
2.8.1 Nominal ELEVATION 11
2.8.2 Vertical Surface Correctors 11
2.8.3 ELEVATION SOLUTION GROUP 112
2.8.4 NODE GROUP 113
2.9. CERTIFICATION 133
2.9.1. INTENT OF THE BAG DIGITAL SIGNATURE SCHEME 133
2.9.2. DSS IMPLEMENTATION 133
2.9.3. STRUCTURE OF THE BAG CERTIFICATION BLOCK 144

3. ENCAPSULATION 16

4. AXIOMATIC DEFINITIONS 22

4.1. PURPOSE 22
4.2. COORDINATE SYSTEM ORIENTATION AND GEO-REFERENCING 22
4.3. UNITS 22
4.4. RESOLUTION AND PRECISION OF DATA 23

5. THE BAG API 24

5.1. STRUCTURE OF THE SOURCE TREE 24
5.2. BASIC DATA ACCESS 24
5.3. METADATA ACCESS 25
5.3.1 BAG XML METADATA LIBRARY 26
5.3.1.1 MD_METADATA 27
5.3.1.2 IDENTIFICATIONINFO 28
5.3.1.3 METADATACONSTRAINTS 31
5.3.1.4 DATAQUALITY 32
5.3.1.5 SPATIALREPRESENTATIONINFO 38
5.3.1.6 REFERENCESYSTEMINFO 41

5.3.1.7 LANGUAGE 45
5.3.1.8 CONTACT 45
5.3.1.9 DATESTAMP 46
5.3.1.10 METADATASTANDARDNAME 46
5.3.1.11 METADATASTANDARDVERSION 46
5.4. TRACKING LIST ACCESS 46
5.5. DIGITAL SIGNATURES 46
5.6. ERROR CODES AND REPORTING 47

6. BAG ARCHITECTURE REVIEW BOARD 48

6.1. INTENT OF THE ARB 48
6.2. COMPOSITION AND ADMINISTRATION 48

7. REVISION CONTROL AND CODE AVAILABILITY 49

8. HOW TO APPLY FOR AN EXTENSION/BUG FIX 50

8.1. NOMINATION PROCESS 50
8.2. REQUEST FOR AN EXTENSION HDF GROUP 50
8.3. REQUEST FOR AN EXTENSION TO AN EXISTING GROUP 51
8.4. REQUEST FOR A BUG FIX 51

GLOSSARY OF ACRONYMS 52

REFERENCES 53

APPENDIX A 54

1. Introduction*

1.1. Motivation

The design of the 'Navigation Surface' concept [2, 3] envisioned a new structure for hydrographic product creation from basic surveys. Relying on a database to hold all the original data, processed into the form of grids of the best available representation of the true nature of the seafloor, the Navigation Surface concept extracts whatever data is required for a particular product and, through automatic manipulation and/or cartography, constructs a product suitable for a particular purpose.

Depending on task, this might be a uniform grid at the best available resolution (e.g., for flow modeling), or a hydrographic vector-based chart for safety of navigation. In order to make this possible, the database must contain the best available data for each area, at the highest achievable resolution, and will contain data from a number of different sources, potentially from a number of different software packages.

In order to make this database possible, there is a need for a uniform file format that allows data to be passed between software packages, and between agencies involved in the collection, processing and dissemination of the data, while maintaining the integrity of the data and metadata at all times. The file format should also be sufficiently flexible to support processing of data without format conversion where possible, since format conversion is an extremely expensive process. The Open Navigation Surface (ONS) Project has as its mandate the task of building such a data file format, and developing and maintaining the source code for a software library to read and write this format so that adoption of the technology is eased for any developer. The source code library has been developed on the Open Source model [4] so that the source code is freely available; all of the members of the ONS Working Group (ONSWG) and their respective employers have provided their effort on this basis.

This document describes the format, the Application Programming Interface (API) associated with it, and the conduct of the ONSWG and its derivatives that maintain the BAG file format source code. It also describes where to find the code, how to obtain access to the current source tree, how to apply for extensions to the format, and how to provide bug-fixes. The development effort is entirely voluntary, and participation from other interested parties is actively encouraged.

1.2. Nomenclature

The term "Navigation Surface" was coined to describe the combination of a data object representing the bathymetry and associated uncertainty, and the methods by which such objects could be manipulated, combined and used for a number of tasks, including products in support of safety of navigation. These multiple goals have led to some uncertainty about what exactly constitutes a Navigation Surface. To avoid any further confusion, a revised nomenclature has been designed.

* Some parts of this document have been adapted from the paper on the Open Navigation Surface published in the International Hydrographic Review [1]; a copy of the full paper is available from the project's web-site, http://www.opennavsurf.org.

In the ONS model, a unit of bathymetry is termed a Bathymetric Attributed Grid (BAG). A single BAG object represents one contiguous area of the skin of the Earth at a single resolution, but can represent data at any stage of the process from raw grid to final product. The name Navigation Surface (NS) is reserved for a final product BAG destined specifically for safety-of-navigation purposes. The status of any particular BAG is distinguished solely by the certification section of metadata embedded in the file.

1.3. Properties of the BAG

The Navigation Surface concept requires that in addition to estimation of depth, an estimate of the uncertainty associated with the depth must be computed and preserved. In order to make the system suitable to support safety of navigation applications, there is a means to over-ride any automatically constructed depth estimates with 'Hydrographer Privilege', (essentially, a means to specify directly the depth determined by a human observer as being the most significant in the area - irrespective of any statistical evidence to the contrary).

There is also the requirement to provide data on the data (i.e. metadata, which describe all aspects of the data's life, from methods of capture to processing methods, and from geospatial extents to responsible party.

Finally, there must be a means to certify that the data in the file has been inspected by someone with appropriate experience and authority, that the data has been verified as suitable for some specific purpose, and that the file has not changed since this certification was made: in essence, a digital replacement for the Hydrographer's signature.

Means to incorporate all of these requirements in a portable, extensible, platform neutral, vendor neutral format are provided.

1.4. Status of the Project

The ONSWG maintain both a web-site, `http://www.opennavsurf.org`, which contains the current recommended release of the source code and supporting documentation, and a SubVersion [5] server, which contains the current development version of the source code. Details of availability of the code are given in Section 7. There are no mechanisms for 'nightly builds' or other intermediate releases.

The library is designed to be built from source code, including all of the component libraries that the BAG uses. All of the component libraries are covered under Open Source style licenses, although not all the licenses are alike. Users should ensure that they are compliant with the terms of the appropriate license before using the component libraries. The ONS source code library itself is Open Source in the sense that you are free to modify, adapt and otherwise reuse the source code, including the construction of derivative products based on the source code. However, if a bug is found or there is a way to improve part of the library, it is requested that this be communicated to the ONSWG (as described in Section 8). Extensions to the library (e.g., new HDF groups (Section 3) or additions to metadata elements) shall not be added without permission of the BAG Architecture Review Board (Section 6). Applications for extensions are also covered in Section 8.

1.5. History

The idea for what became the Open Navigation Surface Project had been discussed within the US hydrographic community for some time, but first crystallized in late 2003 through comments of M. Paton at IVS3D Ltd., which were adopted by B. Calder at CCOM/JHC with the intent of acting as an independent third-party broker for the development of the library. The ONSWG first met in early 2004, and outlined the requirements for the BAG file structure, the ethos of the project and the basic functionality of the source code library. A presentation on the structure of the project was made at US Hydro 2005 in March, and development continued until the second ONSWG meeting in mid 2005, when the majority of the first release of the library was pulled together in a little under a week. A second presentation, including demonstrations of the library linked into SAIC SABER, CARIS HIPS and IVS Fledermaus, was given at Shallow Survey 2005 in September and development of the library continued using e-mail and telephone conferencing until the first Candidate Release on 8 February 2006. Following comments from users and further development, the first official release of the library was made on 8 April 2006.

1.6. Identification

This document describes the Bathymetric Attributed Grid file format, version 1.5.1. This version corresponds to the SubVersion release tag "`release-fr-1.5.1`", which may be obtained from the Open Navigation Surface (ONS) Project web-site, `http://www.opennavsurf.org`, as either source code or compiled libraries. It may also be obtained from the ONS SubVersion server with appropriate authorization. Details of how to apply for access are given in Section 7.

This release of the library supports, and has been tested on, Windows 2000, XP, Vista and 7 and Red Hat Enterprise Linux 3.0, 4.0, and 5.8. The source code release is generic ANSI C and C++, and should therefore compile on most Unix-like systems if required. Users who successfully compile the library on other platforms are urged to provide information on any modifications required to the project so that these platforms can be supported in future releases.

It is the intent of the ONSWG to support as many other Unix-like platforms as possible in future releases, including Open Solaris.

2. Overview of the BAG Structure

2.1. Top-Level Requirements

Several top-level requirements were identified during the inaugural ONSWG meeting early in 2004. These requirements provided the foundation on which many of the architectural decisions have been made. In particular, the ONSWG adopted the following high level requirements:

1. A freely-distributable source code library shall be available for the BAG.

2. BAG shall adhere to a defined format, expressed via the source code library and accompanying documentation.

3. The ONSWG shall provide and oversee a change process to allow for incorporation of bug fixes to the library and extensions to the format.

4. The BAG library and file format shall be operating system independent, and supported on at least the Win32, and Linux operating systems.

5. The BAG library and file format shall support files larger than 2 gigabytes.

6. The BAG file format shall include mandatory data elements, and optional extensions.

7. The BAG mandatory elements include: metadata, elevation grid, vertical uncertainty grid, change tracking list, and digital signature.

8. The BAG file format and access library shall maintain all mandatory data elements within a single, self contained, file structure.

9. The BAG digital signature shall provide a mechanism to describe intended use of the dataset.

10. The BAG digital signature shall provide a mechanism to validate that the contents of the file have not changed since original signature.

11. Extensions to the BAG file format shall be fully defined and approved by the ONSWG prior to adoption.

12. The ONSWG shall maintain a web site that describes the status of the project and allows for releases of the library to be downloaded.

13. The ONSWG shall maintain a revision control system to manage configuration baselines of the BAG library software and documentation.

2.2. Technology Base

As part of the BAG design process, members of the ONSWG reviewed several alternatives for the core technology base for data encapsulation, file I/O, metadata encapsulation, and cryptography. The group's basic concept was to adapt existing software technologies where appropriate, with obvious rational based on time and cost to develop. After some review, the group agreed to adopt Hierarchical Data Format version 5 (HDF-5) to provide the basic mechanisms of data encapsulation, and file I/O [10]. The group also agreed to adopt XERCES [11] as the preferred XML parser. Beecrypt was adopted to satisfy BAG's cryptography needs [12]. Fig. 1 provides a graphic overview of the external libraries on which the BAG library depends.

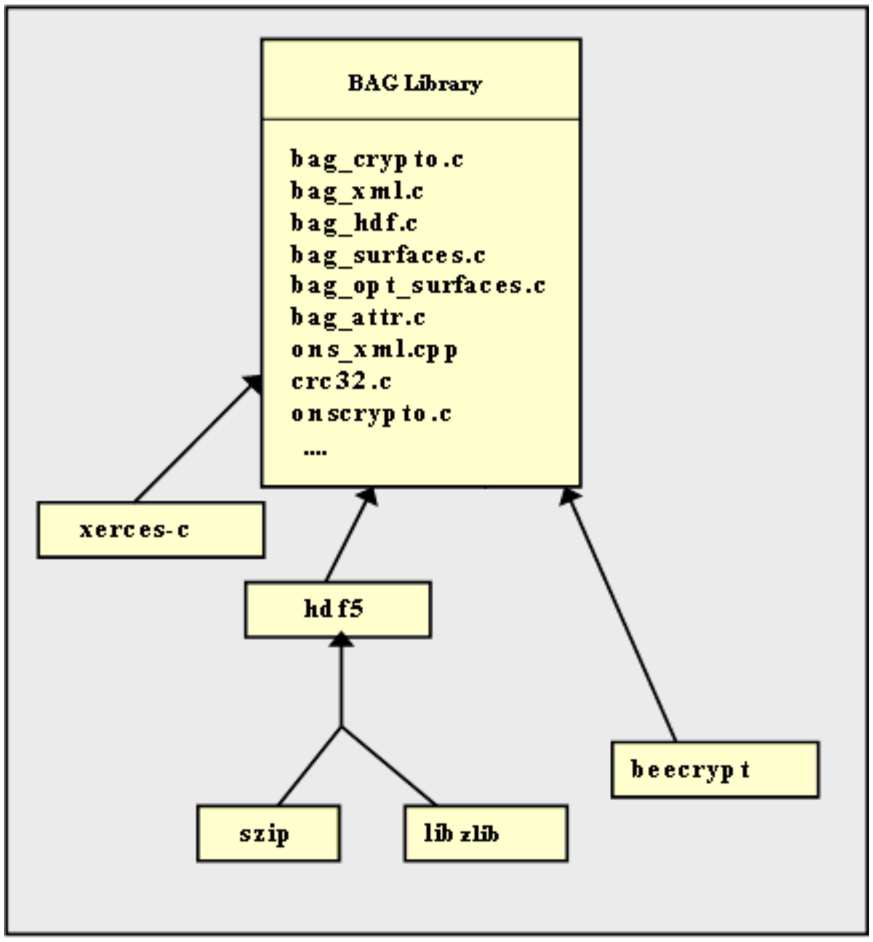


Fig. 1: HIERARCHY OF BAG EXTERNAL LIBRARY DEPENDENCIES.

2.3. BAG File Layout

Fig. 2 provides a conceptual view of the contents of each of the mandatory data elements of a BAG. The elevation grid contains the two-dimensional array of bathymetry data. The uncertainty grid is co-located with the elevation grid and contains a two-dimensional array describing the vertical uncertainty of the elevation grid. The tracking list details hydrographer modifications, and the certification section specifies authenticity and intended use of data.

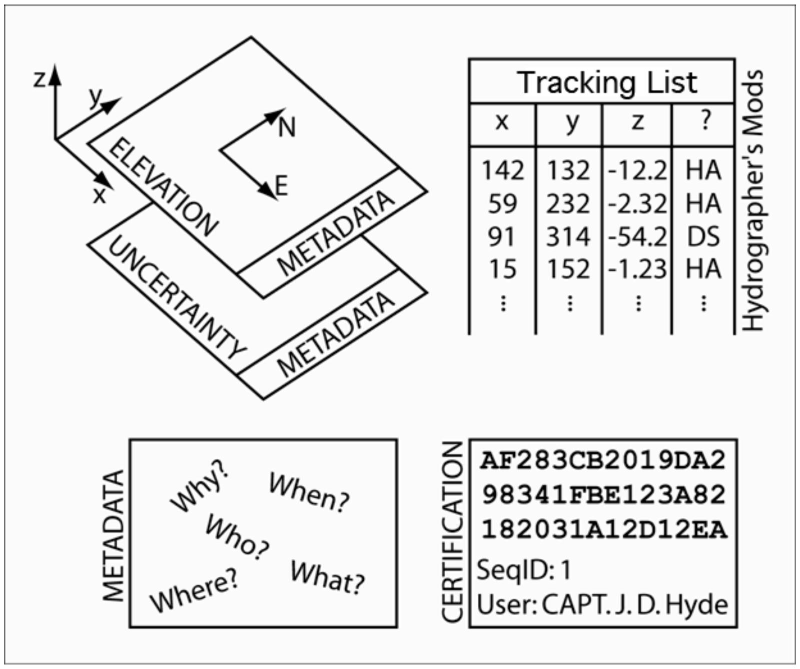


Fig. 2: MANDATORY ELEMENTS OF A BAG.

Fig. 3 shows the structural layout of the mandatory elements of the BAG. The version tag is a simple text string that specifies the version of the format at the time the BAG was created. The metadata, elevation, uncertainty, and change-list, are each HDF-5 datasets with a dataset substructure appropriate for the information being stored. The signature section is simply a byte stream appended after the end of the HDF-5 file using standard C file access mechanisms. The rationale for adding the signature byte stream after the end of the HDF-5 file is to ensure that the contents of the signature do not modify the file that it is trying to protect.

The BAG file format also allows for optional layers to be added to the structure in addition to the mandatory elements. Fig 6 shows the addition of optional surface layers as defined in the BAG_SURFACE_PARAMS structure. These optional surface layers are each HDF-5 datasets defined at the root level of the BAG and follow the same substructure as the elevation and uncertainty datasets.

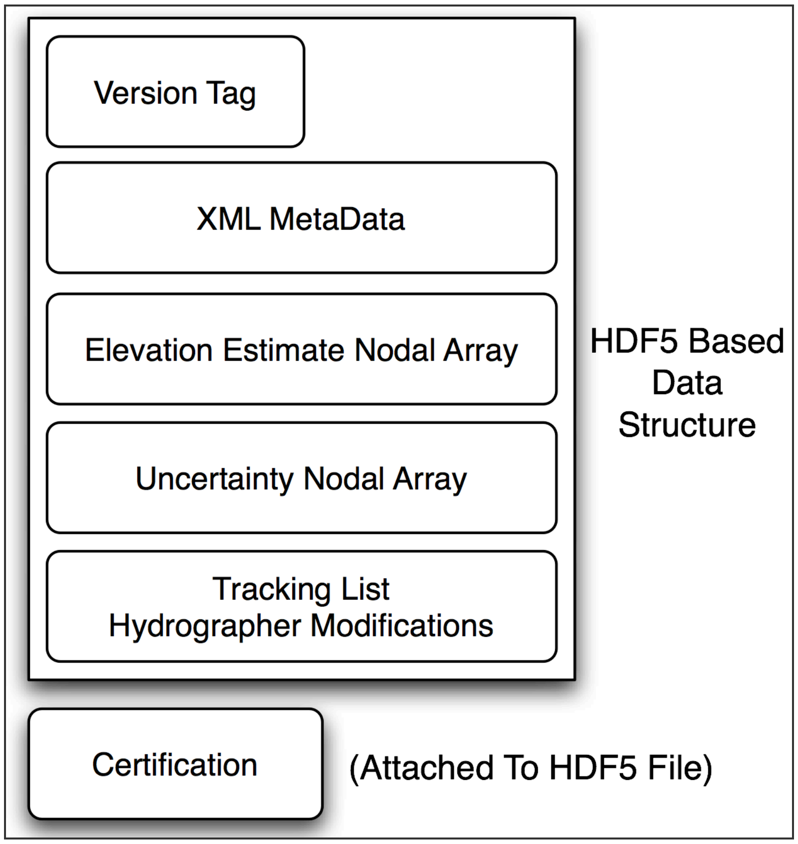


Fig. 3: BAG FILE STRUCTURE.

In version 1.5.1, a BAG has a single fixed node-spacing (referred to as grid resolution), and represents a contiguous region of the surface of the earth. A future version of BAG may allow for storage of multiple grids. For all practical purposes, grid size is constrained only by available disk space. Row and column values are dimensioned as unsigned 32 bit integers, and HDF-5 places no fixed upper limit on row or column dimension or on file size. In version 1.4.0 compression capability was added to the BAG API to reduce disk space usage [13].

Section 4, Axiomatic Definitions, details the coordinate system and standard units of measure used by the BAG library. Every effort has been made to ensure that the numeric values passed through the BAG library function interfaces all adhere to the definitions detailed in Section 4.

2.4. Metadata

2.4.1. Requirements for metadata

With the evolution of digital data, also comes the necessity to accurately track and attribute this data. BAG is no exception. One of the primary goals behind the BAG initiative was to provide a way to uniformly encode, decode and exchange the information about who, what, when, where, and how the BAG file was created.

2.4.2. Standard object model

During the first meeting of the Open Navigation Surface Working Group in January 2004, it was concluded that the ISO metadata standards 19115 and 19115-2 would be used as the common framework to store the metadata information needed by the BAG format. A sub-working group was selected and assigned to work on creating the appropriate metadata profile to describe the BAG file contents. The profile created is depicted in Fig. 4. This profile describes the minimum required fields for a valid BAG file. However, the user is not limited to this profile, and may include any of the other components described in the 19115 standard.

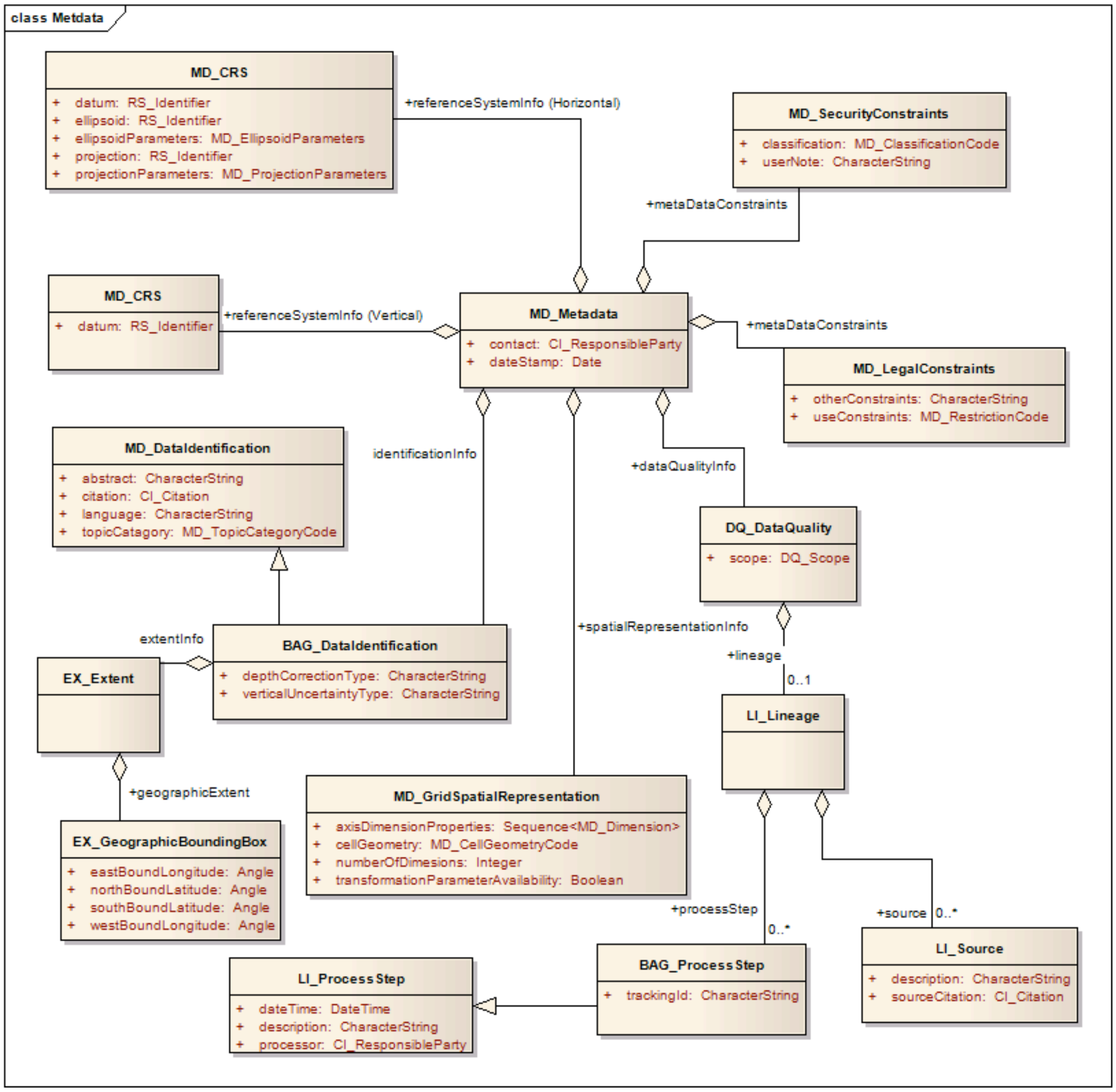

class Metdata
MD_CRS
+ datum: RS_Identifier
+ ellipsoid: RS_Identifier
+ ellipsoidParameters: MD_EllipsoidParameters
+ projection: RS_Identifier
+ projectionParameters: MD_ProjectionParameters
+referenceSystemInfo (Horizontal)
MD_SecurityConstraints
+ classification: MD_ClassificationCode
+ userNote: CharacterString
+metaDataConstraints
MD_CRS
+ datum: RS_Identifier
+referenceSystemInfo (Vertical)
MD_Metadata
+ contact: CI_ResponsibleParty
+ dateStamp: Date
+metaDataConstraints
MD_LegalConstraints
+ otherConstraints: CharacterString
+ useConstraints: MD_RestrictionCode
MD_DataIdentification
+ abstract: CharacterString
+ citation: CI_Citation
+ language: CharacterString
+ topicCatagory: MD_TopicCategoryCode
identificationInfo
+dataQualityInfo
DQ_DataQuality
+ scope: DQ_Scope
extentInfo
BAG_DataIdentification
+ depthCorrectionType: CharacterString
+ verticalUncertaintyType: CharacterString
+spatialRepresentationInfo
+lineage
0..1
EX_Extent
LI_Lineage
+geographicExtent
MD_GridSpatialRepresentation
+ axisDimensionProperties: Sequence<MD_Dimension>
+ cellGeometry: MD_CellGeometryCode
+ numberOfDimesions: Integer
+ transformationParameterAvailability: Boolean
EX_GeographicBoundingBox
+ eastBoundLongitude: Angle
+ northBoundLatitude: Angle
+ southBoundLatitude: Angle
+ westBoundLongitude: Angle
+processStep
0..*
+source
0..*
LI_Source
+ description: CharacterString
+ sourceCitation: CI_Citation
BAG_ProcessStep
+ trackingId: CharacterString
LI_ProcessStep
+ dateTime: DateTime
+ description: CharacterString
+ processor: CI_ResponsibleParty

Fig. 4: UML MODEL FOR BAG METADATA PROFILE.

The ISO 19115 object model provided most of the needed classes and attributes required to fully describe the BAG content. There were three extensions made to the model to allow for detailed description of the data. First, the `verticalUncertaintyType` attribute was added to allow the BAG file to accurately describe the source and meaning of the encoded Uncertainty grid layer. Table 1 shows the list of supported values as of the 1.5.1 release.

Secondly, the `trackingId` extension allows internal Tracking List entries to be associated with a unique entry in the metadata so that the changes can be properly attributed, described and easily referenced.

Thirdly, the `depthCorrectionType` extension was added to allow the BAG file to accurately describe the correction performed on the data. Table 2 shows the list of supported values as of the 1.5.1 release.

Enumerated Value	Description
Unknown	"Unknown" - The uncertainty layer is an unknown type
Raw_Std_Dev	"Raw Standard Deviation" - Raw standard deviation of soundings that contributed to the node
CUBE_Std_Dev	"CUBE Standard Deviation " - Standard deviation of soundings captured by a CUBE hypothesis (i.e., CUBE's standard output of uncertainty)
Product_Uncert	"Product Uncertainty" - NOAA standard product uncertainty V1.0 (a blend of CUBE uncertainty and other measures).
Historical_Std_Dev	"Historical Standard Deviation " – Estimated standard deviation based on historical/archive data.
Average_TPE	Average of all of the contributing sounding TPEs within the node.

Table 1: TABLE OF UNCERTAINTY TYPES

Enumerated Value	Description
True_Depth	"True Depth" – Depth corrected for sound speed
Nominal_Depth_Meters	"Nominal at 1500m/s"- Depth at assumed sound speed of 1500m/s
Nominal_Depth_Feet	"Nominal at 4800ft/s" – Depth at assumed sound speed of 4800ft/s
Corrected_Carters	"Corrected via Carter's Tables" – Corrected depth using Carter's tables
Corrected_Matthews	"Corrected via Matthew's Tables" – Corrected depth using Matthew's tables
Unknown_Correction	"Unknown" – Unknown depth correction type or mixture of above types

Table 2: TABLE OF DEPTH CORRECTION TYPES

2.4.3. Common encoding

After choosing the ISO standard object model for metadata encapsulation, a format was needed to store this structure. The ISO had already begun work on the 19139 specification for storing the 19115 structure in XML format. It was decided that XML seemed to be a logical medium as it was both machine and human readable, and is already a well known storage format for many other standards.

To keep the library API flexible, the metadata is provided to the library as a character stream. The stream may be validated and is stored in its own HDF-5 block at the head of the file. This data block in the HDF-5 file is configured as an extendable block to allow the metadata to grow in the future and not disturb the remaining data blocks. In a similar fashion, the metadata can be extracted from the file in its native XML character stream and thus can be processed by any regular XML tools.

2.4.4. Internal parsing and validation

Each time a metadata stream is processed by the library (for read, creation, or update) it must be validated to ensure that it is well-formed XML and, more importantly, meets the requirements of the defined BAG profile. In order to perform this validation, the library requires an internal XML parser. Given that the 19139 XML schema is quite complex, the XERCES parser is used to read the XML stream, associated schema files, and perform all validation. This validation occurs transparently to the user during the open or creation API routines.

2.5. Elevation Grid

The elevation section of the BAG contains a two-dimensional matrix of elevation values, organized in row major order, and starting from the south-western most data point, where each value is defined to be the elevation at an exactly specified geographic point (node).

An important distinction to note is the contrast between this node-based approach where each grid value describes the elevation exactly at a specified geographic point, as compared with a cell oriented approach, where each grid value might describe the elevation as applying over the extents of the cell size. The elevation data in a BAG describe the elevation only at the exact node locations, and offer no information about the elevation between the nodes.

The units of the elevation values are meters, and the sign convention is for z to be positive for values above the vertical datum. The reference vertical datum for the BAG is one of the mandatory Metadata items. This sign convention follows directly from the right hand coordinate system definition that BAG adheres to. See section 4, for additional information. The elevation values are encoded as four byte floats within an HDF-5 dataset. The x and y location of each elevation value is not explicitly saved within the dataset.

Rather, the x, y location of the south-western most point is saved within the metadata section, along with the x node spacing and the y node spacing. Fixed, regular spacing between nodes is required. The unknown state for elevation is defined to be 1,000,000.0 (1.0e6).

2.6. Uncertainty Grid

The uncertainty section of the BAG contains a two-dimensional matrix of values that specify the vertical uncertainty at each node. The elevation grid and the uncertainty grid are explicitly co-aligned. The values are expressed as positive quantities in units of meters.

As detailed in Table 1, the uncertainty grid supports multiple definitions of vertical uncertainty. This allows BAGs to span the expected range of data products from raw, full resolution grid to final compiled product. For example, a BAG at the stage of final survey data processing should contain uncertainty information germane to the survey data itself and intended to be used for information during compilation. A BAG intended for navigational purposes would need to specify the overall uncertainty to the mariner – these two values for uncertainty may be quite different.

A recipient of a BAG file can refer to the uncertainty definition in the Metadata to gain an understanding of how the uncertainty was computed. The uncertainty values are encoded as four byte floats within an HDF-5 dataset. The unknown state for uncertainty is defined to be 0.0.

2.7. Tracking List

The tracking list contains a simple list of the original elevation and uncertainty values from any node of the surface that has been modified to account for hydrographer over-rides of the basic surface definition (e.g., as originally computed by an algorithmic method). The tracking list dataset and corresponding information contained in the metadata exist to provide an audit trail record of changes made to the data by manual intervention. The contents of the tracking list dataset are shown in Fig. 5.

Field	Size
row	4
col	4
depth	4
uncertainty	4
track_code	1
list_series	2

Fig. 5 TRACKING LIST ITEM – SIZE IN BYTES

The *row* and *col* fields describe the node that has been modified. The *depth* and *uncertainty* fields contain the original values prior to manual intervention. The *track_code* field provides a reason code for the manual modification of a grid surface node.

The *list_series* field of the tracking list item structure is used to identify an entire series of tracking list items and associate them with supplemental details in the BAG Metadata. There should be corresponding Metadata entries for each suite of *list_series'* tracking list items. One can then gather from the Metadata the circumstances and motivation behind a particular group of edits to the BAG depth and uncertainty surfaces.

2.8. Extensions

Extensions are optional and not required for a HDF file to be qualified as a BAG. Vendors or other third parties can apply for extensions to the format by the method outlined in this requirements document (please refer to section 8).

2.8.1. Nominal Elevation

Note Fig. 6 shows an extension made to the BAG API for Version 1.5.1, regarding the addition of optional datasets. As listed in the BAG_SURFACE_PARAMS, Nominal Elevation is one of the optional layers that can be included in the BAG.

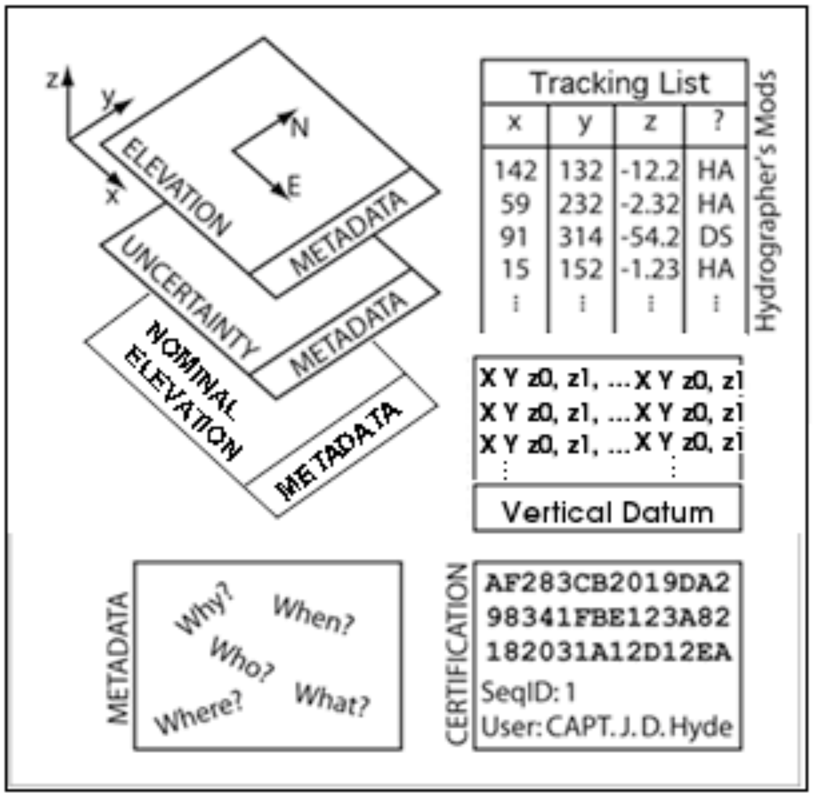


Fig 6. EXAMPLE OF BAG WITH OPTIONAL DATASET OF NOMINAL ELEVATION

2.8.2. Vertical Surface Correctors

This optional dataset may comprise a regularly-spaced grid topography or a set of irregularly spaced coordinates. The intended approach is that the BAG creator provides sufficient data density to allow a simplistic inverse distance interpolation scheme to generate correction values at the full resolution of the BAG elevation nodes. The library includes an API function to perform this computation based on the surface correctors dataset and the desired BAG surface for either type of topography (regular or irregular spacing).

2.8.3. Elevation Solution Group

The elevation solution group is designed to contain the shoal elevation, standard deviation, and number of soundings associated with a node in the grid. The *shoal elevation* is the elevation value of the least-depth measurement selected from the sub-set of measurements that contributed to the elevation solution. The *number of soundings* is the number of elevation measurements selected from the sub-set of measurements that contributed to the elevation solution. The *stddev* is the standard deviation computed from all elevation values that contributed to the node. Note that the *stddev* value is computed from all measurements contributing to the node, whereas *shoal elevation* and *number of soundings* relate to the chosen elevation solution.

2.8.4. Node Group

The *node group* has two components: *hypothesis strength* and *number of hypotheses*. The *hypothesis strength* and *number of hypotheses* are computed in the CUBE/CHRT algorithm.

2.9. Certification

2.9.1. Intent of the BAG Digital Signature Scheme

In a traditional hydrographic processing workflow, there is a strict chain of custody for all data that is to be used for nautical charting. At each stage of the chain, a responsible authority reviews the data and the processes applied to it, and certifies that the data is fit for some intended purpose. This may be that the data is ready for final plotting, that it is ready to be combined with other data in a compilation, or that the compilation is suitable as an aid to safe surface navigation. Generally, this is done by some physical signature on appropriate archival documentation, which is traditionally the hydrographic smooth sheet or fair sheet.

With an all-digital product, however, there is no opportunity to affix a physical signature to the data object. In addition, with a dense data object such as a BAG, the opportunity for single-bit errors in transmission to cause navigationally significant changes to the data, that are otherwise undetectable, is greatly increased. The Digital Signature Scheme (DSS) for the BAG is designed to provide an equivalent analogue for the physical hydrographer's signature, and to ensure that any modifications to the data, either by mistake or malicious action, are readily detectable.

This section describes the implementation for the DSS interaction with the BAG file format in outline. Full details of the process are available in the whitepaper, "Digital Signature Scheme for the Bathymetric Attributed Grid (BAG)" [6], which is available from the project's website.

2.9.2. DSS Implementation

The basic entity of the DSS is the Digital Signature (DS), a multi-byte sequence of digits computed from the contents of the BAG file excluding the certification information and another number, known as the secret key (SK), belonging to the person or entity signing the BAG, known as the Signature Authority (SA). The SK is known only to the SA, and as the name suggests should be kept confidential since knowledge of the SK would allow anyone to certify BAGs as if they were the SA. The DS value can be shown to be probabilistically unique for the contents of the BAG and the SK in the sense that, with vanishingly small probability, no two BAGs would generate the same DS with a particular SK, and no two SKs would generate the same DS with the same BAG.

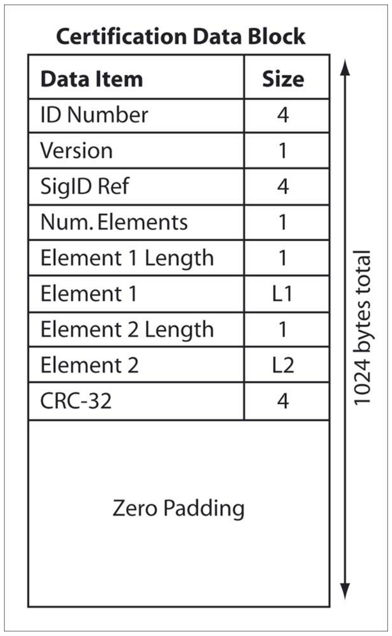
Corresponding to the SK, there is a public key (PK) that can be distributed freely. There is no way to compute the DS using the PK. However, given a BAG and a DS purported to have been constructed with the SK, it is simple to verify whether the BAG has changed, or if another SK was used to construct the certification.

In addition to the basic DS required for the DSS, the BAG certification block contains a 32-bit integer used to link the certification event with an entry in the metadata's lineage section which describes the reasons for certification. The intent of this is to ensure that the user can provide suitably flexible descriptions of any conditions attached to the certification event, or the intended use of the data so certified. This 'Signature ID' shall be a file-unique sequentially constructed integer so that a certification block can be unambiguously associated with exactly one lineage element.

The DSS is not mandatory, in the sense that the API does not enforce checks for certification blocks or DS results as BAG files are opened, written or closed. A BAG without a certification block shall be considered valid. The BAG API provides means to construct and verify DS values, but does not address questions of key distribution, certificate generation or certificate signing. Users are urged to consult an appropriate reference (e.g., [7]) for details of these processes.

2.9.3. Structure of the BAG Certification Block

The BAG DS information shall be maintained in a certification block of length 1024 bytes, appended to the end of the HDF-5 data. The structure of the certification block shall be as shown in Fig. 7. The ID number shall be a 'magic number' to identify the block, and the version byte shall be used to identify the structure of the remainder of the block between different versions of the algorithm. The SigID number corresponds to the Signature ID described above, and shall be followed immediately by the DS values which shall be stored sequentially as a length byte followed by the digits of the element. The CRC-32 checksum shall be used to ensure that any accidental or intentional corruption of the certification block will be detectable. The block shall be stored in little endian format, and zero padded to the full size of the block.



Certification Data Block

Data Item	Size
ID Number	4
Version	1
SigID Ref	4
Num. Elements	1
Element 1 Length	1
Element 1	L1
Element 2 Length	1
Element 2	L2
CRC-32	4

Zero Padding
1024 bytes total

Fig. 7: STRUCTURE OF THE BAG DIGITAL SIGNATURE SCHEME CERTIFICATION BLOCK

3. Encapsulation

The BAG structure utilizes HDF-5. HDF-5 is a hierarchical data format product consisting of a data format specification and a supporting library implementation. HDF-5 files are organized in a hierarchical structure, with two primary structures; groups and datasets. They are defined as:

- HDF-5 Group: a grouping structure containing instances of zero or more groups or data sets, together with supporting metadata.

- HDF-5 Data set: a multidimensional array of data elements, together with supporting metadata or attributes.

An HDF-5 "Group" provides the top-level structure for the data contents of a BAG. The major subcomponents are defined using the HDF-5 "Dataset" types, and "Attribute" types. Within each "Dataset", further structural decomposition is specified via the DATATYPE and DATASPACE parameters. "Attributes" are included where appropriate to provide "Dataset" specific metadata. Following the high level BAG file structure described in Fig. 3, the specific HDF-5 type definitions that define the BAG encapsulation structure are illustrated in Fig. 8. Note that the digital signature is not shown in Fig. 8. As described in Section 2, the digital signature byte stream is appended to the end of the HDF-5 group.

```
Group "BAG_root" {
                Attribute "BAG Version"
                Dataset "metadata" {
                        DATATYPE        String
                        DATASPACE       1-dimension, 0-N
                        DATASET {"XML…"}
                }
                Dataset "elevation" {
                        DATATYPE        Floating point 4bytes
                        DATASPACE       2-dimensions, 0-N, 0-M
                        DATASET {{}}
                        Attribute "Minimum Elevation Value"
                        Attribute "Maximum Elevation Value"

                }
                Dataset "uncertainty" {
                        DATATYPE        Floating point 4bytes
                        DATASPACE       2-dimensions, 0-N, 0-M
                        DATASET {{}}
                        Attribute "Minimum Uncertainty Value"
                        Attribute "Maximum Uncertainty Value"
                }
                Dataset "tracking list" {
                        DATATYPE        bagTrackingListItem
                        DATASPACE       1-dimension, 0-N
                        DATASET {}
                        Attribute "Tracking List Length"
                }
        }
```

Fig. 8 STRUCTURE OF BAG DATA ENCAPSULATD USING HDF-5

Table 3 defines the contents of the HDF data elements belonging to the BAG_root Group.

BAG_root Group Definition		
Entity Name	**Data Type**	**Domain**
BAG Version	String	Maximum 32 bytes available
metadata	Dataset	Detailed in table 4
elevation	Dataset	Detailed in table 5
uncertainty	Dataset	Detailed in table 6
tracking list	Dataset	Detailed in table 7, and in table 8

Table 3: CONTENTS OF BAG_ROOT GROUP

Table 4 defines the metadata items used with in the BAG library. These items must be present and properly defined for BAG I/O operations to succeed. Note that this listing of metadata items does not specify the mandatory metadata items required by the ISO 19115 standard.

Metadata Dataset Character String Contents			
Entity Name	**XML Tag Nesting**	**Data Type**	**Domain**
CoordSys			
Coordinate System code	Reference System Info/ projection/ Identifier/ code	Non Null String	Geodetic GEOREF Geocentric Local_Cartesian MGRS UTM UPS Albers_Equal_Area_Conic Azimuthal_Equidistant BNG Bonne Cassini Cylindrical_Equal_Area Eckert4 Eckert6 Equidistant_Cylindrical Gnomonic Lambert_Conformal_Conic Mercator Miller_Cylindrical Mollweide Neys NZMG Oblique_Mercator Orthographic Polar_Stereo Polyconic Sinusoidal Stereographic Transverse_Cylindrical_Equal_Area Transverse_Mercator Van_der_Grinten
Zone	Reference System Info/ projection	integer	[-60,-1] U [1,60]

	Parameters/ zone		
Standard Parallel	Reference System Info/ projection Parameters/ standard Parallel	Decimal Latitude	0 to 2 decimal numbers of range: [-90.0,+90.0]
Longitude Of Central Meridian	Reference System Info/ projection Parameters/ longitude Of Central Meridian	Decimal Longitude	range: [-180.0, +180.0]
Latitude Of Projection Origin	Reference System Info/ projection Parameters/ latitude Of Projection Origin	Decimal Latitude	range: [-90.0,+90.0]
False Easting	Reference System Info/ projection Parameters/ false Easting	Non Negative Decimal	[0.0, …), decimal is guaranteed at least 18 digits
False Northing	Reference System Info/ projection Parameters/ false Northing	Non Negative Decimal	[0.0, …), decimal is guaranteed at least 18 digits
False Easting Northing Units	Reference System Info/ projection Parameters/ false Easing Northing Units	Unit Of Measure	string
Scale Factor at Equator	Reference System Info/ projection Parameters/ scale Factor At Equator	Positive Decimal	[0.0, …)
Height of Prospective Point Above Surface	Reference System Info/ projection Parameters/ height Of Prospective Point Above Surface	Positive Decimal	[0.0, …)
Longitude of Projection Center	Reference System Info/ projection Parameters/ longitude Of Projection Center	Decimal Longitude	range: [-180.0, +180.0]
Latitude of Projection Center	Reference System Info/ projection Parameters/ latitude Of Projection Center	Decimal Latitude	range: [-90.0,+90.0]

Scale Factor at Center Line	Reference System Info/ projection Parameters/ scale Factor At Center Line	Positive Decimal	[0.0, …)
Straight Vertical Longitude from Pole	Reference System Info/ projection Parameters/ straight Vertical Longitude From Pole	Decimal Longitude	range: [-180.0, +180.0]
Scale Factor at Projection Origin	Reference System Info/ projection Parameters/ scale Factor At Projection Origin	Positive Decimal	[0.0, …)
Oblique Line Azimuth Parameter	Reference System Info/ projection Parameters/ oblique Line Azimuth Parameter	Oblique Line Azimuth	AzimuthAngle, azimuthMeasurePointLongitude
Oblique Line Point Parameter	Reference System Info/ projection Parameters/ oblique Line Point Parameter	Oblique Line Point	obliqueLineLatitude, obliqueLineLongitude
Semi-Major Axis	Reference System Info/ Ellipsoid Parameters/ semi Major Axis	Positive Decimal	[0.0, …)
Axis Units	Reference System Info/ Ellipsoid Parameters/ axis Units	Unit Of Measure	String
Spatial Extent			
Horizontal Datum	Reference System Info/ datum/ Identifier/ code	Non Null String	NAD83 – North American 1983 WGS72 – World Geodetic System 1972 WGS84 – World Geodetic System 1984
Number of Dimensions	Spatial Representation Info/ number Of Dimensions	Positive Integer	[0,1,2,…)
Resolution per Spatial Dimension	Spatial Representation Info/ Dimension/ resolution/value	Decimal	(0.0, 1.0e18) Guaranteed 18 digits with optional '.', or leading signs, '+/-'.

Size per Dimension	Spatial Representation Info/ Dimension/ dimension Size	nonnegative integer	[0,1,2,...,2^16-1]
Corner Points	Spatial Representation Info/ corner Points/ Point/ coordinates	Coordinates	1 to 4 points of pointPopertyType [-360.0,+360.0] decimal degrees
West Bounding Longitude	Data Identification/ extent/ geographic Element/ west Bound Longitude	Approximate Longitude	[-180.00, 180.00], maximum 2 fractional digits
East Bounding Longitude	Data Identification/ extent/ geographic Element/ east Bound Longitude	Approximate Longitude	[-180.00, 180.00], maximum 2 fractional digits
South Bounding Latitude	Data Identification/ extent/ geographic Element/ south Bound Latitude	Approximate Latitude	[-90.00, 90.00], maximum 2 fractional digits
North Bounding Latitude	Data Identification/ extent/ geographic Element/ north Bound Latitude	Approximate Latitude	[-90.00, 90.00] , maximum 2 fractional digits
Bag Metadata Extension			
Tracking List ID	Data Quality/ Lineage/ process Step/ tracking Id	Positive Integer	Short (2byte) integer

Vertical Uncertainty Type	Data Identification/ vertical Uncertainty Type	Character String	Unknown = 0,
			Raw_Std_Dev = 1,
			CUBE_Std_Dev = 2,
			Product_Uncert = 3,
			Historical_Std_Dev = 4,
			Average_TPE = 5

Depth Correction Type	Data Identification/ depth Correction Type	Character String	True_Depth = 0,
			Nominal_Depth_Meters = 1,
			Nominal_Depth_Feet = 2,
			Corrected_Carters = 3,
			Corrected_Matthews = 4.
			Unknown_Correction = 5

Table 4: GROUP LEVEL METADATA – GRID PARAMETERS

Data Set Level Attributes - Elevation		
Entity Name	**Data Type**	**Domain**
Elevation	Float 32[][]	(FLT_MIN, FLT_MAX)

Minimum Elevation Value	Float 32	(FLT_MIN, FLT_MAX)
Maximum Elevation Value	Float 32	(FLT_MIN, FLT_MAX)

Table 5: ELEVATION DATASET ATTRIBUTES

Data Set Level Attributes - Uncertainty		
Entity Name	**Data Type**	**Domain**
Uncertainty	Float 32[][]	(FLT_MIN, FLT_MAX)
Minimum Uncertainty Value	Float 32	(FLT_MIN, FLT_MAX)
Maximum Uncertainty Value	Float 32	(FLT_MIN, FLT_MAX)

Table 6: UNCERTAINTY DATASET ATTRIBUTES

Data Set Level Attributes – Tracking List		
Entity Name	**Data Type**	**Domain**
Tracking List Item	Bag Tracking List Item	N/A
Tracking List Length	Unsigned Integer32	[0, 2^32-1]

Table 7: TRACKING LIST DATASET ATTRIBUTES

BAG Tracking List Item Definition		
Entity Name	**Data Type**	**Domain**
Row	Unsigned Integer 32	location of the node of the BAG that was modified
Col	Unsigned Integer 32	location of the node of the BAG that was modified
Depth	Float 32	original depth before this change
Uncertainty	Float 32	original uncertainty before this change
track_code	Char	reason code indicating why the modification was made
list_series	Unsigned Integer 16	index number indicating the item in the metadata that describes the modifications

Table 8: DEFINITION OF CONTENTS OF THE BAG TRACKING LIST ITEM

4. Axiomatic Definitions

4.1. Purpose

A number of components of the BAG file structure are fixed by default. This sub-section defines these components, which are used implicitly and axiomatically throughout the file structure.

4.2. Coordinate System Orientation and Geo-referencing

All valid BAGs shall be represented with a right-handed Cartesian coordinate system. This system shall have the x-axis oriented towards positive eastings (for projected grids), or east (for geographic grids), and y-axis oriented towards positive northings (for projected grids), or north (for geographic grids). These definitions imply that the z-axis for the sounding data is positive away from the center of mass of the earth (i.e., is positive up), rather than the usual hydrographic convention of positive down (i.e., deeper depths are larger numbers and negative depths are above datum). User-level code is free to make this reflection if required, but must write the data using the positive-up convention. In order to make this distinction clear, the term "elevation" is used for the sounding component of the BAG, rather than "depth". The uncertainty component of the BAG shall have the same coordinate system as the elevation component, with the exception that the z-axis is unipolar, and therefore the concept of direction of positive increase is irrelevant.

The geo-referencing for a BAG shall be node-based, referenced from the southwestern-most node in a grid. Each sample in a grid represents the value in the grid at a point location at the coordinate specified, rather than an estimate over any area with respect to the coordinate. The reference position included in the metadata shall be given in the coordinates used for the grid, and shall contain sufficient digits of precision to locate the grid with accuracy no worse than a millimeter on the surface of the ellipsoid of rotation of the chosen horizontal datum.

The grid data in a BAG (either elevation or uncertainty, and any other surfaces that may be added in time) shall be organized in row-major order from west to east, and south to north in the file. The first sample of the grid is the node at the southwest corner of the grid with location as specified by the geo-referencing parameters, the second is one grid resolution unit to the east of that position and at the same northing or latitude, and the third is two grid resolution units to the east and at the same northing or latitude. For C columns in the grid, the $(C+1)$th sample in the grid is located one grid resolution unit to the north, but on the same easting, or longitude, as the first sample in the grid.

4.3. Units

The units used in all measurements are SI metric units, both in the data representation and in the metadata. Vertical measurements shall be in meters; for projected grids, the horizontal units shall be meters, and for geographic grids, the horizontal units shall be signed decimal degrees. For geographic grids, positive latitude shall be north of the equator, and positive longitude shall be degrees east of Greenwich, measured with respect to the ellipsoid of rotation associated with the horizontal datum declared in the metadata. User-level code shall ensure that geographic coordinates are appropriately mapped into the range $[-180.0, 180.0]^{\circ}$ for longitude, and $[-90, 90]^{\circ}$ for latitude.

Note that nominal depth layers may be tagged in the metadata as 'Nominal_Depth_Feet', but this indicates solely that a sound speed of 4800 ft/s was used to make the nominal depth corrections, rather than 1500 m/s. The actual elevations shall still be recorded in SI metres.

The units of uncertainty shall be as defined by the metadata associated with the BAG. If the uncertainty can be interpreted as a variance, standard deviation or confidence interval on the elevation data, it shall be expressed as either meters or meters squared, as appropriate for the interpretation.

Units in the metadata shall follow the units used in the grids natively. The metadata shall contain sufficient information in the geo-referencing section to allow this distinction to be determined before the user-level code has to interpret any data in the grid.

Time shall be represented in seconds UTC with respect to the standard UNIX epoch of zero seconds, 1970-01-01/00:00:00.

4.4. Resolution and Precision of Data

All data in the BAG grid, metadata and tracking list shall be represented as IEEE-754 [8] floating point numbers without rounding or limitation of precision. Data that is fundamentally integer in nature (e.g., counts of elements) may be represented in integer format for compactness. Grid data shall have single precision (32-bit) representation; metadata shall have at least single precision representation, but may be more if required. All software attempting to manipulate BAGs shall at least preserve the precision of the input data.

All positioning and geo-referencing data within a BAG shall have at least millimeter resolution. All elevation information shall have at least millimeter resolution, although this should not be taken to mean that the fundamental precision of the data is better than the associated uncertainty measurement. All times shall have at least millisecond resolution.

5. The BAG API

5.1. Structure of the Source Tree

The source code for the BAG access library can be obtained as described in Section 7. The basic structure is outlined in Table 9. The BAG Application Programming Interface (API) is defined in the `api` sub-directory, with the primary interface defined in `bag.h`. User-level code should not use any of the deeper interface functions (i.e. those not declared for public consumption in `bag.h`) since they do not present a uniform reporting structure for errors and return codes. Special instructions for compilation and the structure of the library are in a `readme.txt` file in the top level directory. Other `readme.txt` files provide detailed information throughout the remainder of the source tree. Full details of the API structure are given in the `docs/api` sub-directory, constructed with `doxygen` [9] directly from the source code.

Directory		Description
api		BAG API files.
configdata		Configuration binary files, transformation and other geodetic data.
	ISO19139	Meta-data schemas and definitions.
docs		Documentation of the BAG file structure.
	api	`doxygen` documentation of API in HTML form.
examples		Example source files showing how to exercise the API.
	bagcreate	Create an example BAG given metadata in XML form.
	bagread	Read a BAG and write formatted ASCII output.
	excertlib	Sub-library to handle XML DSS certificates.
	gencert	Generate an XML certificate pair for the DSS.
	sample-data	Small example BAG files for testing.
	signcert	Sign an XML public key certificate for the DSS.
	signfile	Sign a BAG file using the DSS.
	verifycert	Verify the signature on a public key DSS certificate.
	verifyfile	Verify the signature of a BAG using the DSS.
extlibs		External libraries used by the BAG API.
	beecrypt	General cryptographic library used for the DSS.
	hasp	Hardware encryption token support library.
	HDF-5	Hierarchical Data Format support library, version 5.
	lib	Storage for built external libraries.
	szip	Scientific code ZIP library (for HDF-5).
	BAG_XML_LIB	Provides access to XML metadata
	xercesc	Comprehensive XML parser library for BAG metadata.
	zlib	ZIP library (for HDF-5).

Table 9: SOURCE TREE STRUCTURE FOR THE BAG API LIBRARY.

5.2. Basic Data Access

The BAG API supports a standard open/read-write/close process for dealing with BAG files, using `bagFileOpen()` and `bagFileClose()` to open/close existing files, and `bagFileCreate()` to create new files. When creating files, the user is responsible for filling out a `bagData` structure with the appropriate parameters and data (see `bag.h` for definitions) before calling `bagFileCreate()`; appropriate XML metadata is required to create a BAG file, `bagInitDefinitionsFromFile()`

can be used, or `bagInitDefinitionsFromBuffer()` can be used if the XML has already been read into memory. A convenience function, `bagInitDefinitionsFromBag()`, for use with pre-existing BAGs will also initialize the BAG definition from the BAG file's Metadata dataset.

The information required to access a BAG file is held in the `bagHandle` structure that is returned from `bagFileOpen()` or `bagFileCreate()`. This must be preserved throughout any process transaction with a BAG file. User level code cannot use `bagHandle` directly since it is opaqued in `bag_private.h`. However, access functions such as `bagGetDataPointer()` can be used to obtain any relevant information from the structure, such as a pointer to the data definition arrays, so that user-level code can access file-global definitions like the number of rows or columns in the data grids.

Once the file is open, data can be read either node by node using `bagReadNode()` or `bagReadNodeLL()` for projected and geographic grids, respectively (the type of grid can be found from the metadata), by row using `bagReadRow()`, within a sub-region using `bagReadRegion()` or as a full dataset using `bagReadDataset()`. The last three functions operate in node space, using row/column indices into the array rather than projected or geographic coordinates. Equivalently named calls (e.g., `bagWriteNode()`, `bagWriteNodeLL()`) are available to write data. Note that all data in the mandatory elements are single-precision floating point numbers, but the access calls use pointer-to-void formal parameters in order to opaque this restricted data type for future expansion.

The BAG structure is a uniform grid, defined by the geo-referencing point and a grid resolution in east and north directions. Therefore, no coordinates are required on a per-node basis since they may be computed implicitly from the row/column of the node in question. To assist in this, calls such as `bagReadNodePos()`, `bagReadRowPos()` or `bagReadDatasetPos()` augment the similarly named calls described previously by computing the positions of the rows and columns, which are returned in two linear arrays (one for vertical position of the rows, and one for the horizontal position of the columns) with respect to the grid's coordinate system. Note that this is the only recommended way of computing physical coordinates for nodes, and these positions cannot be computed subsequent to the read/write call.

With the addition of the optional dataset layer in Verion 1.1 the call `bagCreateOptionalDataset()`, was implemented to create the optional dataset with a surface type parameter as described in the BAG_SURFACE_PARAMS structure. Once the dataset(s) have been created calls to `bagWriteNode()`, `bagWriteRegion()`, and the like, can be used by providing the type parameter to identify the intended optional surface to write the data to the datasets. Read capability can be achieved by providing the matching type parameter to the function calls such as `bagReadNode()`, and `bagReadRegion()`, which are the same functions used for the reading of the elevation and uncertainty datasets. To assist in the determination of the contents of the BAG, the call `bagGetOptDatasets()` has been added to retrieve the number of optional datasets contained in the BAG and the surface type of those layers as defined in the BAG_SURFACE_PARAMS.

5.3. Metadata Access

XML metadata is treated as a simple binary stream of bytes. The XML stream can be read and written with `bagReadXMLStream()` and `bagWriteXMLStream()` respectively. When complete, the user code should call `bagFreeXMLMeta()` so that any dynamically allocated memory associated with the XML data parser is released to avoid memory leaks.

5.3.1 BAG XML Metadata Library

The purpose of the BAG XML Metadata library is to provide BAG software developers easy access to the individual components of the BAG metadata. The BAG XML Metadata library may be used to read and write individual, upper-level metadata components within the BAG XML metadata stream. The metadata library supplies data structures that correspond to the upper-level metadata components within the metadata XML schema. The library also supplies methods to "get" and "set" the individual upper-level metadata components as well as a method to create a valid XML instance of all the metadata components as a string. In addition, the library provides utility to methods to initialize data structures and print data structures. Accompanying the library is an HTML document that provides details of the data structures and the methods provided. Also accompanying the library are sample applications that illustrate use of the library.

The library does not support all possible scenarios that are valid per the metadata XML schema. Instead, the library provides support for navigation specific requirements. The following sections outline the support that is provided by the library. The mapping of library data structures to XML schema is detailed in Appendix A.

Each of the following sections contains graphics of the metadata XML schema, which has been altered to indicate the features supported by the BAG XML Metadata library. Each uppermost node in the metadata XML schema is detailed in a following section. Each section has one or more graphics depicting the items that the BAG XML Metadata Library supports. Within each graphic, the original schema documentation is unaltered. Documentation that begins with "Nav Spec" has been added to the original schema to indicate support provided by the BAG XML Metadata Library. Nav Spec is an abbreviation for "Navigation Specific." In addition, the number of allowed node occurrences has been altered to indicate the number that the library currently supports.

5.3.1.1 MD_Metadata

Following is a graphic depiction of the uppermost XML metadata nodes that the library supports, and the number of node occurrences that are supported by the BAG XML Metadata Library. Note that not all of the upper-level nodes supplied by the metadata schema are supported. For many of the supported nodes, only one occurrence is supported and that one occurrence is required.

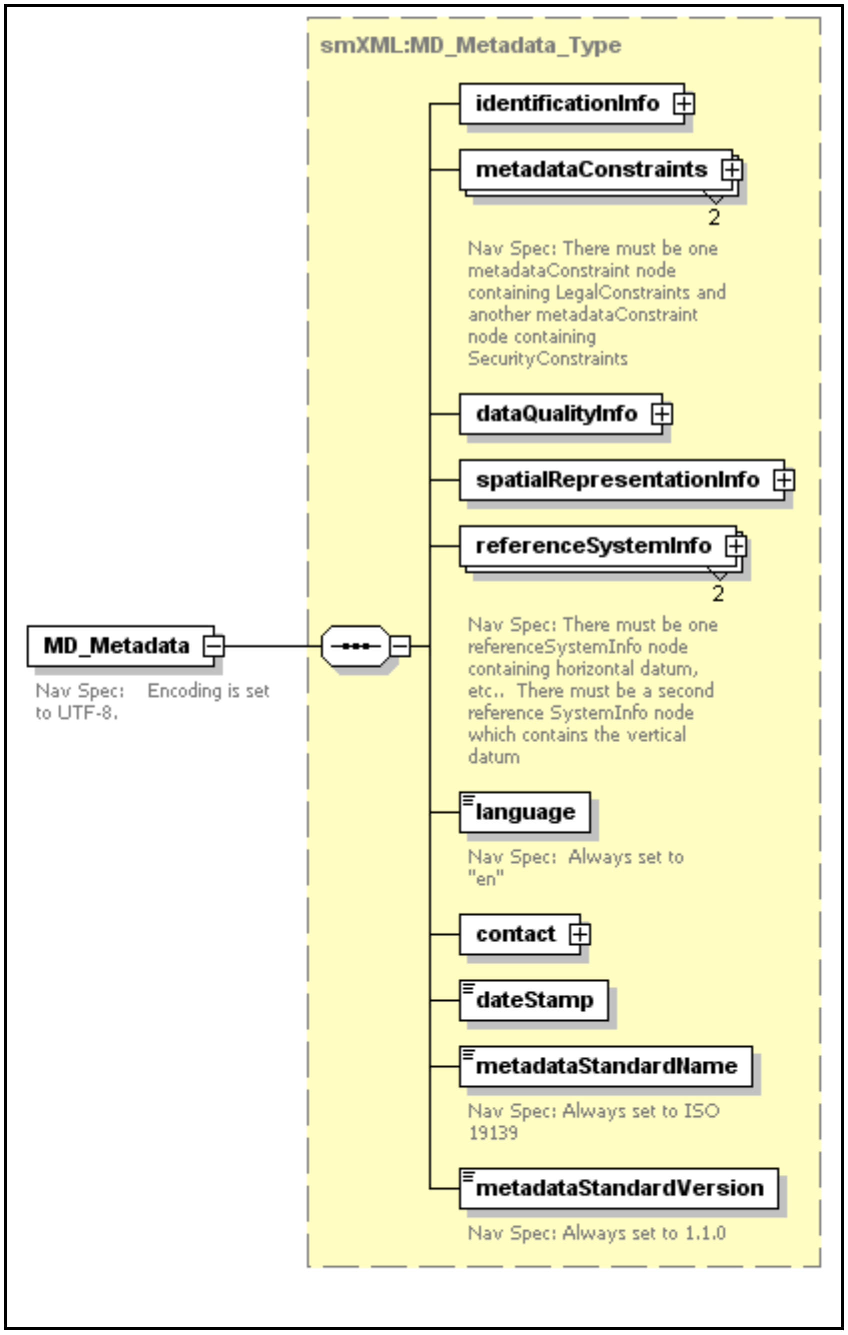

smXML:MD_Metadata_Type
identificationInfo
metadataConstraints
2
Nav Spec: There must be one metadataConstraint node containing LegalConstraints and another metadataConstraint node containing SecurityConstraints
dataQualityInfo
spatialRepresentationInfo
referenceSystemInfo
2
Nav Spec: There must be one referenceSystemInfo node containing horizontal datum, etc.. There must be a second reference SystemInfo node which contains the vertical datum
MD_Metadata
Nav Spec: Encoding is set to UTF-8.
language
Nav Spec: Always set to "en"
contact
dateStamp
metadataStandardName
Nav Spec: Always set to ISO 19139
metadataStandardVersion
Nav Spec: Always set to 1.1.0

Fig. 9 UPPERMOST NODES IN METADATA XML SCHEMA

5.3.1.2 identificationInfo

The metadata XML schema allows for one to an unbounded number of identificationInfo nodes. The BAG XML Metadata Library supports only one occurrence of this node. Note that an XML element named "depthCorrectionType" has been added. Multiple figures below depict the drill down into the identificationInfo node.

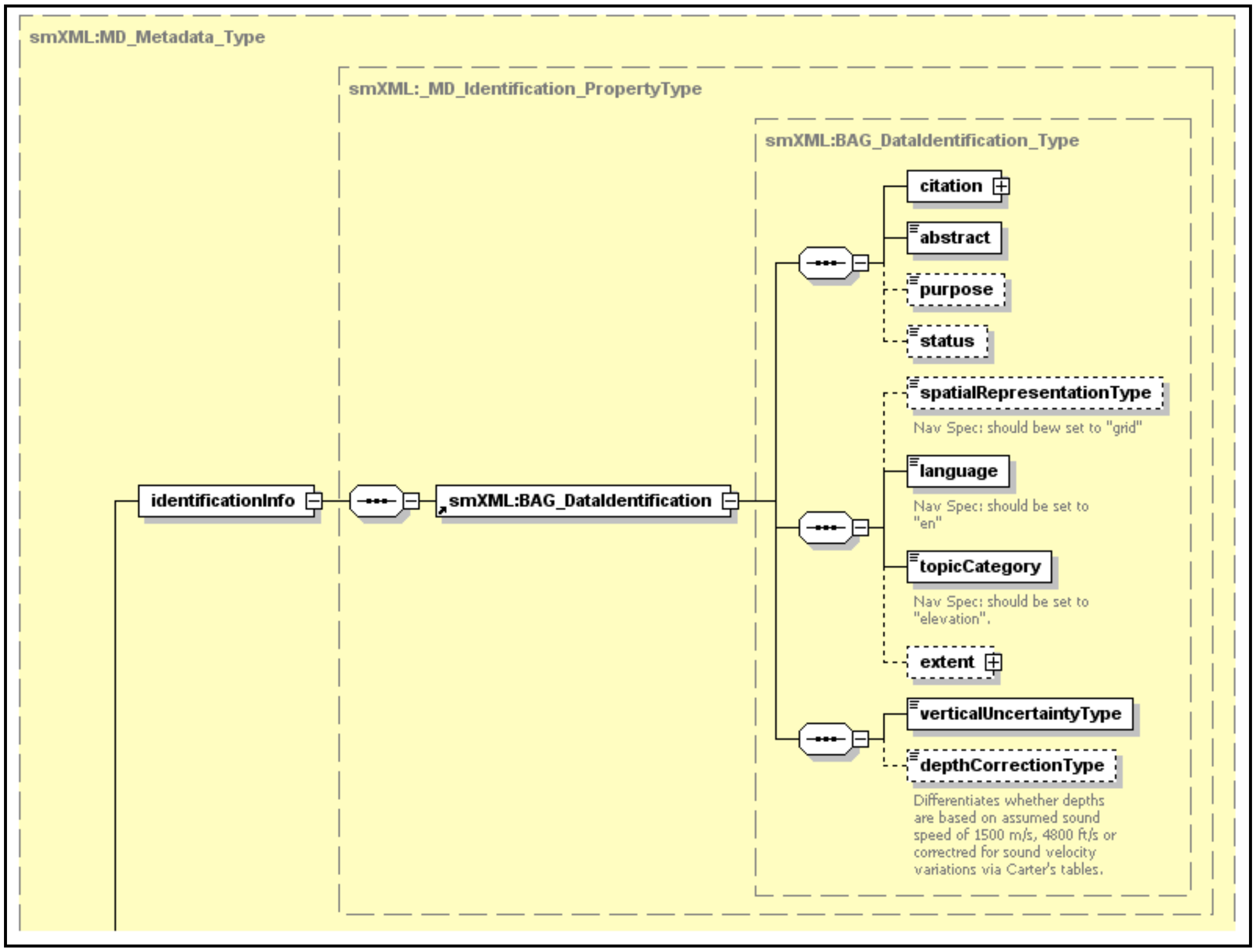

smXML:MD_Metadata_Type
smXML:_MD_Identification_PropertyType
smXML:BAG_DataIdentification_Type
citation
abstract
purpose
status
spatialRepresentationType
Nav Spec: should bew set to "grid"
language
Nav Spec: should be set to "en"
topicCategory
Nav Spec: should be set to "elevation".
extent
verticalUncertaintyType
depthCorrectionType
Differentiates whether depths are based on assumed sound speed of 1500 m/s, 4800 ft/s or corrected for sound velocity variations via Carter's tables.
identificationInfo
smXML:BAG_DataIdentification

Fig. 10 IDENTIFICATIONINFO NODE

The citation node is depicted below. Note that the library does not support all optional nodes listed in the metadata XML schema.

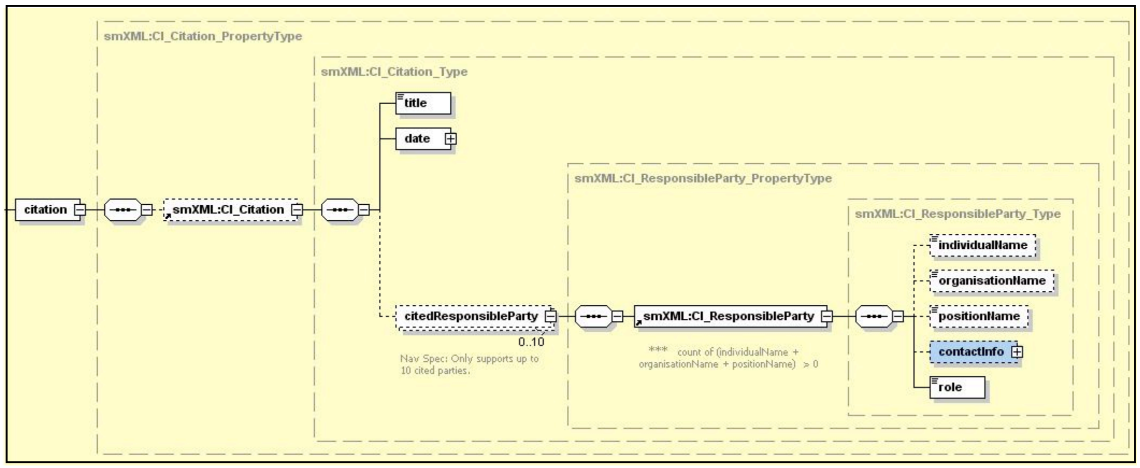

smXML:CI_Citation_PropertyType
smXML:CI_Citation_Type
title
date
citation
smXML:CI_Citation
smXML:CI_ResponsibleParty_PropertyType
smXML:CI_ResponsibleParty_Type
individualName
organisationName
citedResponsibleParty
smXML:CI_ResponsibleParty
positionName
0..10
Nav Spec: Only supports up to 10 cited parties.
*** count of (individualName + organisationName + positionName) > 0
contactInfo
role

Fig. 11 CITATION NODE

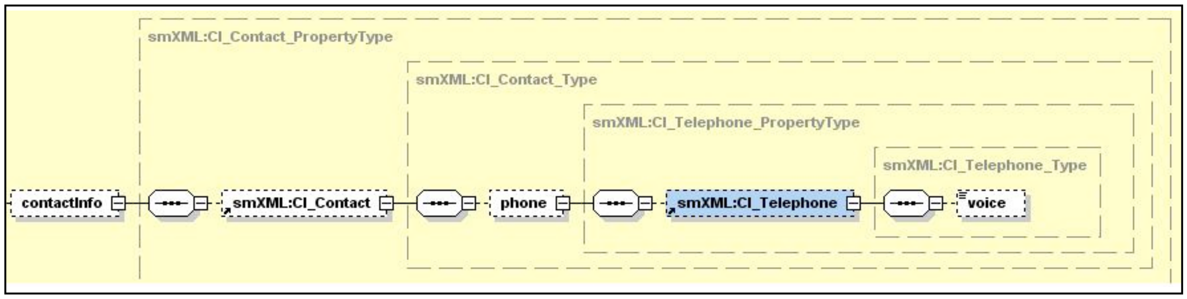


Fig. 12 CONTACTINFO NODE

The extent node is depicted below.

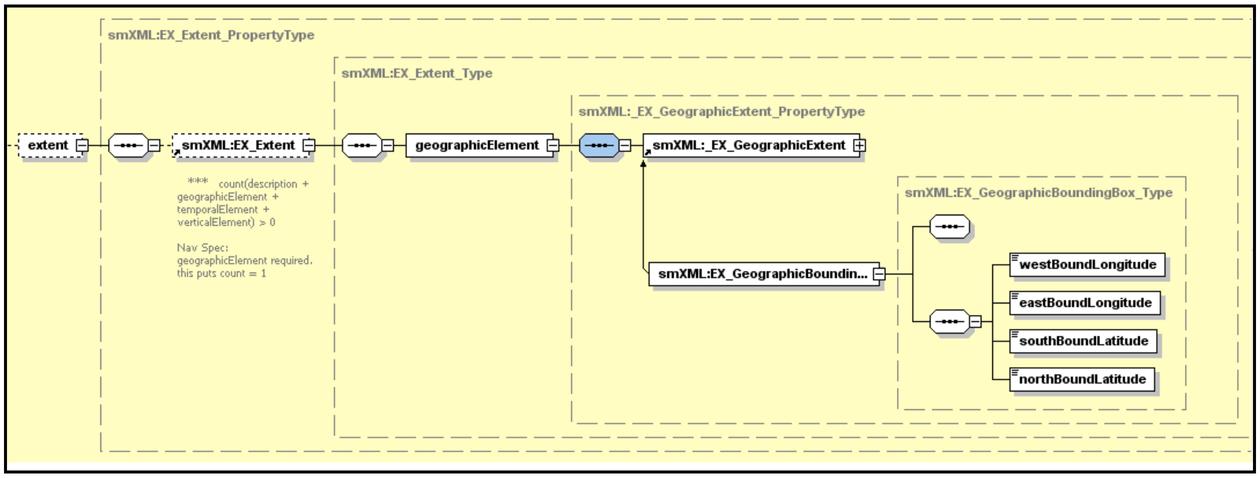

smXML:EX_Extent_PropertyType
smXML:EX_Extent_Type
smXML:_EX_GeographicExtent_PropertyType
extent
smXML:EX_Extent
geographicElement
smXML:_EX_GeographicExtent
*** count(description + geographicElement + temporalElement + verticalElement) > 0
Nav Spec: geographicElement required, this puts count = 1
smXML:EX_GeographicBoundingBox_Type
smXML:EX_GeographicBoundin...
westBoundLongitude
eastBoundLongitude
southBoundLatitude
northBoundLatitude

Fig. 13 EXTENT NODE

Following are the BAG XML Metadata Library structures that represent the above XML nodes. Mapping from the XML nodes to the library data structures is listed in Appendix A.

```
typedef struct
{
    NV_CHAR             title[100];
    NV_CHAR             date[20];
    NV_CHAR             dateType[20];
    RESPONSIBLE_PARTY   responsibleParties[MAX_CI_RESPONSIBLE_PARTIES];
    NV_CHAR             abstract[8000];
    NV_CHAR             purpose[100];
    NV_CHAR             status[100];
    NV_CHAR             spatialRepresentationType[100];
    NV_CHAR             language[100];
    NV_CHAR             topicCategory[100];
    NV_FLOAT64          westBoundingLongitude;
    NV_FLOAT64          eastBoundingLongitude;
    NV_FLOAT64          southBoundingLatitude;
    NV_FLOAT64          northBoundingLatitude;
    NV_CHAR             verticalUncertaintyType[40];
    NV_CHAR             depthCorrectionType[32];      /* Navigation Specific. */

} IDENTIFICATION_INFO;
```

Fig. 14 IDENTIFICATION_INFO LIBRARY STRUCTURE

```
typedef struct
{
    NV_CHAR individualName[100];
    NV_CHAR organisationName[100];
    NV_CHAR positionName[100];
    NV_CHAR phoneNumber[17];
    NV_CHAR role[100];

} RESPONSIBLE_PARTY;
```

Fig. 15 RESPONSIBLE_PARTY LIBRARY STRUCTURE

5.3.1.3 metaDataConstraints

The metadata XML schema allows for zero to an unbounded number of metadataConstraint nodes. The BAG XML Metadata Libarary supports two metadataConstraint nodes. There must be one metadataConstraint node containing LegalConstraints and another metadataConstraint node containing SecurityConstraints.

The metadataConstraints node is depicted below.

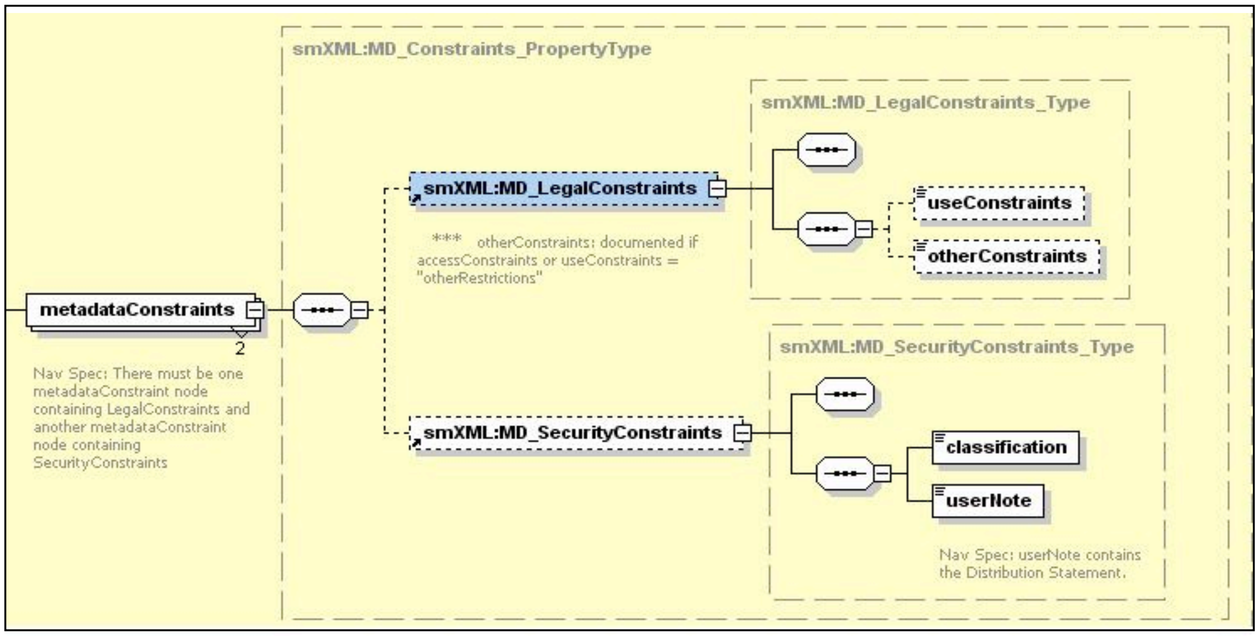

smXML:MD_Constraints_PropertyType
smXML:MD_LegalConstraints_Type
smXML:MD_LegalConstraints
useConstraints
otherConstraints
*** otherConstraints: documented if accessConstraints or useConstraints = "otherRestrictions"
metadataConstraints
2
Nav Spec: There must be one metadataConstraint node containing LegalConstraints and another metadataConstraint node containing SecurityConstraints
smXML:MD_SecurityConstraints_Type
smXML:MD_SecurityConstraints
classification
userNote
Nav Spec: userNote contains the Distribution Statement.

Fig. 16 METADATACONSTRAINTS NODE

Following are the library structures that represent the above nodes.

```
typedef struct
{
    NV_CHAR             useConstraints[40];
    NV_CHAR             otherConstraints[100];

} MD_LEGAL_CONSTRAINTS;
```

Fig. 17 `MD_LEGAL_CONSTRAINTS` LIBRARY STRUCTURE

```
typedef struct
{
    NV_CHAR            classification[40];
    NV_CHAR            userNote[4000]; /* Distribution,declass date,etc. */

} MD_SECURITY_CONSTRAINTS;
```

Fig. 18 `MD_LEGAL_CONSTRAINTS` LIBRARY STRUCTURE

5.3.1.4 dataQuality

The metadata XML schema allows for zero to an unbounded number of dataQuality nodes. The BAG XML Metadata Libarary supports only one occurrence of this node. Multiple graphics below depict the drill down into this node.

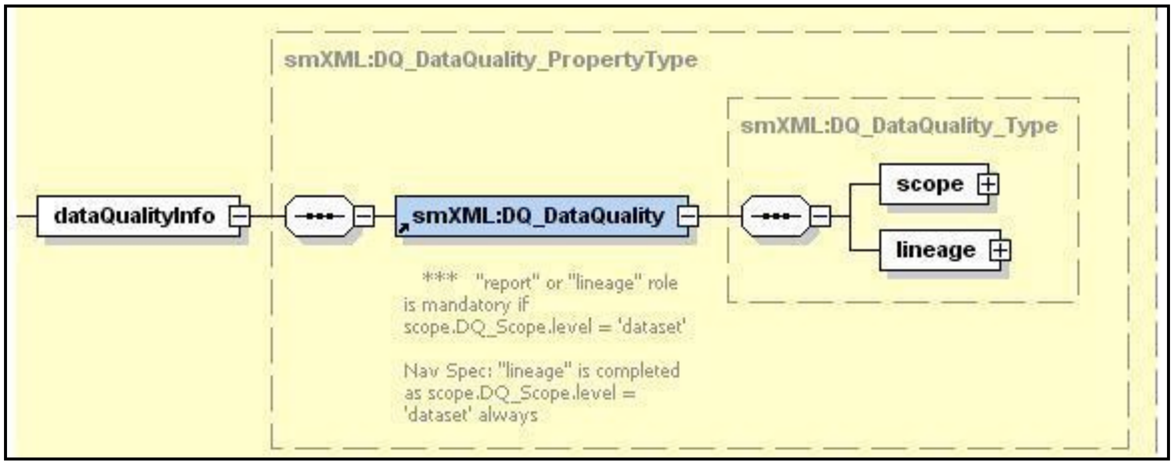

smXML:DQ_DataQuality_PropertyType
smXML:DQ_DataQuality_Type
dataQualityInfo
smXML:DQ_DataQuality
scope
lineage
*** "report" or "lineage" role is mandatory if scope.DQ_Scope.level = 'dataset'
Nav Spec: "lineage" is completed as scope.DQ_Scope.level = 'dataset' always

Fig. 19 DATAQUALITYINFO NODE

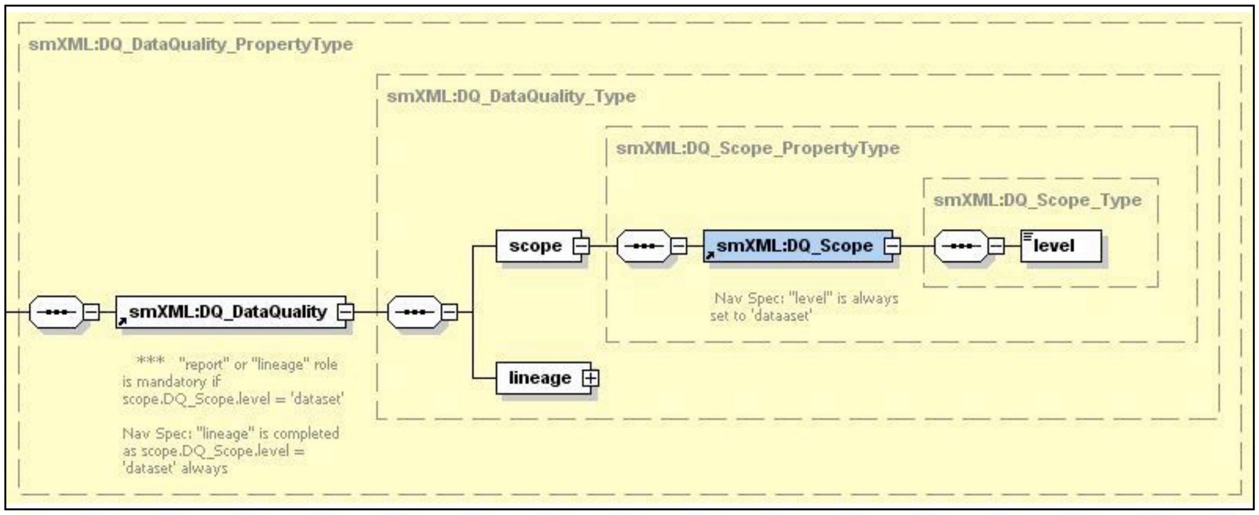

smXML:DQ_DataQuality_PropertyType
smXML:DQ_DataQuality_Type
smXML:DQ_Scope_PropertyType
smXML:DQ_Scope_Type
scope
smXML:DQ_Scope
level
Nav Spec: "level" is always set to 'dataaset'
smXML:DQ_DataQuality
lineage
*** "report" or "lineage" role is mandatory if scope.DQ_Scope.level = 'dataset'
Nav Spec: "lineage" is completed as scope.DQ_Scope.level = 'dataset' always

Fig. 20 SCOPE NODE

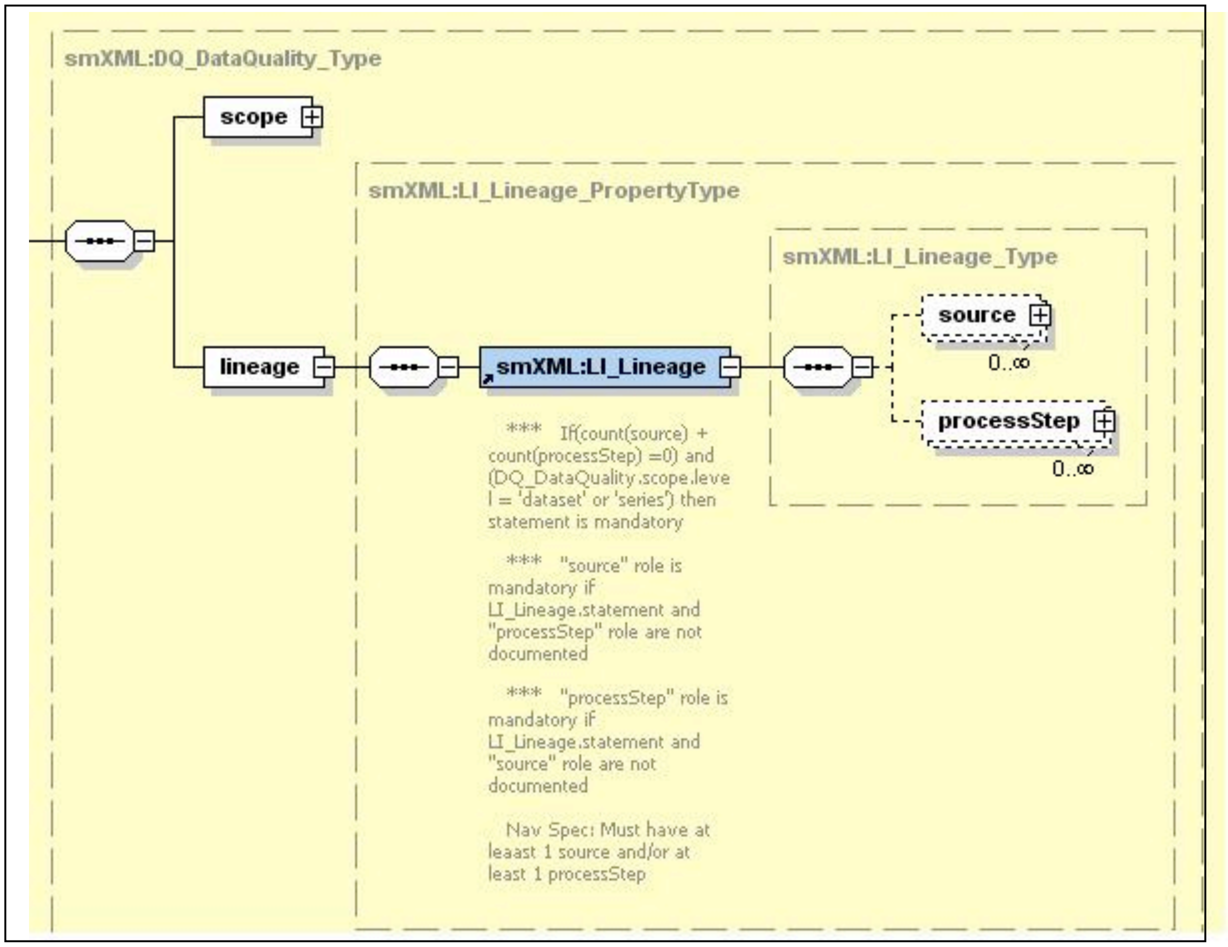

smXML:DQ_DataQuality_Type
scope
smXML:LI_Lineage_PropertyType
smXML:LI_Lineage_Type
source
0..∞
lineage
smXML:LI_Lineage
processStep
0..∞
*** If(count(source) + count(processStep) =0) and (DQ_DataQuality.scope.level = 'dataset' or 'series') then statement is mandatory
*** "source" role is mandatory if LI_Lineage.statement and "processStep" role are not documented
*** "processStep" role is mandatory if LI_Lineage.statement and "source" role are not documented
Nav Spec: Must have at leaast 1 source and/or at least 1 processStep

Fig. 21 LINEAGE NODE

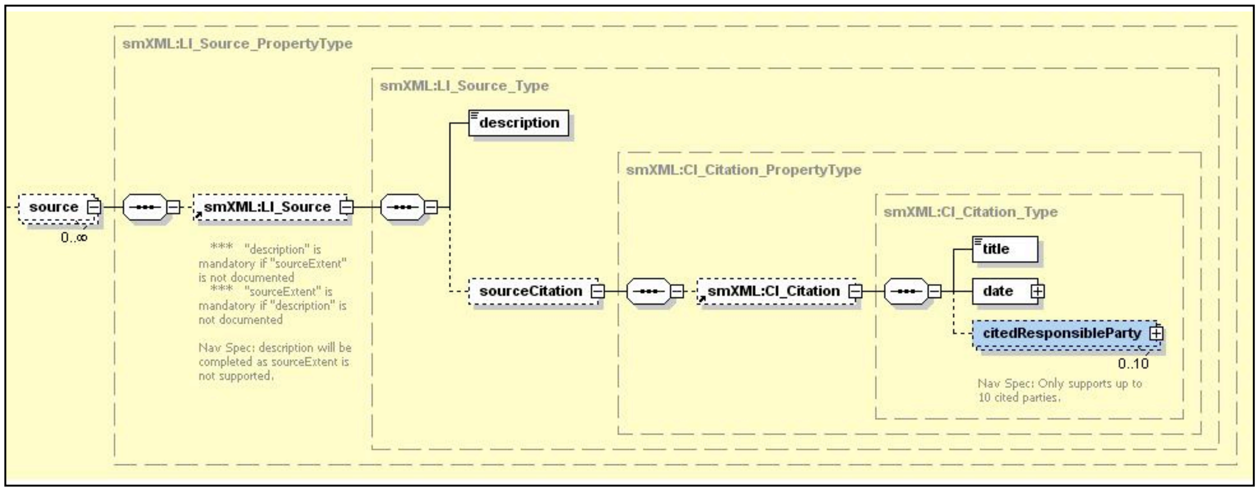

smXML:LI_Source_PropertyType
smXML:LI_Source_Type
description
source
0..∞
smXML:LI_Source
*** "description" is mandatory if "sourceExtent" is not documented
*** "sourceExtent" is mandatory if "description" is not documented
Nav Spec: description will be completed as sourceExtent is not supported.
smXML:CI_Citation_PropertyType
smXML:CI_Citation_Type
sourceCitation
smXML:CI_Citation
title
date
citedResponsibleParty
0..10
Nav Spec: Only supports up to 10 cited parties.

Fig. 22 SOURCE NODE

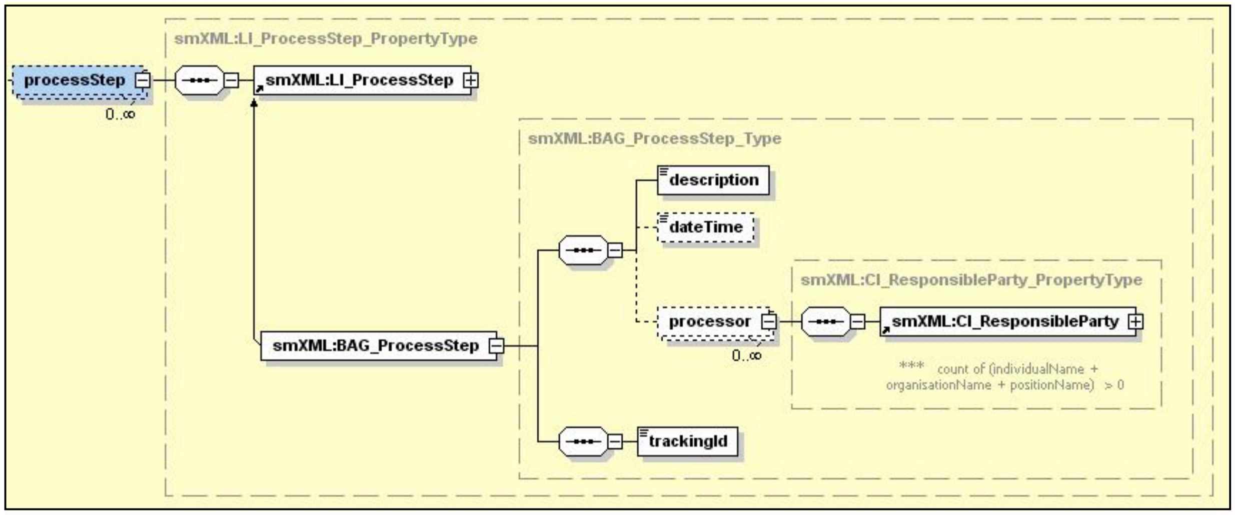

smXML:LI_ProcessStep_PropertyType
processStep
0..∞
smXML:LI_ProcessStep
smXML:BAG_ProcessStep_Type
description
dateTime
smXML:CI_ResponsibleParty_PropertyType
processor
0..∞
smXML:CI_ResponsibleParty
*** count of (individualName + organisationName + positionName) > 0
smXML:BAG_ProcessStep
trackingId

Fig. 23 PROCESS STEP NODE

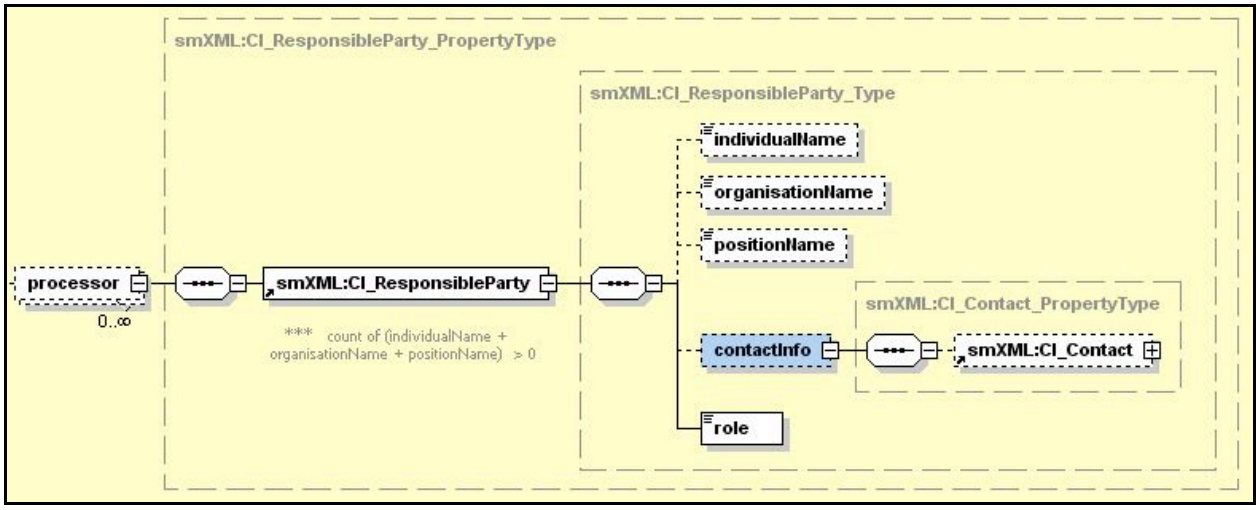

smXML:CI_ResponsibleParty_PropertyType
smXML:CI_ResponsibleParty_Type
individualName
organisationName
positionName
processor
0..∞
smXML:CI_ResponsibleParty
*** count of (individualName + organisationName + positionName) > 0
smXML:CI_Contact_PropertyType
contactInfo
smXML:CI_Contact
role

Fig. 24 PROCESSOR NODE

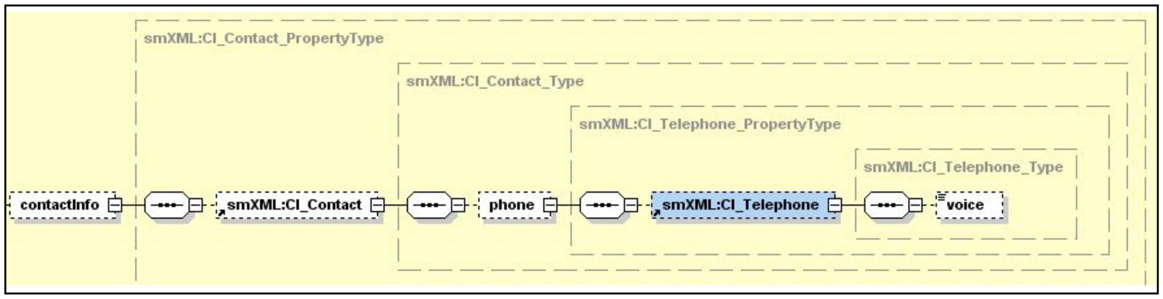


Fig. 25 SMXML:CI_CONTACT NODE

Following are the library structures that represent the above nodes. Mapping from the XML nodes the library data structures is listed in Appendix A.

```
typedef struct
{
    NV_CHAR             scope[100];
    NV_INT16            numberOfSources;
    SOURCE_INFO         *lineageSources;
    NV_INT16            numberOfProcessSteps;
    PROCESS_STEP_INFO   *lineageProcessSteps;

} DATA_QUALITY_INFO;
```

Fig. 26 DATA_QUALITY_INFO LIBRARY STRUCTURE

```
typedef struct
{
    NV_CHAR             description[200];
    NV_CHAR             title[100];
    NV_CHAR             date[20];
    NV_CHAR             dateType[20];
    RESPONSIBLE_PARTY   responsibleParties[MAX_CI_RESPONSIBLE_PARTIES];

} SOURCE_INFO;
```

Fig. 27 SOURCE_INFO LIBRARY STRUCTURE

```
typedef struct
{
    NV_CHAR             description[200];
    NV_CHAR             dateTime[21];
    RESPONSIBLE_PARTY   processors[MAX_CI_RESPONSIBLE_PARTIES];
    NV_CHAR             trackingId[5];

} PROCESS_STEP_INFO;
```

Fig. 28 PROCESS_STEP_INFO LIBRARY STRUCTURE

```
typedef struct
{
  NV_CHAR individualName[100];
  NV_CHAR organisationName[100];
  NV_CHAR positionName[100];
  NV_CHAR phoneNumber[17];
  NV_CHAR role[100];

} RESPONSIBLE_PARTY;
```

Fig. 29 RESPONSIBLE_PARTY LIBRARY STRUCTURE

5.3.1.5 spatialRepresentationInfo

The metadata XML schema allows for one to an unbounded number of spatialRepresentationInfo nodes. BAG XML Metadata Library supports having only one occurrence of this node.

Multiple graphics below depict the drill down into this node. The graphics indicate the elements and number of occurrences supported by the library.

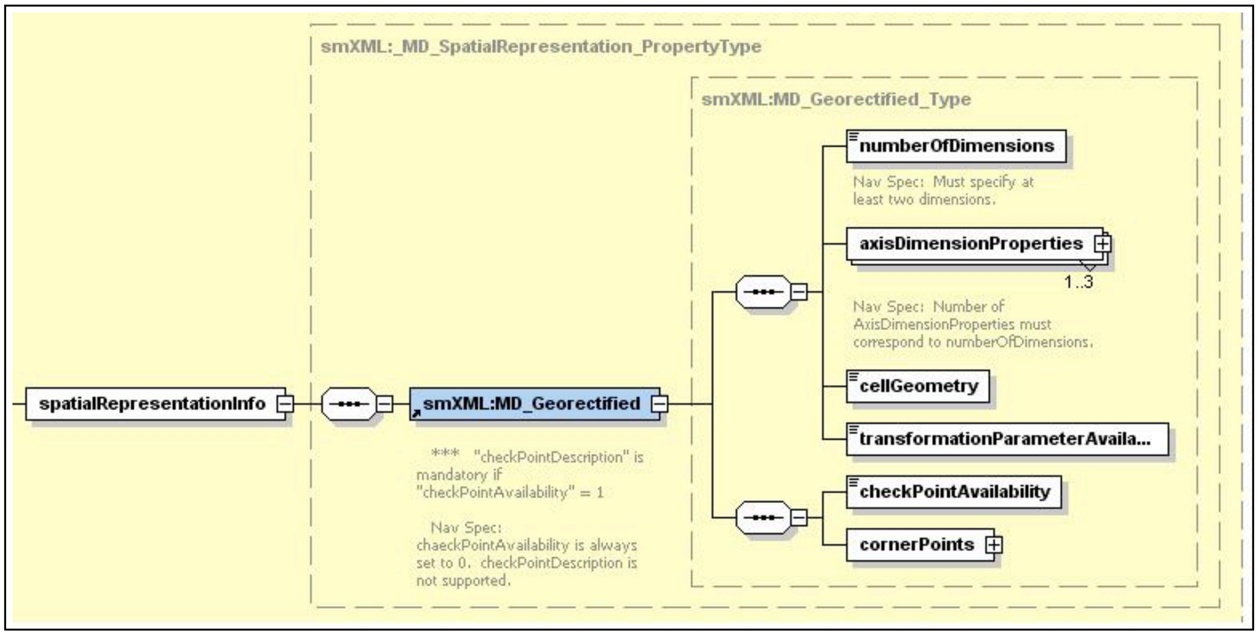

smXML:_MD_SpatialRepresentation_PropertyType
smXML:MD_Georectified_Type
numberOfDimensions
Nav Spec: Must specify at least two dimensions.
axisDimensionProperties
1..3
Nav Spec: Number of AxisDimensionProperties must correspond to numberOfDimensions.
cellGeometry
transformationParameterAvaila...
checkPointAvailability
cornerPoints
spatialRepresentationInfo
smXML:MD_Georectified
*** "checkPointDescription" is mandatory if "checkPointAvailability" = 1
Nav Spec: chaeckPointAvailability is always set to 0. checkPointDescription is not supported.

Fig. 30 SPATIALREPRESENTATIONINFO NODE

Note that the BAG XML Metadata Libarary constrains the axisDimensionProperties so that only three dimensions are supported.

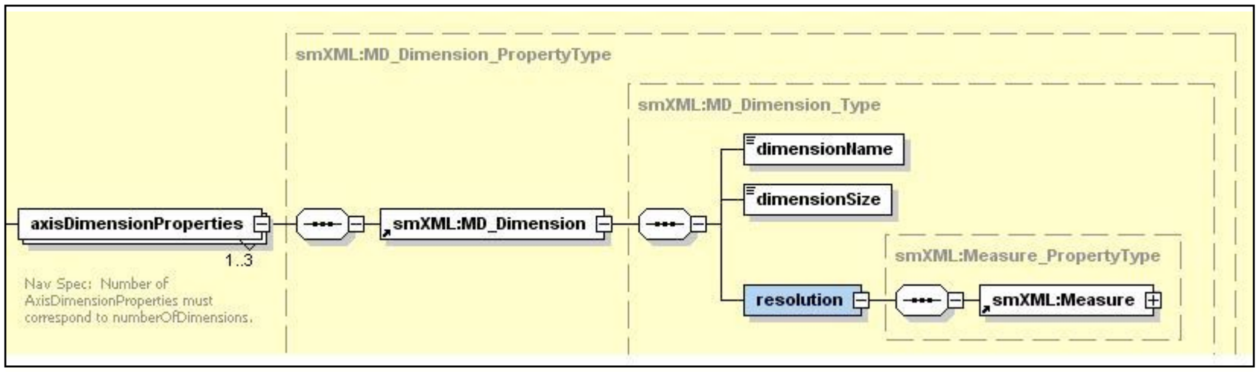

smXML:MD_Dimension_PropertyType
smXML:MD_Dimension_Type
dimensionName
dimensionSize
smXML:Measure_PropertyType
axisDimensionProperties
1..3
smXML:MD_Dimension
resolution
smXML:Measure
Nav Spec: Number of AxisDimensionProperties must correspond to numberOfDimensions.

Fig. 31 AXISDIMENSIONPROPERTIES NODE

In the resolution node, uom_r node is written but no value is assigned.

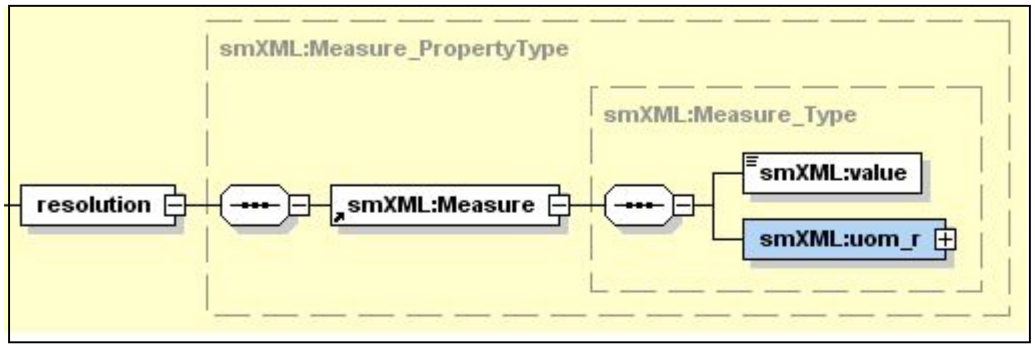
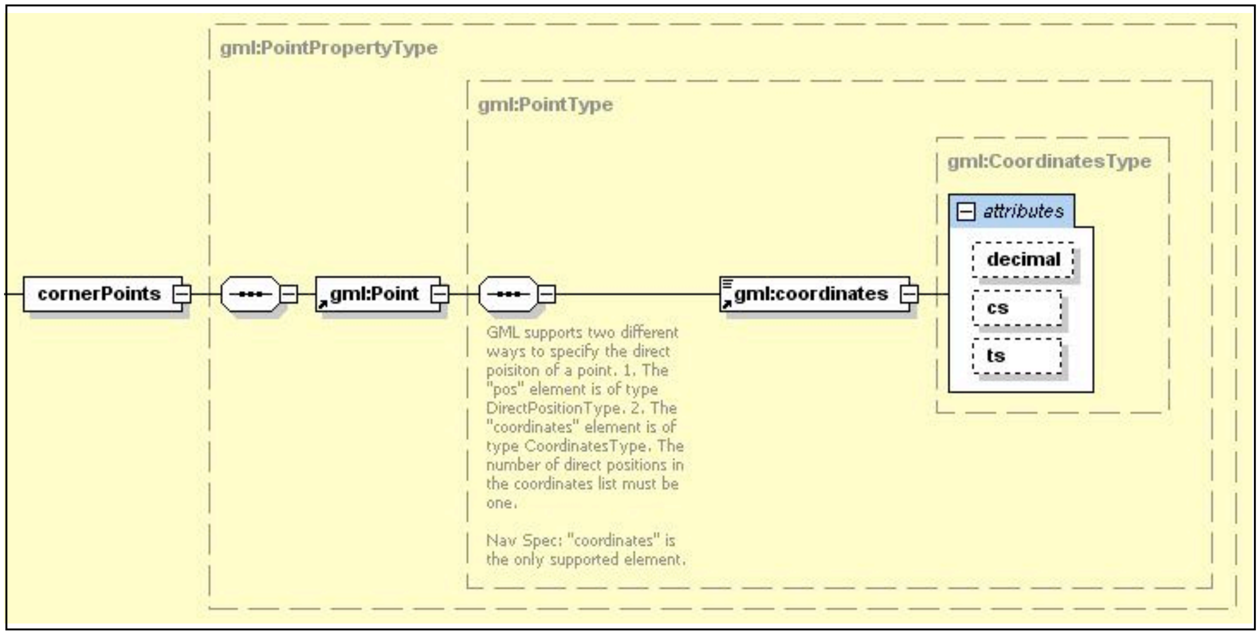


Fig. 32 RESOLUTION NODE



GML supports two different ways to specify the direct poisiton of a point. 1. The "pos" element is of type DirectPositionType. 2. The "coordinates" element is of type CoordinatesType. The number of direct positions in the coordinates list must be one.

Nav Spec: "coordinates" is the only supported element.

Fig. 33 CORNERPOINTS NODE

Following is the library structure that represents the above XML nodes.

```
typedef struct
{
    NV_INT16          numberOfDimensions;
    NV_CHAR           dimensionName[3][20];
    NV_INT32          dimensionSize[3];
    NV_FLOAT64        resolutionValue[3];
    NV_CHAR           cellGeometry[10];
    NV_CHAR           transformationParameterAvailability[6];
    NV_CHAR           checkPointAvailability[2];
    NV_FLOAT64        llCornerX;
    NV_FLOAT64        llCornerY;
    NV_FLOAT64        urCornerX;
    NV_FLOAT64        urCornerY;

} SPATIAL_REPRESENTATION_INFO;
```

Fig. 34 SPATIALREPRESENTATIONINFO LIBRARY STRUCTURE

5.3.1.6 referenceSystemInfo

The metadata XML schema allows for zero to an unbounded number of referenceSystemInfo nodes.

BAG XML Metadata Library supports only two occurrences of this node. The first node contains horizontal datum, etc. The second node contains only the vertical datum information. Multiple graphics below depict the drill down into the node structure.

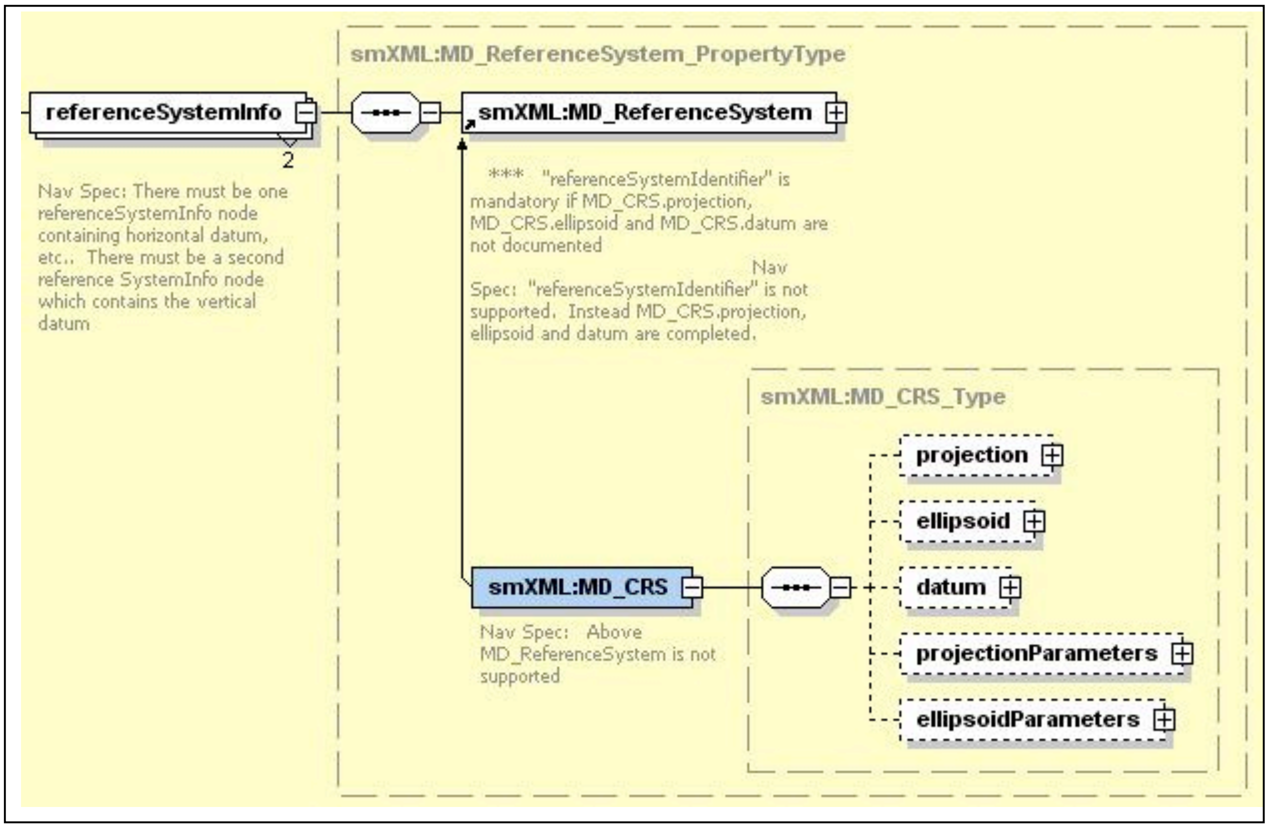

smXML:MD_ReferenceSystem_PropertyType
referenceSystemInfo
2
Nav Spec: There must be one referenceSystemInfo node containing horizontal datum, etc.. There must be a second reference SystemInfo node which contains the vertical datum
smXML:MD_ReferenceSystem
*** "referenceSystemIdentifier" is mandatory if MD_CRS.projection, MD_CRS.ellipsoid and MD_CRS.datum are not documented
Nav Spec: "referenceSystemIdentifier" is not supported. Instead MD_CRS.projection, ellipsoid and datum are completed.
smXML:MD_CRS_Type
projection
ellipsoid
smXML:MD_CRS
datum
Nav Spec: Above MD_ReferenceSystem is not supported
projectionParameters
ellipsoidParameters

Fig. 35 REFERENCESYSTEMINFO NODE

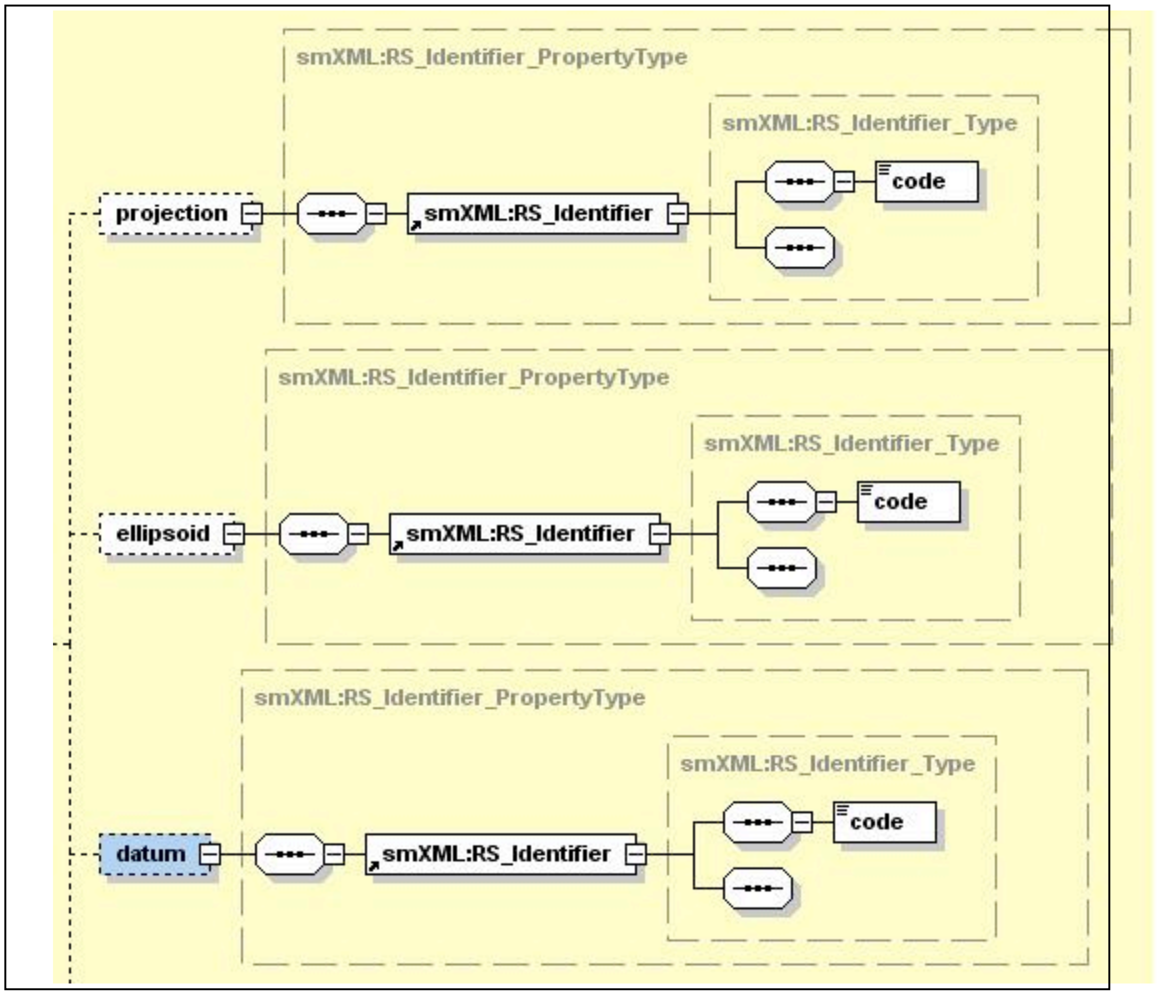
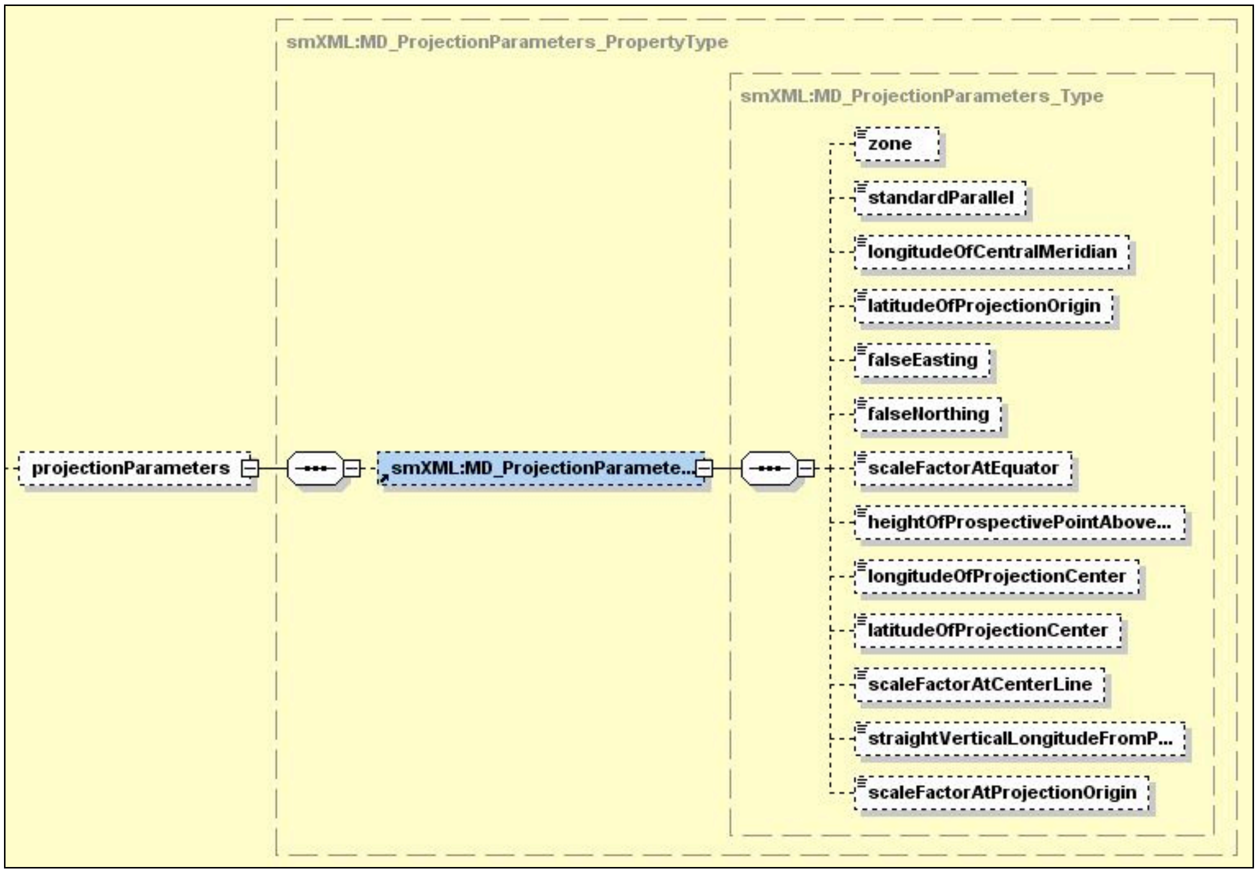
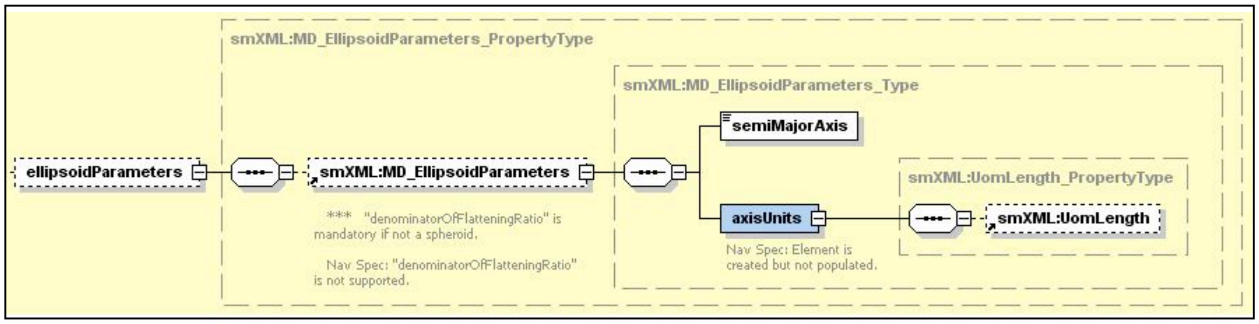


Fig. 36 PROJECTION, ELLIPSOID AND HORIZONTAL DATUM NODES



smXML:MD_ProjectionParameters_PropertyType
smXML:MD_ProjectionParameters_Type
projectionParameters
smXML:MD_ProjectionParamete...
zone
standardParallel
longitudeOfCentralMeridian
latitudeOfProjectionOrigin
falseEasting
falseNorthing
scaleFactorAtEquator
heightOfProspectivePointAbove...
longitudeOfProjectionCenter
latitudeOfProjectionCenter
scaleFactorAtCenterLine
straightVerticalLongitudeFromP...
scaleFactorAtProjectionOrigin

Fig. 37 PROJECTIONPARAMETERS NODE



smXML:MD_EllipsoidParameters_PropertyType
smXML:MD_EllipsoidParameters_Type
ellipsoidParameters
smXML:MD_EllipsoidParameters
*** "denominatorOfFlatteningRatio" is mandatory if not a spheroid.
Nav Spec: "denominatorOfFlatteningRatio" is not supported.
semiMajorAxis
axisUnits
Nav Spec: Element is created but not populated.
smXML:UomLength_PropertyType
smXML:UomLength

Fig. 38 ELLIPSOIDPARAMETERS NODE

Following are the library structures that represent the above nodes. Mapping from the XML nodes to the library data structures is provided in Appendix A.

```
typedef struct
{

    NV_CHAR            projection[100];
    NV_CHAR            ellipsoid[100];
    NV_CHAR            horizontalDatum[100];
    NV_INT16           zone;
    NV_FLOAT64         standardParallel;
    NV_FLOAT64         longitudeOfCentralMeridian;
    NV_FLOAT64         latitudeOfProjectionOrigin;
    NV_FLOAT64         falseEasting;
    NV_FLOAT64         falseNorthing;
    NV_FLOAT64         scaleFactorAtEquator;
    NV_FLOAT64         heightOfProspectivePointAboveSurface;
    NV_FLOAT64         longitudeOfProjectionCenter;
    NV_FLOAT64         latitudeOfProjectionCenter;
    NV_FLOAT64         scaleFactorAtCenterLine;
    NV_FLOAT64         straightVerticalLongitudeFromPole;
    NV_FLOAT64         scaleFactorAtProjectionOrigin;
    NV_FLOAT64         semiMajorAxis;
    NV_CHAR            verticalDatum[100];

} REFERENCE_SYSTEM_INFO;
```

Fig. 39 REFERENCE_SYSTEM_INFO LIBRARY STRUCTURE

The group of fields from the field named projection through the field named semiMajorAxis represent the contents of the first referenceSystemInfo node. The field named verticalDatum represents the contents of the second referenceSystemInfo node .

5.3.1.7 language

Optional in the metadata XML schema. The BAG XML Metadata Library requires this node. There is no library data structure that represents this metadata. Developers should use a character pointer to represent this metadata.

5.3.1.8 contact

The metadata XML schema allows for one to an unbounded number of contact nodes. The BAG XML Metadata Library supports only one occurrence of the contact node.

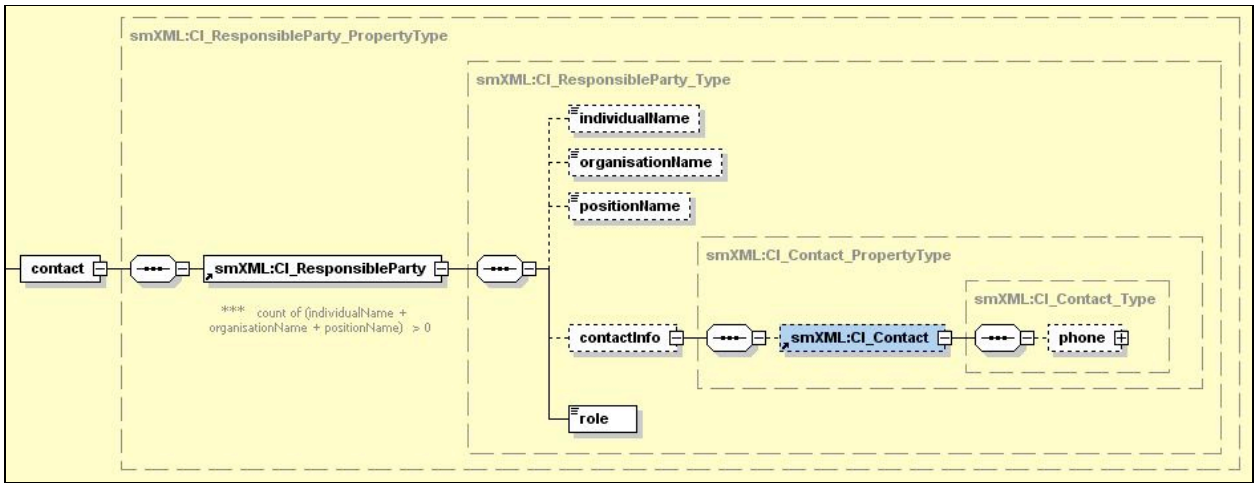

smXML:CI_ResponsibleParty_PropertyType
smXML:CI_ResponsibleParty_Type
individualName
organisationName
positionName
contact
smXML:CI_ResponsibleParty
*** count of (individualName + organisationName + positionName) > 0
smXML:CI_Contact_PropertyType
smXML:CI_Contact_Type
contactInfo
smXML:CI_Contact
phone
role

Fig. 40 CONTACT NODE

Following is the library structure that represents the above XML node.

```
typedef struct
{
    NV_CHAR individualName[100];
    NV_CHAR organisationName[100];
    NV_CHAR positionName[100];
    NV_CHAR phoneNumber[17];
    NV_CHAR role[100];

} RESPONSIBLE_PARTY;
```

Fig. 41 RESPONSIBLE_PARTY LIBRARY STRUCTURE

5.3.1.9 dateStamp

The BAG XML Metadata Library requires this node. There is no library data structure that represents this metadata. A developer must declare a string variable to represent this structure.

5.3.1.10 metadataStandardName

The metadataStandardName element is optional in metadata XML schema. The BAG XML Metadata Library requires this node. There is no library data structure that represents this metadata and the developer need not create one. The XML node is automatically populated by the metadata library when an XML string is created.

5.3.1.11 metadataStandardVersion

The metadataStandardVersion element is optional in metadata XML schema. The BAG XML Metadata Library requires this node. There is no library data structure that represents this metadata and the developer need not create one. The XML node is automatically populated by the metadata library when an XML string is created.

5.4. Tracking List Access

The tracking list component of the BAG file is accessed via direct calls. The number of elements in the list can be read with `bagTrackingListLength()`, and individual nodes in the list may be obtained using `bagReadTrackingListIndex()` using linear indexing into the list. Multiple tracking list items can be read at a time according to a number of different criteria:

`bagReadTrackingListNode()` returns all of the items associated with a particular grid node, `bagReadTrackingListCode()` returns all items which are tagged with a particular reason code, and `bagReadTrackingListSeries()` returns all items which are tagged with the same metadata series number (i.e., which were all generated with one metadata lineage entry). Similarly named routines to write tracking list entries are also included. If required, the nodes of the tracking list can be sorted according to any of the criteria above using routines such as `bagSortTrackingListByNode()`, `bagSortTrackingListBySeries()`, etc.

5.5. Digital Signatures

The Digital Signature Scheme for the BAG file is briefly defined in Section 2.9, and in more detail in the DSS whitepaper [6]. Key pairs are generated with `bagGenerateKeyPair()`, message digests are computed and signed with `bagComputeMessageDigest()` and `bagSignMessageDigest()` respectively, and file signatures can be computed directly using `bagComputeFileSignature()` if the message digest is not required separately.

Certification blocks are read, written and verified by `bagReadCertification()`, `bagWriteCertification()` and `bagVerifyCertification()` respectively. These routines are capable of silently creating a new certificate block at the end of the BAG if one is not present on write.

As convenience for the user who does not want to get into the details of the DSS, the `bagSignFile()` and `bagVerifyFile()` routines are provided to execute all of the stages required to complete signature and verification of a file, respectively. Similarly, the `bagConvertCryptoFormat()` routine can be used to convert signatures, digests or keys into ASCII format so that user-level code can write the data to suitable output files as required. It is the user's responsibility to ensure that secret keys are kept appropriately secret. An example of how to handle this is provided by the `excertlib` project in directory `examples/excertlib/excertlib.c`.

5.6. Error Codes and Reporting

All routines from `bag.h` return error codes from the `bagError` enumerated type, which is split into sections corresponding to the components of the library. Human-readable errors messages are available by passing the error code as an argument to `bagGetErrorString()`.

6. BAG Architecture Review Board

6.1. Intent of the ARB

The BAG Architecture Review Board (ARB) is intended to act as a vendor neutral central clearing house for all activities associated with the BAG format, API and access library. The scope of the ARB is to include:

- Organization of new releases of the library.
- Review of Request for Engineering (RFE) on the API or format, including new API calls, extensions to existing parts of the format, or addition of new HDF groups.
- Resolution of Request for Fix (RFF) on the library to address any bugs reported.
- Direction of the future development of the format, API and/or library.
- Administration of the project's website, SubVersion revision control system, documentation and archives.

6.2. Composition and Administration

The ARB shall consist of at least three people, and preferably no more than five. The initial membership of the ARB shall be nominated from within the developer group responsible for the initial development of the library and FSD. New members may be co-opted from time to time as required by a simple majority of the current members of the ARB. Membership of the ARB shall be limited to active developers of the BAG source code library. New members may be nominated by the general public at any time by sending e-mail to the development list `navsurf_dev@ccom.unh.edu`.

RFEs and RFFs shall be accepted from any source when sent by e-mail to `navsurf_dev@ccom.unh.edu` as detailed in Section 8. The ARB shall review the information in a timely manner, and accept or reject the request based on a simple majority of the members. The results of the review shall be communicated to the donator on a best-effort basis, and may be incorporated into intermediate or full releases of the BAG library at any point thereafter.

The ARB shall conduct its business primarily by e-mail, but may from time to time sponsor physical meetings as required by current or pending RFEs and RFFs, or any other business as may be required. Meetings of this type shall be advertised on the project's website at least two weeks in advance of the meeting date, and by an e-mail to `navsurf_general@ccom.unh.edu`. Whenever possible, the ARB shall schedule any meetings to take advantage of extant events (e.g., common-attendance hydrographic conferences) in order to minimize costs associated with ONS work.

7. Revision Control and Code Availability

Baselines of the BAG source code library, external libraries, and BAG documentation are revision controlled using a Subversion repository hosted by the Center for Coastal and Ocean Mapping/Joint Hydrographic Center (CCOM/JHC) at the University of New Hampshire (UNH). Each baseline release is referenced by a Subversion release tag that includes a numeric value corresponding to the release version number. Access to these archives is granted on a case by case basis to active developers. Requests for access to the Subversion archives can be made via email to the development list (navsurf_dev@ccom.unh.edu).

The current release is available for general download from the ONSWG web site at: http://www.opennavsurf.org/download.html. The release distribution includes all source code required to build the BAG access library and the external library dependencies. The ONSWG web site provides contact information and a description of how to request being added to one or more of the BAG email lists. In addition, the web site includes a wealth of historical background information from meeting notes, publications, and presentations.

The BAG structure is taken to be as defined by the Format Specification Document (FSD) and the corresponding baseline release of the source code. In the case of a conflict between the FSD and the source code, precedence is given to the source code as the reference structure definition.

Section 8 details the process for reporting deficiencies identified by the user community, and for requesting extensions or enhancements to the format. Feedback and contributions from the community to improve the BAG format are encouraged. Feedback can take the form of issue reports and/or provision of implementations of actual changes to either the BAG software library or the BAG documentation.

8. How to Apply for an Extension/Bug Fix

This FSD is intended to be a living document, evolving as the requirements for the BAG format change. Over time, it is expected that extensions to the HDF groups in the BAG will be required, and new elements of other groups might be required. This section describes how to apply for an extension or bug fix.

8.1. Nomination Process

Any requests for extension shall be considered by the BAG Architecture Review Board (see Section 6) as a group. All communication shall be by e-mail only, using the `navsurf_dev@ccom.unh.edu` address. Originators should include details appropriate to their request as described below, and be ready to answer any subsequent questions that might be required.

A 'receipt notice' e-mail shall be returned to the originator immediately, and a reply to the request shall be returned as quickly as possible. The decision making process shall be as defined in Section 6.

8.2. Request for an Extension HDF Group

Requests for an additional HDF group to be added to the base structure of the BAG must be accompanied by a full description of the data structure to be encoded. The request must be accompanied by a supporting document, e.g., an academic paper, user manual with appropriate details or a URI, and by preference code to read/write the data format. If the location of the section within the BAG structure is important, a recommendation for location may also be submitted.

The submission format may be plain text, Adobe PDF, or Microsoft Word. Other formats may be supported; please check with the BAG-ARB before sending however.

Since the FSD and the BAG format are open source, it is very important that the submission must be able to be published. This includes the source code submitted in support of the request. By sending the request to the group, the submitter explicitly agrees that:

- They are the owner of any intellectual property associated with the information in the request, and/or have the appropriate authority to transfer the associated intellectual property.
- The information in the request is not covered by any restrictions (e.g., security constraints, commercial secrets) that would prevent it from being used in the Open Navigation Surface project.
- There are no limitations on the publication, dissemination or other transmission of the data structure.
- Any source code provided may be used, adapted, or otherwise transformed for use in the source code base of the Open Navigation Surface project, including re-distribution of the code through any means in which the source code is generally made available.

Any requests that do not meet these requirements will be returned to the submitter, with comments as to cause, without further consideration by the BAG-ARB. The BAG-ARB, at their discretion, may request, in writing, confirmation of any or all of the above terms, or any others as may seem appropriate at the time, from the originator of the request. Provision of this confirmation shall be a mandatory condition for acceptance and adoption of the request.

8.3. Request for an Extension to an Existing Group

Extensions to an existing group, for example adding a new ellipsoid or datum definition to the meta-data, or another element to an existing group-specific meta-data attribute, may be submitted in the same way as for an extension HDF group (Section 8.2), including the requirements for free distribution and source code reuse. In addition, requests must contain a strong rationale for why the addition should be adopted.

8.4. Request for a Bug Fix

Requests for a Bug Fix should include a full description of the problem, with as much information about causes and conditions as possible, including the revision of the BAG library being used, and the version of the BAG format being constructed.   If available, details about the machine's system architecture, platform, or operating system would also be beneficial to the BAG-ARG for assessing the requested bug fix.

A suitable test case should also be included, if possible, that allows the problem to be exercised. Do not send binary data attachments to the e-mail list in the first instance, since this list is distributed widely. If required, a member of the BAG-ARB will request supporting data by other means.

If a fix for the problem is known, it should be included with the initial submission. Please note that the conditions on distribution, adaptation and re-use of source code in Section 8.2 apply to any source code submitted as a potential fix.

Glossary of Acronyms

Acronym	Definition
AES	Advanced Encryption Standard
API	Application Programming Interface
BAG	Bathymetric Attributed Grid
BAG-ARB	Bathymetric Attributed Grid Architecture Review Board
DS	Digital Signature
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
FSD	File Specification Document
HDF-5	Hierarchical Data Format, Version 5
IEEE	Institute of Electrical and Electronic Engineers, Inc.
NS	Navigation Surface [2,3]
ONS	Open Navigation Surface
ONSWG	Open Navigation Surface Working Group
PK	Public Key
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
SI	International System of Units
SK	Secret Key
URI	Universal Resource Indicator
XML	eXtensible Markup Language

Table 10: GLOSSARY OF ACRONYMS

References

[1] Calder, B. R., R. T. Brennan, J. D. Case, J. S. Byrne, B. Lamey, B. Gallagher, D. Fabre, R. W. Ladner, F. Moggert, M. Paton (2005). *The Open Navigation Surface Project*. Int. Hydro. Review, 6(2), pp.1-10.

[2] Smith, S. M. (2003). *The Navigation Surface: A Multipurpose Bathymetric Database*, Masters Thesis, Center for Coastal and Ocean Mapping & Joint Hydrographic Center, University of New Hampshire.

[3] Smith, S. M., L. Alexander, & A. A. Armstrong. (2002). *The Navigation Surface: A New Database Approach to Creating Multiple Products from High-Density Surveys*, Int. Hydro. Review, Vol. **3**, No. 2, pp. 12-26.

[4] Raymond, E. A., 2000. *The Cathedral and the Bazaar*. (`http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar` [XHTML; note that other versions, including PostScript and XML are available from the same location.]).

[5] Collins-Sussman, B., B. W. Fitzpatrick and C. M. Pilato. *Version Control with Subversion*. O'Reilly Media, 2006. (online, `http://svnbook.red-bean.com`).

[6] Calder, B.R., and the Open Navigation Surface Working Group (2004). *Digital Security Scheme for the Bathymetric Attributed Grid (BAG)*, Version 1.1, February 2004. (`http://www.opennavsurf.org/papers/ons_digitalsig.pdf`).

[7] Schneier, B. *Applied Cryptography*, 2ed. Wiley, 1995.

[8] IEEE Standard Committee 754 (1985). *IEEE Standard for Floating Point Binary Arithmetic, IEEE Std. 754-1985*. IEEE Inc., 345 East 47th St., New York, NY 10017, USA.

[9] `doxygen` User Manual, online. (`http://www.stack.nl/~dimitri/doxygen/download.html#latestman`).

[10] HDF-5 Documentation (`http://hdf.ncsa.uiuc.edu/HDF5/doc/`).

[11] XERCES Documentation (`http://xml.apache.org/xerces-c/`).

[12] Deblier, R. (2004) BeeCrypt API Documentation, (`http://beecrypt.sourceforge.net/doxygen/c/index.html`).

[13] McDonald, W. *BAG 1.4.0 – Compression Proposal*, 2007 (`http://www.opennavsurf.org/papers/BagCompressionP.pdf`)

Appendix A

See spreadsheet named “Appendix A.xls”