



NGA STANDARDIZATION DOCUMENT

NGA.STND.0055-02_1.0_MIPCFFDD

Modality Independent Point Cloud (MIPC)

Volume 2

File Format Description Document (FFDD)

Version: 1.00

Date: 2016-06-08

CONTACTS

The following point of contact is provided for assistance in understanding the contents of this implementation profile

NGA/TAEA

Office of the Chief Information Officer, Information Technology Directorate (CIO-T)

7500 GEOINT Drive

Springfield, VA 22150

CHANGE LOG

Date	Version	Description	DR/CA
2016-02-29	0.01	Draft for Government review	
2016-03-02	1.00	Draft for NTB review	
2016-06-08	1.00	Final baseline for GWG	

TBR/TBD LOG

Page	TBR/TBD	Description	Date Addressed

Executive Summary

This Modality Independent Point Cloud (MIPC) standard, Volume 2, File Format Description Document (FFDD), describes the conventions, methods, and processes for storing and extracting geospatial point cloud information based on the MIPC standard for data and metadata content established by the National Geospatial-Intelligence Agency (NGA). The NGA has selected the Hierarchical Data Format 5 (HDF5) as the data transport layer for MIPC. This document describes methods whereby data producers can place MIPC-compliant data and metadata in the HDF5 file format in accordance with the MIPC standard. It also describes methods whereby users of MIPC datasets may extract data components from the HDF5 file. The corresponding Volume 1 of the MIPC standard provides the design and description of the content and structure of MIPC data and metadata.

The main objective of MIPC is to create a general purpose point cloud file for storage and transmission within the National System for Geospatial-Intelligence (NSG) in a standard form that will maximize interoperability and data fusion. A related standard, Sensor Independent Point Cloud (SIPC), standardizes a single point cloud created by a single Light Detection and Ranging (LIDAR) sensor in a single pass, and therefore includes sufficient LIDAR metadata to enable custom processing by image scientists as well as exploitation by image analysts. The MIPC specification can be considered a downstream product, generalizing and abstracting the three-dimensional (3-D) point cloud data structure, while still maintaining critical metadata from the originating modalities. As such, the MIPC standard is appropriate for merging LIDAR point clouds from different collections or point clouds derived from multiple two-dimensional (2-D) images. Therefore, the MIPC standard can accommodate point clouds created from other modalities besides LIDAR, such as Electro-Optical (EO) imagery, Radio Detection and Ranging (RADAR), and Wide Area Motion Imagery (WAMI) systems. Since most MIPC files accommodate temporal, radiometric, and spectral attributes per point, MIPC can be considered a multi-dimensional point cloud.

A MIPC file is structured such that the data and metadata are designed to be modality independent to the extent possible and appropriate. This allows all point clouds conforming to the MIPC standard to be treated in a similar manner for archiving, visualization, and exploitation. Consequently, a single MIPC file may contain multiple point clouds from the same or different sensors or even from different modalities. As GEOINT data, all data within a MIPC file pertains to a single SCENE of interest. MIPC provides an interoperable mechanism for disseminating, exploiting, and visualization different sensor data for that scene.

The container for MIPC content is the HDF5 file format. HDF5 is a non-proprietary, open, industry standard format for storing complicated, scientific (numeric) data. It was chosen by NGA as the optimum format for ingest, storage, and dissemination of exploitation-ready point cloud data via NSG archives. Legacy point cloud data standards have limited in metadata content, are rigid in data typing, and store wideband data as monolithic binary blocks. These formats often lack any logical grouping or self-description mechanisms within the dataset. The HDF5 file format, as well as the MIPC metadata content and data structures, provide efficient discovery and random access of data across multiple point clouds within a

single file.

This document provides an HDF5 profile for MIPC data. Should a future file format prove more beneficial to MIPC, a new profile volume could be written for that format without changing the content in Volume 1 of the MIPC standard; but that format would need to be capable of handling the structures within MIPC. HDF5 is also a self-discoverable format. If the content of the MIPC Volume 1 standard changes due to future metadata and data requirements or enhancements, this HDF5 implementation profile would require very minor changes with minimal impact to software tools developed against this HDF5 interface.

Contents

1	Introduction	1
1.1	Scope	1
1.2	Applicable Documents	2
2	File Format	4
2.1	Relevant HDF5 Aspects	5
2.1.1	Physical Structure	5
2.2	Components of the API	6
2.3	Generic Code Examples	8
2.3.1	C++	8
2.3.2	Python	10
2.3.3	IDL	14
2.4	MIPC Specific Implementations	15
2.4.1	Groups	15
2.4.2	Datasets	20
3	File Workflows	23
3.1	Process	23
3.2	Select Coordinate System and Datum	23
3.3	Assemble Point Data	24
3.4	Define Channels	25
3.5	Tile Point Data	25
3.5.1	Translate and Rotate Points	29
3.5.2	Convert Position Data to Scaled Integer	29
3.6	Error Data	30
3.6.1	Global Error	31
3.6.2	GPM	31
3.6.3	Per Point Error	31
4	Extending MIPC	32
4.1	Extending Groups	32
4.2	Extending Datasets	32
4.3	Adding Imagery	32

List of Figures

1	High level groups within a sample MIPC file.	17
2	CLOUD and SENSOR indices restart counting under each MODALITY group.	18
3	Groups under a CLOUD group within a sample MIPC file.	19
4	String datasets within a sample MIPC file. Data is fictional.	22
5	Numeric datasets within a sample MIPC file. Data is fictional.	23
6	ECEF-Referenced Local Coordinate System.	26

List of Tables

1	Applicable Documents	3
2	HDF Resources	5
3	MIPC Data Types	7

1 Introduction

1.1 Scope

The Modality Independent Point Cloud (MIPC) File Format Description Document (FFDD) provides descriptions for how point cloud data sets are stored in, and read from, files conforming to the MIPC standard and required file format - Hierarchical Data Format 5 (HDF5). This document provides guidance to producers of point cloud data sets delivered to the National Geospatial-Intelligence Agency (NGA), as well as consumers of National System for Geospatial-Intelligence (NSG) point cloud data, regardless of the sensor modality.

In keeping with the precedent set by forerunners of the MIPC, most notably the Sensor Independent Complex Data (SICD) and Sensor Independent Point Cloud (SIPC) standards, this document is designated Volume 2 of the MIPC series because it describes for data providers the placement of MIPC point cloud data and metadata in the HDF5 file format. It also describes methods whereby users of MIPC data can read and properly extract the data components from the HDF5 file. MIPC Volume 1, *Design and Implementation Description Document (DIDD)*, specifies the content from the MIPC file in terms of the data and metadata object definitions, the hierarchical structure, data types, object naming and tag conventions, data tiling conventions, and coordinate systems. The SICD series of documents includes a Volume 3, *Image Projections Description Document*. An analogous Volume 3 is not required for MIPC because point cloud data is inherently georeferenced and potentially coregistered by the time it is processed to the level at which it is placed in the MIPC file.

The MIPC defines a standard for point clouds developed from any remote sensing modality. A MIPC file is structured such that the data and metadata are designed to be modality independent to the extent possible and appropriate. This allows all point clouds conforming to the MIPC standard to be treated in a similar manner for archiving, visualization, and exploitation. However, there is a need for a small amount of critical modality specific metadata, and such information is in a specific group within the file, called METADATA, separate from the rest of the modality independent content.

While the MIPC family of documents describe the content of an MIPC file and how it organizes data, specifications and standards on the HDF5 file format itself are maintained by an external consortium and are outside the scope of this document.

With other file formats, especially those based on a Key-Length-Value (KLV) approach, the concept of the byte-ordered *location* of specific data objects within a binary data file is extremely important. One needs to know, for instance, how many bytes every particular data object uses, as well as their precise order and quantity within the file in order to ascertain precisely which bytes in the binary file correspond to any particular piece of data or metadata. For instance, to read a 4-byte data field A, one needs to compute the number of data fields which precede it and how many bytes they consume, X, skip X bytes into the file and read bytes X+1 to X+4. Many file format description documents analogous to this one devote the bulk of the documentation to describing the order and placement of specific data

fields at specific byte locations in the binary file, which is necessary in order to map data components to their locations in the file.

HDF5, on the other hand, is an object-oriented system in which data objects are written to, and read from, HDF5 files through calls to an open and publicly available application programming interface. Each data component is treated as an object, and the Application Programming Interface (API) encapsulates and tracks the internal locations, sizes, and types of all the objects. These details of how each data component maps to a specific byte location in a file are invisible and irrelevant to the HDF5 user. Consequently, it is unnecessary for this document to specify many parameters that comprise the bulk of other similar file format specifications and interface control documents. These irrelevant concerns for HDF5 include issues such as the size, data type, and location of various data records, record size limitations and overflow extensions, header and subheader locations, content, and organization.

Instead, this document will present high-level workflows and API implementation examples for reading and writing data and metadata to and from a data file that conforms to the MIPC standard. The API examples used throughout this document are presented in a small set of common languages in order to demonstrate concepts clearly, but are easily translatable to other languages by those readers proficient with programming in those languages.

The corresponding Volume 1 specification provides the design and description of the MIPC content. The HDF5 was selected as the file format for MIPC based on lessons learned from the development of the SIPC standard and the associated SIPC File Format Trade Study [1] conducted in 2012.

1.2 Applicable Documents

The documents listed in Table 1 were referenced in the development of this standard. Users of this document should investigate recent editions and change notices of the items listed.

Table 1: Applicable Documents

Title	Version
CMMD Level 2, LIDAR Data and Metadata: Conceptual Model and Metadata Dictionary for Enterprise Level 2 Volume Return Products	Ver. 1.0
Generic Point Cloud error model ¹	Ver. 1.0
Intelligence Community (IC), Information Security Marking (ISM) Metadata Specification ²	Ver. 13
NGA, SIPC Volume 1, Sensor Independent Point Cloud, Design and Implementation Description Document	Ver. 1.02
NGA, Sensor Independent Derived Data (SIDD), Vol. 1, Design and Implementation Description Document, NGA.STND.0025-1_1.0	01 Aug 2011
NITFS, The Compendium of Controlled Extensions (CE) for the National Imagery Transmission Format (NITFS), STDI-0002, Appendix E, Airborne Support Data Extensions (ASDE)	Ver. 2.1, 16 Nov 2000
NITFS, The Compendium of Controlled Extensions (CE) for the National Imagery Transmission Format (NITFS), STDI-0002, Appendix L, HISTOA Extension	Ver. 1.0, 01 Aug 2007
NITFS, The Compendium of Controlled Extensions (CE) for the National Imagery Transmission Format (NITFS), STDI-0002, Appendix O, Multi-image Scene (MiS) Table of Contents (MITOCA) Tagged Record Extension (TRE)	Ver. 1.0, 31 Mar 2006
NITFS, The Compendium of Controlled Extensions (CE) for the National Imagery Transmission Format (NITFS), STDI-0002, Appendix X, General Purpose Band Parameters (BANDSB) Tagged Record Extension (TRE)	Ver. 1.0, 30 Sep 2004
NMF Part 1, NSG Metadata Foundation, Core	Ver. 2.1
W3C, XML Schema Part 2: Datatypes Second Edition ³	Accessed 13 Jul 2015

¹<https://nsgreg.nga.mil/doc/view?i=1799>

²<http://www.dni.gov/index.php/about/organization/chief-information-officer/information-security-marking-metadata>

³<http://www.w3.org/TR/xmlschema-2>

2 File Format

The file format used as a container for MIPC is the [HDF5](#) format. This format was developed by the National Center for Supercomputing Applications (NCSA), and used extensively by National Aeronautics and Space Administration (NASA) for remote sensing data, and already in use for other GEOINT platforms supporting the NGA. HDF5 is a Geospatial Intelligence Standards Working Group (GWG) approved standard and provides the following benefits:

self-discoverable HDF files inform the user of its contents (field names, dimensions, bit depth), so a data description document is not needed to read an HDF file and extract content, but a document can provide additional support context, such as in this document

api HDF has an extensive [API](#) with bindings in every major programming language

flexible HDF5 can store any typical data structure including vector, images, video, point clouds, and meshes; and at any bit depth defined and interpreted as needed

scalable HDF5 has no theoretical limitations in file size or dataset size; however, current computer limitations recommend files less than 2 Terabytes (TB) in size

parallel HDF5 supports parallel file access using [Message Passing Interface \(MPI\)](#)

free The basic HDF5 API code and a viewer are free software

maintained HDF5 tools and code libraries are maintained by a dedicated working group

platform independent HDF files can be used on any major operating system, including Windows, Linux, Unix, and FreeBSD.

Multiple [API](#) libraries are available for use in the following programming languages:

- C
- C++
- Java
- Python
- FORTRAN
- Interactive Data Language (IDL)
- MATLAB
- Mathematica

There are several internet sites that provide information on using HDF5 in the various programming languages. Such sites provide documentation, modules, and code examples of implementations. Some of these sites are listed in Table 2.

Table 2: HDF Resources

Title, Description	URL
The HDF Group, primary resource for HDF information and tools	https://www.hdfgroup.org/HDF5
National Center for Supercomputing Resources	http://www.ncsa.illinois.edu
HDFView, free tool for viewing any HDF file	https://www.hdfgroup.org/products/java/hdfview/index.html
HDF5 code examples	https://www.hdfgroup.org/HDF5/examples
h5py, free Python library for interfacing with HDF5	http://www.h5py.org/
HDF5 programing overview for IDL	http://www.exelisvis.com/docs/HDF5_Overview.html
HDF5 programing overview for MATLAB	http://www.mathworks.com/help/matlab/hdf5-files.html?s_tid=gn_loc_drop

2.1 Relevant HDF5 Aspects

HDF5 provides several components and capabilities useful for the storage of complicated data. However, there was a design goal to keep the MIPC structure as simple as possible to reduce the type of interfaces and coding functions that would need to be developed. This section describes those HDF5 components that are mandated for this standard along with their mapping to conceptual data items described in the Volume 1 of the MIPC standard.

2.1.1 Physical Structure

The Volume 1 of the MIPC standard intentionally used terminology that maps directly to HDF5 terminology. Although this was not completely necessary, it greatly simplifies the instructions that follow. For example, the high level collections of information were called *groups*, and this is the same term used within HDF5 implementation for that type of information. Since Volume 1 was conceptual, these groups could have been called **classes**, as they are in other similar specifications, such as the Conceptual Model and Metadata Dictionary (CMMD) and NSG Metadata Foundation (NMF).

Therefore, the conceptual groups in MIPC Volume 1 are implemented as HDF5 groups within the file structure.

Similarly, the *datasets* described in Volume 1 of the MIPC standard are implemented as HDF5 *datasets* within the file structure. These elements are the atomic data structures containing numeric and string arrays, whereas the groups are used to organize this information. As in Volume 1, groups will be named in all UPPERCASE characters, and dataset names

will be written in `CamelCase`, unless the group is inherited from another specification with a different convention, such as the NMF and Generic Point-Cloud Model (GPM) standards.

In summary, the HDF5 format allows for a physical implementation to match the conceptual and logical design identically.

Attributes are a somewhat useful construct within the HDF5 library; however, there is only minor advantage to using attributes over datasets for textual metadata. Although reading and writing **attributes** are easier operations than string datasets, using attributes would require a separate type of function to access this type of data as opposed to datasets. Additionally, attributes are not compressible. Furthermore, attributes are not as salient in some HDF data viewing tools such as HDFView.

However, **attributes** are not restricted by the MIPC standard either, and are therefore available for use by various programs to store additional information (meta-metadata), as desired. Attributes are a simple mechanism to store any additional information to aid in the use or interpretability of the dataset to which it is assigned. Attributes should only be used for informal notes. Column names are a good example of the type of information that could be placed in attributes. Attributes should not be used to store critical data that is not included in the MIPC standard. This use of the term attributes within HDF5 should not be confused with point-wise data attributes.

HDF5 also has a concept called *tables*, which are compound data structures that permit different columns to be different data types, and adds a title to each column. There is also a convenient Python module for working with these called **PyTables**. However, this table concept may not translate to other formats, and might therefore be HDF5 specific. The intention of this standard is to be flexible enough to accommodate more advanced formats in the future, which is why Volume 1 is format agnostic. Therefore, this standard does not require the data to be in tables, but we do not believe that using tables in specific MIPC writers would break this standard.

They are additional elements available within HDF5 not discussed in this document, such as **enumeration**, **hyperslabs**, and **hardlinks**. These additional concepts are not necessarily precluded either. The implementation decision lies within the MIPC software tools and does not need to be mandated by this specification. As long as the required groups and datasets are included as specified in the two volumes of the MIPC standard, the file is compliant. Similarly, advanced HDF5 techniques, such as **chunking**, **compression**, and **Fastbit**, are not forbidden as long as the content remains within specification. Linking to information in separate HDF5 files is also possible, but such an implementation is a systems engineering decision.

2.2 Components of the API

An **API** is essentially a code library of functions or classes created to facilitate complex processes by encapsulating and standardizing a series of algorithmic steps. They are meant

to be an interface for tool developers. Information about supported languages, software development environments, and the HDF5 APIs themselves can be downloaded from the HDF Group website⁴. Although some languages require a download or installation of a library, HDF5 functions are built in to the current versions of MATLAB and the Interactive Data Language (IDL).

Once the API installation is complete, a programmer can start developing code that makes calls to the different functions of the API. The atomic data type supported in HDF5 are given in Table 3. Compound types (arrays of structures) are discouraged given the simplicity and structure of MIPC.

Table 3: MIPC Data Types

Abbreviation	Data Type	Size (Bytes)
UInt8	unsigned integer	1
UInt16	unsigned integer	2
UInt32	unsigned integer	4
UInt64	unsigned integer	8
Int8	signed integer	1
Int16	signed integer	2
Int32	signed integer	4
Int64	signed integer	8
UInt variable	unsigned integer	variable
Float32	single precision floating point	4
Float64	double precision floating point	8
String	character string of variable length	variable
String(XX)	character string of fixed length	XX

⁴<http://www.hdfgroup.org>

2.3 Generic Code Examples

This section provides sample code for working with any HDF5 file, whether MIPC, SIPC, or any other product type. This document attempts to cover the most anticipated programming languages for MIPC.

2.3.1 C++

Listing 1: Sample C++ code for creating a group in an HDF5 file

```

1  /* Create a group in the file under root "/" */
2
3  first_group = H5Gcreate(file , "/FIRSTGROUP" , H5P_DEFAULT, H5P_DEFAULT,
   H5P_DEFAULT);

```

The previous code snippet shows how to create a group under the **root** (the highest possible hierarchical level) of an HDF5 file. Code Listing 2 provides an example of how to create a group under a group, a technique that will be particularly useful in creating the multi-level hierarchical data structure of SIPC.

Listing 2: Sample C++ code for creating child groups in an HDF5 file

```

1  /* Create new child group under an existing group
2  by specifying the absolute path of the group */
3
4  new_child = H5Gcreate(file , "/PARENT/CHILD" , H5P_DEFAULT, H5P_DEFAULT,
   H5P_DEFAULT);

```

The following code in Listing 3 demonstrates some of the functions used to store a dataset within HDF5.

Listing 3: Sample C++ code for creating a dataset in an HDF5 file

```

1  /* Create a new file using H5F_ACC_TRUNC access ,
2  default file creation properties , and
3  default file access properties. */
4
5  try
6  {
7      H5File file( FILE_NAME, H5F_ACC_TRUNC );
8      // Define the size of the array and create the data space for fixed size
9      dataset.
10     hsize_t    dimsf[2]; // dataset dimensions
11     dimsf[0] = NX;
12     dimsf[1] = NY;
13     DataSpace dataspace( RANK, dimsf );
14     // Define datatype for the data in the file.
15     // We will store little endian INT numbers.
16     IntType datatype( PredType::NATIVE_INT );

```

```
16 datatype.setOrder( H5T_ORDER_LE );
17 // Create a new dataset within the file
18 // using defined dataspace and datatype and default dataset creation
   // properties.
19 DataSet dataset = file.createDataSet( DATASETNAME, datatype, dataspace );
20 // Write the data to the dataset
21 // using default memory space, file space, and transfer properties.
22 dataset.write( data, PredType::NATIVE_INT );
23 } // end of try
24
25 // Catch failure caused by the H5File operations
26 catch( FileIOException error )
27 {
28     error.printStackTrace();
29     return -1;
30 } // end of catch
```

2.3.2 Python

This section provides generic example code for working with any HDF5 file in Python. The examples are based on the h5py library, and there may be other techniques within h5py as well as other libraries for some of these processes. First, Listing 4 demonstrates many functions related to writing content to a new HDF5 file, including groups and datasets.

Listing 4: Sample Python code for creating an HDF5 file

```
1 # -*- coding: utf-8 -*-
2
3 # Example code for writing HDF5 files
4
5 import h5py
6 import numpy as np
7
8 filepath = r'C:\Test\myfile.h5'
9
10 print('Creating and opening file for streamed writing...')
11 f = h5py.File(filepath, 'a')
12
13 print('Creating groups...')
14
15 parent = f.create_group('Parent')
16 print('Group Name: ', parent.name)
17
18 print('Creating a subgroup...')
19 child = parent.create_group('Child')
20 print('Subgroup Name: ', child.name)
21
22 print('Creating a group and subgroups directly...')
23 grp3 = f.create_group('A/B/C')
24 print(grp3.name)
25
26 print('Deleting a leaf group...')
27 grp4 = f['/A/B']
28 del grp4['C']
29
30 print('Creating a scalar dataset...')
31 child['data1'] = 3.41
32
33 print('Creating an array dataset, method 1...')
34 signal = np.arange(2000)
35 child['signal1'] = signal
36
37 print('Creating an array dataset, method 2...')
38 signal2 = np.arange(1000)
39 signal2set = child.create_dataset('signal2', data=signal2)
40
41 print('Creating hard links on groups...')
42 f['/A/B/HARDLINKED'] = f['/Parent/Child']
43
```

```

44 print('Create attributes on a dataset...')
45 signal2set.attrs['Source'] = 'NGA'
46
47 print('Creating a fixed length ASCII string dataset...')
48 ds2 = f.create_dataset("/A/FixedAscii", (100,), dtype="S10")
49 ds2[0] = b'ABCDEFGHJKLM'
50 ds2[1] = b'NOPQRSTUVWXYZ'
51
52 print('Creating a variable length ASCII string dataset...')
53 db = h5py.special_dtype(vlen=bytes)
54 ds3 = f.create_dataset("/A/VariableAscii", (100,), dtype=db)
55 ds3[0] = b'ABCDEFGHJKLM'
56 ds3[1] = b'NOPQRSTUVWXYZ'
57 print('ID: ', ds3.id)
58 print('Value: ', ds3.value)
59
60 print('Labeling dimensions...')
61 f['/A/LABELED'] = np.ones((4, 3, 2), 'f')
62 f['/A/LABELED'].dims[0].label = 'z'
63 f['/A/LABELED'].dims[1].label = 'y'
64 f['/A/LABELED'].dims[2].label = 'x'
65
66 print('Ragged arrays within cells...')
67 dt = h5py.special_dtype(vlen=np.dtype('int32'))
68 dset = f.create_dataset('variableLengthInts', (100,), dtype=dt)
69 dset[0] = [1,2,3]
70 dset[1] = [1,2,3,4,5]
71 f['variableLengthInts'].attrs['column_names'] = b'My Column Name'
72
73 print('Enumerations...')
74 de = h5py.special_dtype(enum=('i', {"RED": 0, "GREEN": 1, "BLUE": 2}))
75 print('Enumeration: ', h5py.check_dtype(enum=de))
76 ds = f.create_dataset("/A/EnumColors", (100,100), dtype=de)
77 print('Dataset kind: ', ds.dtype.kind)
78 # Use integer values within code, and string keys written to cells
79 ds[0,:] = 2
80 ds[1,:] = 1
81 ds[2,:] = 0
82 print(ds[0,0])
83 print(ds[1,1])
84 print(ds[2,2])
85
86
87 f.close()

```

Second, Listing 5 demonstrates many functions related to reading content from an existing HDF5 file.

Listing 5: Sample Python code for reading HDF5 file

```
1 # -*- coding: utf-8 -*-
2
3 # Example Python code for reading HDF5 files
4
5 import h5py
6
7 filepath = r'C:\Test\myfile.h5'
8
9 # Open file for streamed reading
10 f = h5py.File(filepath, 'r')
11
12 print('Root group: ', f.name)
13
14 # Print name of all top level groups under root
15 for groups in f:
16     print(groups)
17
18 # Test if group exists
19 test = '/Parent/Child' in f
20 print(test)
21
22 # Get group
23 # Use dictionary method
24 child = f['Parent']['Child']
25
26 # Get dataset under group
27 dataset = child['signal2']
28
29 # Get dimensions of dataset
30 print(dataset.shape)
31 print(dataset.dtype)
32
33 # Read numeric array from dataset
34 data = dataset[()]
35 print(data)
36
37 # Test if attribute exists
38 test = 'Source' in dataset.attrs
39 print(test)
40
41 # Get attributes of dataset
42 value = dataset.attrs['Source']
43 print(value)
44
45 # Get string dataset
46 # Use full path group method
47 sdataset = f['/A/FixedAscii']
```

```
48 sdata = sdataset [()]
49 print(sdata)
50
51 # Read strings from dataset
52
53 f.close()
```

2.3.3 IDL

This section provides generic code for working with HDF5 files within the IDL. Listing 6 demonstrates how to read content from an HDF5 file programmatically.

Listing 6: Sample IDL code for working with an HDF5 file

```

1 ; Get tree structure of group paths, but not data
2 groups = H5.PARSE(filepath)
3
4 ; Open a file
5 file_id = H5F.OPEN(filepath)
6
7 ; Find out how many members are under a group
8 n_members = H5G.GETNMEMBERS(file_id, group_path)
9
10 ; Get the name of a specific member at position n
11 ; This can be used to build an object path
12 member_name = H5G.GETMEMBERNAME(file_id, group_path, n)
13
14 ; Get the data object at a specific object path
15 data_id = H5D.OPEN(file_id, object_path)
16 data = H5D.READ(data_id)
17
18 ; Close the data object
19 H5D.CLOSE, data_id
20
21 ; Close the file
22 H5F.CLOSE, file_id

```

Listing 7 shows how to call an IDL tool to display HDF5 data interactively. The tool is very similar to HDFView with two additional capabilities. First, 2D arrays are automatically displayed as images within the tool. Second, the tool includes a button to load any dataset into the IDL workspace as a variable of the same name.

Listing 7: Sample IDL code for loading HDF5 data into an IDL viewer

```

1 ; Display metadata and data
2 d = H5.BROWSE(filepath)
3
4 ; Manually import mydataset from H5.BROWSE into IDL as a structure variable
5
6 ; Then get the data from that variable structure
7 data = mydataset._data

```

The variable is imported into the workspace as a structure, with one of the fields (`._data`) containing the actual dataset array. So the last line of the code example would actually be executed within the IDL workspace.

2.4 MIPC Specific Implementations

This section provides additional explanation of specific HDF5 implementation details for MIPC through examples. Screen shots of sample data are provided using HDFView, and all sample MIPC code presented will be from the Python programming language as the most simplified programming example. These examples could be easily extended to other programming languages as needed by someone familiar with the HDF5 API for the desired language, or at least the examples in § 2.3.

It is anticipated that extensive software classes for MIPC systems will be developed and evolve for data manipulation and streamlining of the I/O processes, at least convenience functions for wrapping HDF5 API functions calls for MIPC that are common and repetitive. However, sufficient capability is available in the core HDF5 API.

2.4.1 Groups

Listing 8 demonstrates simple methods for creating a MIPC file containing some of the high level groups. In particular, repeated groups containing incremental indices are demonstrated in this example.

Listing 8: Sample Python code for writing groups to a MIPC file with repetition indices

```

1 # -*- coding: utf-8 -*-
2
3 # Example MIPC code
4 # Adding datasets to groups
5
6 import h5py
7 import numpy as np
8
9 # Special HDF5 data types
10 # Variable length ASCII strings
11 st = h5py.special_dtype(vlen=str)
12 db = h5py.special_dtype(vlen=bytes)
13
14 # Groups as lists by level
15 root = [ 'FILE', 'REFERENCES', 'MISSION', 'SCENE' ]
16 file = [ 'IDENTIFICATION', 'SOURCE', 'SECURITY' ]
17 mission = [ 'PURPOSE', 'TARGETS' ]
18 scene = [ 'SPATIAL', 'TEMPORAL', 'SPECTRAL', 'RADIOMETRIC' ]
19 # Dictionary of groups by list
20 groups = { 'FILE':file, 'MISSION':mission, 'SCENE':scene }
21
22 # Constants
23 productClass = 'MIPC'
24 productType = 'Point Cloud'
25
26 # Functions
27
28 # Converting a list of variable length strings to ASCII instead of Unicode
29 def prepStringList(stringList):

```

```

30     asciiList = [n.encode("ascii", "ignore") for n in stringList]
31     return asciiList
32
33 # Adding list of variable length strings
34 def writeStringList(file, groupPath, stringList):
35     asciiList = prepStringList(stringList)
36     n = len(asciiList)
37     dset = f.create_dataset(groupPath, (n,), dtype=db)
38     for i in range(n):
39         dset[i] = asciiList[i]
40     return file
41
42 # Program
43
44 # Open file
45 productName = 'my_mipc_2'
46 filepath = 'C:\\Test\\' + productName + '.h5'
47 print('Creating and opening file for streamed writing...')
48 f = h5py.File(filepath, 'w')
49
50 print('Creating groups...')
51 for x in groups:
52     print(x)
53     temp = f.create_group(x)
54     for y in groups[x]:
55         temp = f[x].create_group(y)
56
57 # Adding string scalar datasets
58 f['FILE']['IDENTIFICATION']['ProductName'] = productName
59 f['FILE']['IDENTIFICATION']['ProductClass'] = productClass
60 f['FILE']['IDENTIFICATION']['ProductType'] = productType
61
62 f['FILE']['SOURCE']['SiteName'] = 'Harris CPED'
63
64 f['FILE']['SECURITY']['classification'] = 'UNCLASSIFIED'
65
66 # Adding numeric array datasets
67 lats = [36.123, 38.124]
68 lons = [67.123, 67.124]
69 alts = [-1.123, 1000.123]
70 coverage = np.array([lats, lons, alts])
71 f['SCENE']['SPATIAL']['Coverage'] = coverage
72
73 datums = np.array([0, 0])
74 f['SCENE']['SPATIAL']['Datums'] = datums
75 f['SCENE']['SPATIAL']['PointCoordinateSystem'] = 0
76
77 countries = ['AF', 'TJ', 'UZ']
78
79 groupPath = '/SCENE/SPATIAL/Countries'
80 f = writeStringList(f, groupPath, countries)
81
82 # Check reading strings
83 data = f['FILE']['IDENTIFICATION']['ProductName']

```

```
84 datas = data[()]\n85 print(datas)\n86\n87 f.close()
```

Figure 1 provides a screen shot of a MIPC file depicting the high level groups as they appear in HDFView. Notice that there are multiple **MODALITY**, **CLOUD**, and **SENSOR** groups each numbered with their index, starting with 0. This index number is defined in Volume 1 as letter placeholders, such as **MODALITY_a**, **SENSOR_b**, etc. for the purposes of generic, conceptual explanation. All index numbering in MIPC is 0-based since HDF5 numbering is inherently 0-based.

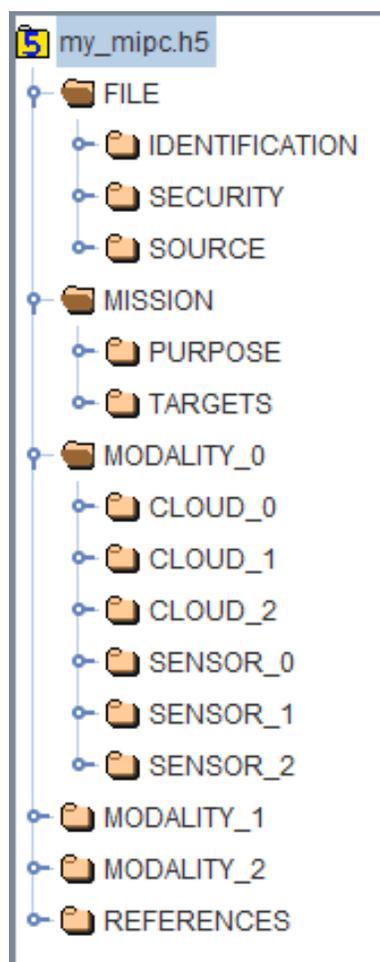


Figure 1: High level groups within a sample MIPC file.

The group indices restart from 0 under each parent as demonstrated in Figure 2. This means that a group name is only unique if the entire path is included. Although it may seem intuitive that **CLOUD** should be under sensor, recall from Volume 1 that a cloud can be made from different sensors, provided (at this time) that those sensors are from the same modality.

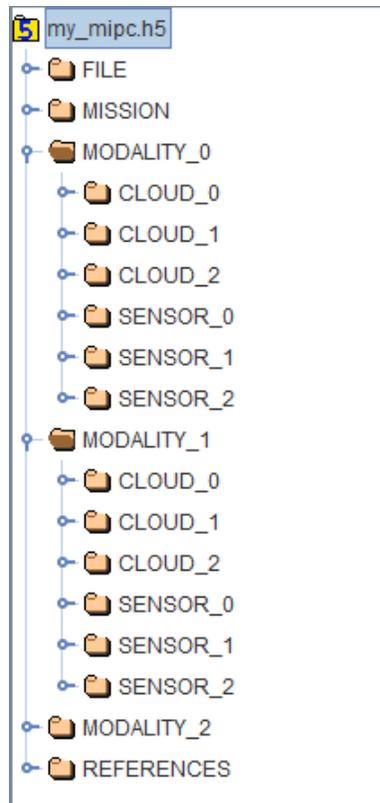


Figure 2: CLOUD and SENSOR indices restart counting under each MODALITY group.

Figure 3 depicts the possible groups under a **CLOUD** group. Shown are the group containing **GPM** error information as well as the **POINTS** group containing the actual data points for that cloud. There will be a separate **POINTS** group under each **CLOUD**. Recall from Volume 1 that **GPM** data is per **CLOUD** only at this time, but may be expanded in the future as processing algorithms mature, stabilize, and standardize.

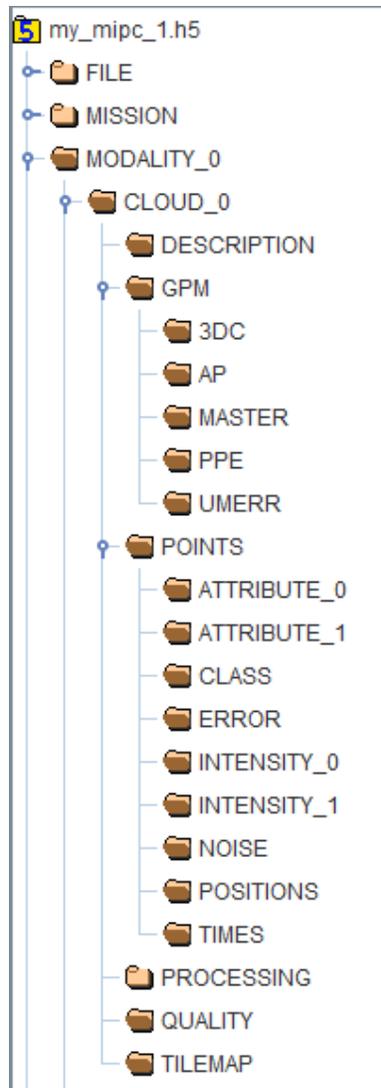


Figure 3: Groups under a CLOUD group within a sample MIPC file.

2.4.2 Datasets

Listing 9 demonstrates writing datasets to groups within a MIPC file. The example groups are simple, but identification for large datasets. In particular, string arrays are included in the example. String arrays are the only topic in HDF5 that can be confusing, and there are varying references on the subject. The code listing includes convenience functions at the top for simplifying the processing of string arrays. Notice that **ProductClass** and **ProductType** are standard constants within this code.

Listing 9: Sample Python code for writing numeric and string datasets to a MIPC file

```

1 # -*- coding: utf-8 -*-
2
3 # Example MIPC code
4 # Adding groups with indices
5
6 import h5py
7
8 st = h5py.special_dtype(vlen=str)
9
10 filepath = r'C:\Test\my_mipc_1.h5'
11
12 print('Creating and opening file for streamed writing...')
13 f = h5py.File(filepath, 'w')
14
15 print('Creating groups...')
16
17 FILE = f.create_group('FILE')
18 IDENTIFICATION = FILE.create_group('IDENTIFICATION')
19 SOURCE = FILE.create_group('SOURCE')
20 SECURITY = FILE.create_group('SECURITY')
21
22 REFERENCES = f.create_group('REFERENCES')
23
24 MISSION = f.create_group('MISSION')
25 PURPOSE = MISSION.create_group('PURPOSE')
26 TARGETS = MISSION.create_group('TARGETS')
27
28 modalities = [0, 1, 1]
29 n_modalities = len(modalities)
30 MODALITIES = []
31 for a in range(n_modalities):
32     modalityName = 'MODALITY_' + str(a)
33     modality = f.create_group(modalityName)
34     MODALITIES.append(modality)
35     sensors = ['A', 'B', 'C']
36     n_sensors = len(sensors)
37     for b in range(n_sensors):
38         sensorName = 'SENSOR_' + str(b)
39         sensor = modality.create_group(sensorName)
40         description = sensor.create_group('DESCRIPTION')
41         f[modalityName][sensorName]['DESCRIPTION']['SensorID'] = sensors[b]

```

```

42     looks = sensor.create_group('LOOKS')
43     metadata = sensor.create_group('METADATA')
44
45     clouds = ['a', 'b', 'c']
46     n_clouds = len(clouds)
47     for c in range(n_clouds):
48         cloudName = 'CLOUD_' + str(c)
49         cloud = modality.create_group(cloudName)
50         description = cloud.create_group('DESCRIPTION')
51         processing = cloud.create_group('PROCESSING')
52         quality = cloud.create_group('QUALITY')
53         tilemap = cloud.create_group('TILEMAP')
54
55         points = cloud.create_group('POINTS')
56         positions = points.create_group('POSITIONS')
57         times = points.create_group('TIMES')
58         errors = points.create_group('ERROR')
59         classes = points.create_group('CLASS')
60         noise = points.create_group('NOISE')
61         intensity_0 = points.create_group('INTENSITY_0')
62         intensity_1 = points.create_group('INTENSITY_1')
63         attribute_0 = points.create_group('ATTRIBUTE_0')
64         attribute_1 = points.create_group('ATTRIBUTE_1')
65
66         gpm = cloud.create_group('GPM')
67         master = gpm.create_group('MASTER')
68         d3c = gpm.create_group('3DC')
69         ap = gpm.create_group('AP')
70         umerr = gpm.create_group('UMERR')
71         ppe = gpm.create_group('PPE')
72
73 f.close()

```

Figure 4 demonstrates the string variables within a MIPC file, specifically the fields under the **FILE** group. Also shown is the dataset **SCENE.SPATIAL.Countries** depicting a string array of multiple country codes based on the digraphs as defined in Volume 1 of the standard.

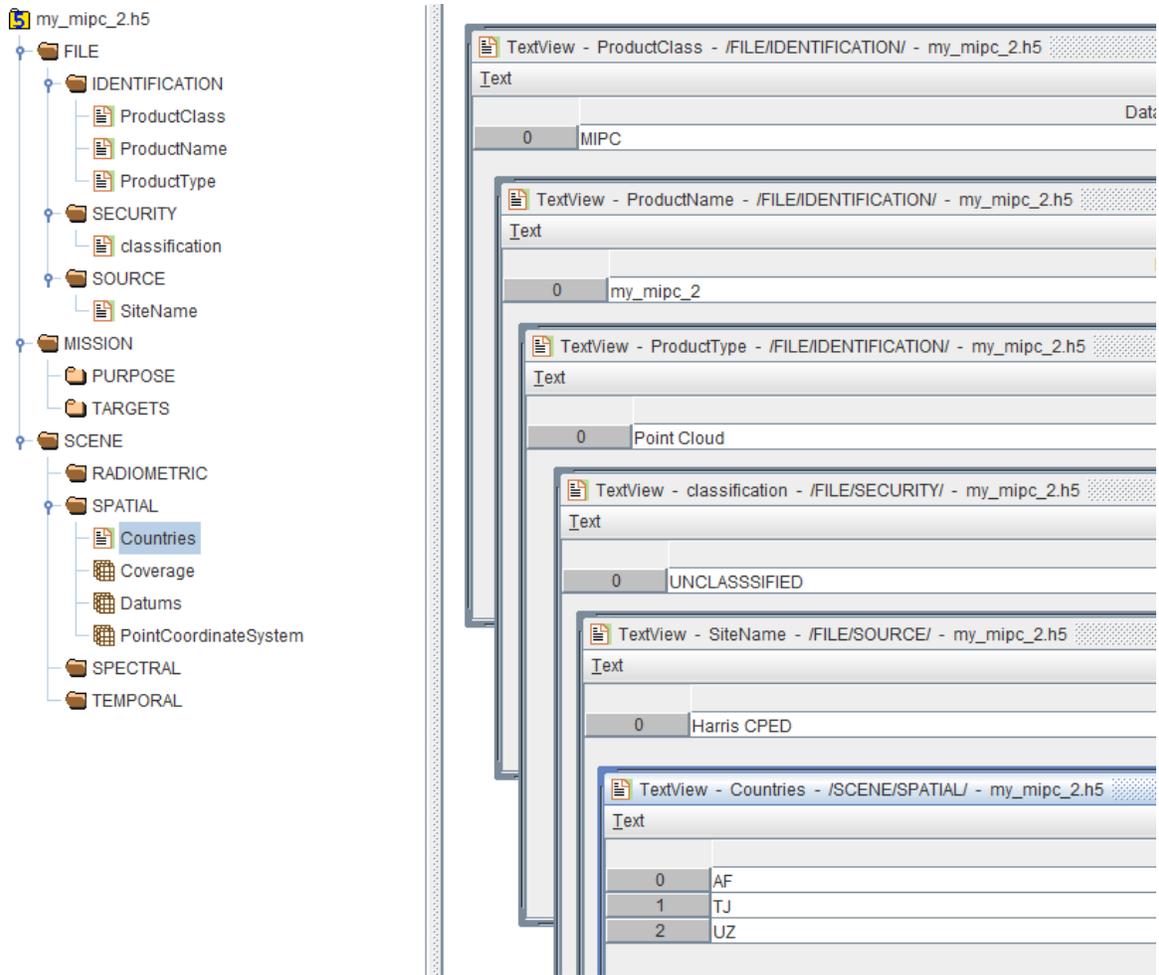


Figure 4: String datasets within a sample MIPC file. Data is fictional.

Figure 5 demonstrates some simple numeric datasets within a MIPC file, specifically the fields under the **SCENE.SPATIAL** group. The **Datums** and **PointCoordinateSystems** datasets contain integers referring to enumeration tables in the **REFERENCES** group as defined in Volume 1 of the standard. The **Coverage** dataset contains the simplified minimum and maximum geospatial extents of the data coverage over this scene from all CLOUDs.

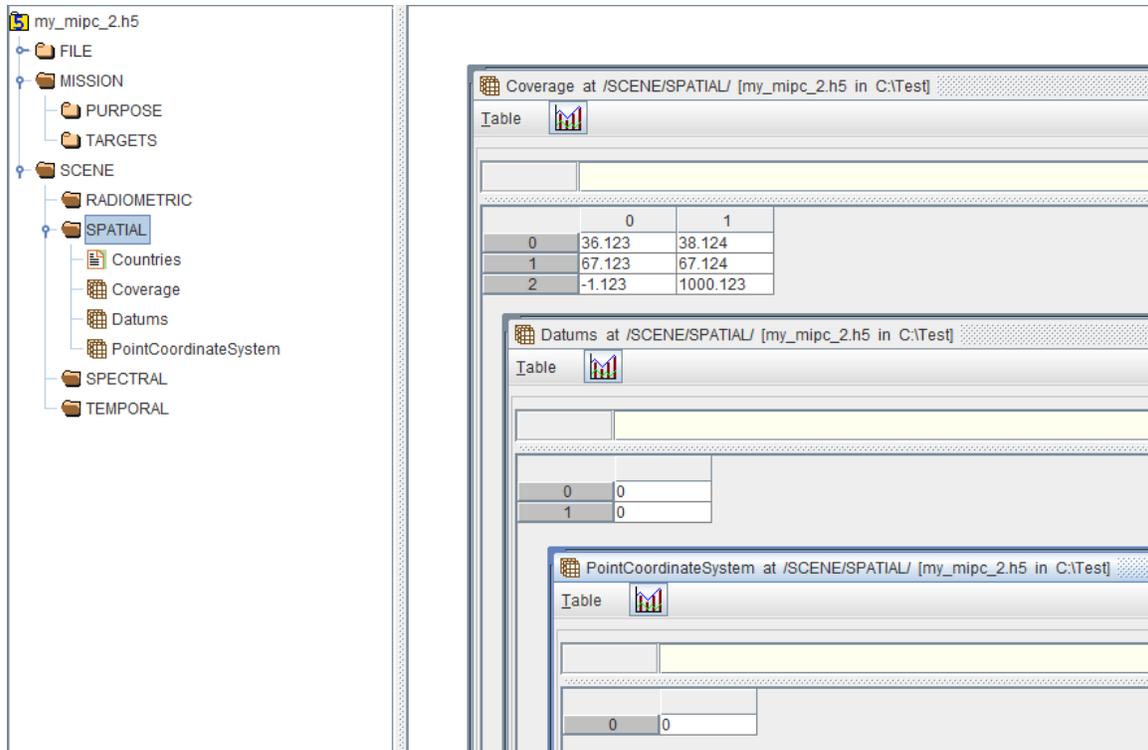


Figure 5: Numeric datasets within a sample MIPC file. Data is fictional.

3 File Workflows

3.1 Process

Given the hierarchical nature of the data, the file generation process can be followed by walking down the tree. As parent objects are defined, children objects become less ambiguous. For example, information in the **FILE** and **SCENE** groups define information that applies to all clouds and point positions, and should be able to be determined prior to channelizing or tiling the points.

Processes for generating or writing a MIPC file are defined in the sections that follow. Processes for reading MIPC files will simply employ the reverse of the subprocesses given below.

3.2 Select Coordinate System and Datum

The first step in the process is to select a reference frame for all point position data. This includes the coordinate system and datum. According to Volume 1, the recommended coordinate system for all point positions is [Earth-Centered, Earth-Fixed \(ECEF\)](#). If other coordinate systems are used, not all benefits from MIPC will be realized, particularly with [two-dimensional \(2-D\) projections](#). WGS-84 is the recommended datum for both horizontal and vertical alignment, based on [Height Above Ellipsoid \(HAE\)](#). This is the most mathe-

matically pure model; and all other datums, including geoids, can be derived downstream from this reference system. This choice must apply to the entire file, and is stored within the following datasets:

SCENE.SPATIAL.PointCoordinateSystem
SCENE.SPATIAL.Datums
SCENE.SPATIAL.EpochDate
SCENE.SPATIAL.Zone

The **EpochDate** dataset specifies the realization for the given Datums. The **Zone** dataset is conditional on:

PointCoordinateSystem = 1

which indicates UTM. The **Zone** field is a signed integer, with negative values indicating the Southern hemisphere.

3.3 Assemble Point Data

All points in a given **CLOUD** should have the same attributes, or types of information. An attribute should not be sparse, meaning that it only applies to a subset of the points, unless there is a well defined null value that is both philosophically and mathematically appropriate, and suitable for visualization. However, the use of null values adds an unnecessary number of bytes to a file. In some cases, if the number of points containing valid values for an attribute is relatively small, it may be more appropriate to make a separate cloud with those points and the attribute.

MIPC standardizes the following point attributes:

- POSITIONS
- ERROR
- NOISE
- TIMES
- CLASS
- INTENSITY

POSITIONS is the only required piece of point data. Geospatial data must have some temporal context. Data in the TIMES group is highly recommended, but if not available per point, the global duration of the file should be captured under:

SCENE.TEMPORAL.Coverage

which is a 1 x 2 array of the datetime in seconds, where the columns indicate:

minimum time, maximum time

respectively, in the entire file. From LIDAR composite clouds, these would be the earliest point and the latest point. For clouds created from imagery, this would be the collection times of the earliest image and the latest image.

The NOISE group is intended to store the probability that a given point is not from a true surface. Some sources of noise are universal to the nature of point clouds, and some are modality specific. The metrics for uncertainty may vary across algorithms. Regardless, in the end, this must be converted to an estimate of the probability in order to be modality agnostic as well as useful for human analysis, software filtering, or advanced visualization.

3.4 Define Channels

Once assembled, the point records must first be segregated into **channels** according to the nature of their intensity data (e.g., wavelength, frequency, polarization state). The channels are first defined per CLOUD, and described under:

MODALITY_a.CLOUD_c.DESCRPTION.Channels

which is an array with **g** rows and 3 or 4 columns, where **g** is the number of channels in the cloud. Each row is a unique channel, so the row index is the channel identification number used in other references. Clouds from EO and LIDAR systems have 3 columns, and clouds from SAR systems have 4 columns. The information is then populated in accordance with § 3.3.4 in Volume 1 of this standard.

3.5 Tile Point Data

All point data for a given cloud is under:

MODALITY_a.CLOUD_c.POINTS

The data can be tiled using any tiling strategy, such as a simple geospatial pattern, a quadtree, octree, or K-D tree. Metadata for the tile must then be computed to enable rapid searching for relevant tiles within the file. All pointwise attributes are then tiled in the same manner so that the row index within a tile is the point record identifier for the position and all associated attributes.

MIPC uses the ECEF system to fix point cloud data from individual tiles to a common reference frame and to geolocate them at a specific point on the earth. Every tile in a CLOUD has a local three-dimensional (3-D) Cartesian (L3DC) coordinate frame which is ECEF with the option to translate and rotate. This option may be necessary for automated processing and exploitation algorithms that prefer a local vertical axis (i.e., Z up).

To utilize the translate and rotate options, it is necessary only to specify the location of the origin of the local Cartesian frame in ECEF space, and the direction of the three coordinate axes in ECEF space. Points can then be converted back and forth through simple translation and rotation operations. Figure 6 illustrates the Local 3-D Cartesian (L3DC) system, its principal definitions, and relationship to the ECEF frame.

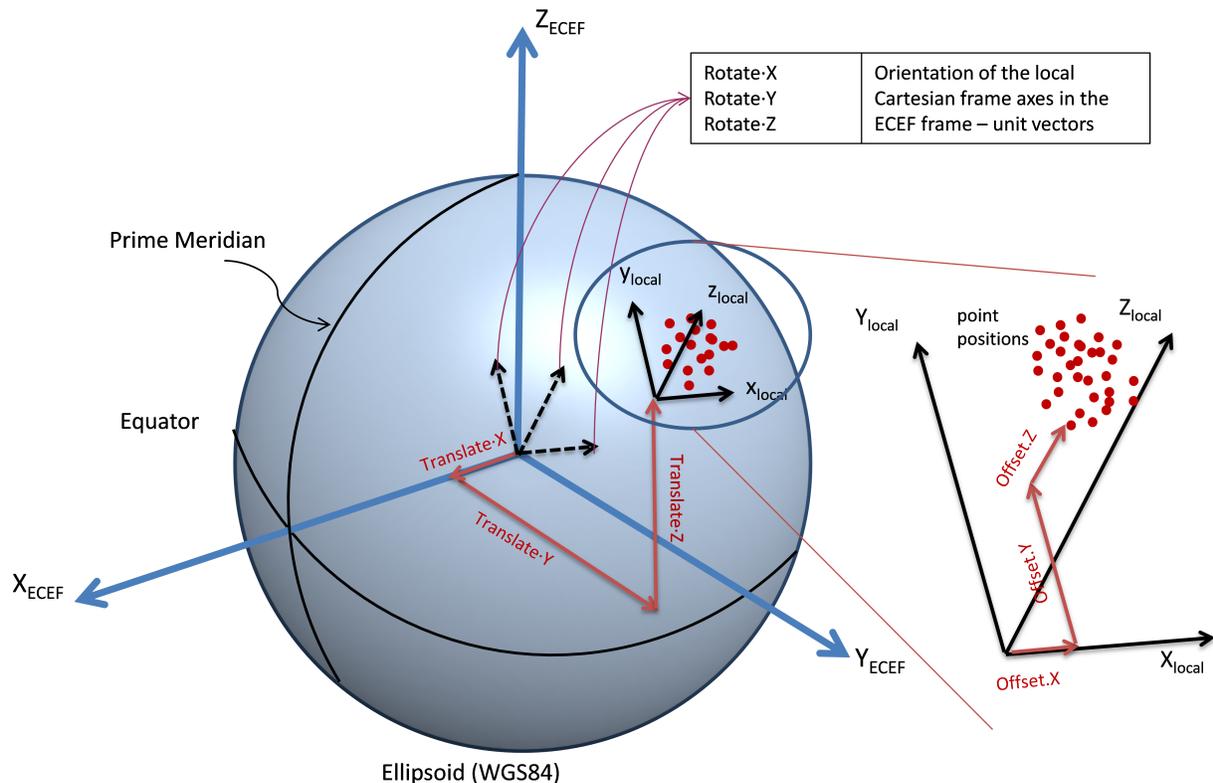


Figure 6: ECEF-Referenced Local Coordinate System.

POINTS.POSITION data is stored as unsigned integers and translated to (x, y, z) positions in meters through linear translations governed by the following metadata:

- MODALITY_a.CLOUD_c.TILEMAP.ECEFTranslate
 - Origin data: specifies the origin of the local Cartesian coordinate system in meters within the ECEF frame on a tile-by-tile basis (not used for UTM).
- MODALITY_a.CLOUD_c.TILEMAP.ECFRrotate
 - Alignment data: unit vectors that specify the direction of the local Cartesian coordinate systems in the ECEF frame on a tile-by-tile basis (not used for UTM).
- MODALITY_a.CLOUD_c.POINT.POSITIONS.Tile_d
 - Position: scaled integer representations of the point positions in the local frame.
- MODALITY_a.CLOUD_c.TILEMAP.Scale

- Scale metadata: provides scale factor (multiplier) applied to the integer position values in the same linear translation.
- MODALITY_a.CLOUD_c.TILEMAP.Offsets
 - Offset metadata: provides the intercept term in a linear translation of the integer position values to meters on a tile-by tile basis.

Once the location of each tile's origin is selected, a local Cartesian frame can be specified where the z-axis is in a direction normal to the ellipsoid surface at the origin location, the x-axis is in the east direction at the origin location, and the y-axis is in the north direction at the origin location. The correspondence of the z-axis to vertical, the x-axis to east, and the y-axis to north is only precisely true at the origin location. At locations away from the origin, the z-direction departs from the true local up direction, and x and y depart from true east and north direction, and this effect increases with distance from the origin. As long as the coordinate conversion is performed properly, this is not a source of error because the positions of the points are still properly placed in 3-D space without any projection distortion. In data sets of small spatial extent, this effect may not be appreciable. This is not a coordinate system where the points have been projected to a local tangent plane. Instead, all points are exactly placed in 3-D space without projection. For example, points that lie exactly on the ellipsoid surface (i.e. height above the ellipsoid is zero) will still vary in z and have a 3-D curvature in the shape of the ellipsoid, whereas in local East-North-Up (ENU) or a Universal Transverse Mercator (UTM) projection, when the ellipsoid is used as the elevation datum, these points would all lie on the x-y plane with a z value of zero.

To find the directions of the L3DC axes in ECEF space, let $F(X, Y, Z)$ be the equation of the WGS-84 ellipsoid in ECEF. The z-axis in local Cartesian space, z' , is defined as the vertical direction at the origin point (X_0, Y_0, Z_0) . The vertical direction is independent from the vertical elevation of the origin above or below the ellipsoid, and so can be determined at the surface. The surface normal is found by taking the divergence of the surface function and normalizing.

$$F(x, y, z) = \frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{b^2} \quad (1)$$

$$\vec{z}'(X_0, Y_0, Z_0) = \nabla F(X_0, Y_0, Z_0) = 2 \left(\frac{X_0}{a^2}, \frac{Y_0}{a^2}, \frac{Z_0}{b^2} \right) \quad (2)$$

$$\widehat{z}'(X_0, Y_0, Z_0) = \frac{\nabla F(X_0, Y_0, Z_0)}{\|\nabla F(X_0, Y_0, Z_0)\|} \quad (3)$$

Where a and b are the WGS-84 ellipsoid constants:

$$a = 6378137.0 \text{ m}$$

$$b = 6356752.314245 \text{ m}$$

The local Cartesian x-axis, pointing in the east direction at the origin, is orthogonal to both the local z-axis and the ECEF Z axis (i.e., it is parallel to the ECEF X-Y plane). Therefore, \mathbf{x}' , can be found by taking the cross product of the local vertical direction \mathbf{z}' with the negative ECEF Z direction. The negative ECEF Z direction is used to ensure it points in the direction of increasing longitude.

$$\widehat{\mathbf{x}}' = \frac{\widehat{\mathbf{z}}' \times (0, 0, -1)}{\|\widehat{\mathbf{z}}' \times (0, 0, -1)\|} \quad (4)$$

The local Cartesian y-axis in the north direction \mathbf{y}' , can then be found by taking the cross product of \mathbf{z}' and \mathbf{x}'

$$\widehat{\mathbf{y}}' = \frac{\widehat{\mathbf{z}}' \times \widehat{\mathbf{x}}'}{\|\widehat{\mathbf{z}}' \times \widehat{\mathbf{x}}'\|} \quad (5)$$

There are two cases where these cross products go to zero, namely if the origin is placed at precisely the north or south pole where the definitions of north and east become pathological. Conventions are established for MIPC and SIPC to deal with these cases. At the north pole, the ECEF X and Y directions will be used for the local x and y axes, respectively. The ECEF Z axis is already the local vertical. At the south pole the ECEF negative Z direction is local vertical, and the local Cartesian x axis will be in the positive ECEF X direction and our local Cartesian Y axis will be in the negative ECEF Y direction.

The vectors defining the local Cartesian axes directions in ECEF should now be stored in: MODALITY_a.CLOUD_c.TILEMAP.ECEFRotate

in accordance with Volume 1 of this standard. Specifically, the rotation matrix is an array for each tile with 3 rows and 3 columns, where columns $j = 1, 2, 3$ represents the unit vector's components in the ECEF x, y, z directions, respectively. Then the array for each tile is a page into a single 3-D array dataset **ECEFRotate**.

Similarly, the tile origin position is stored in:

MODALITY_a.CLOUD_c.TILEMAP.ECEFTranslate

in accordance with Volume 1 of this standard. Specifically, the coordinates are an array for each tile with 1 row and 3 columns, where columns $j = 1, 2, 3$ represents the ECEF x, y, z positions, respectively. Then the array for each tile is a page into a single 3-D array dataset **ECEFTranslate**.

3.5.1 Translate and Rotate Points

These unit vectors can now be used to rotate points between the local and ECEF frames. A 3 x 3 rotation matrix is constructed of the elements:

$$\mathbf{R} = \begin{bmatrix} \widehat{x}'_1 & \widehat{y}'_1 & \widehat{z}'_1 \\ \widehat{x}'_2 & \widehat{y}'_2 & \widehat{z}'_2 \\ \widehat{x}'_3 & \widehat{y}'_3 & \widehat{z}'_3 \end{bmatrix} \quad (6)$$

L3DC coordinates (x', y', z') are computed using equation 7 given the point's (X, Y, Z) coordinates in ECEF and the tiles' origin location (X_0, Y_0, Z_0) in ECEF coordinates:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}_{L3DC} = \mathbf{R}^{-1} \begin{bmatrix} X - X_0 \\ Y - Y_0 \\ Z - Z_0 \end{bmatrix}_{ECEF} \quad (7)$$

Recall that the origin position is stored in:

MODALITY_a.CLOUD_c.TILEMAP.ECEFTranslate

So the inverse rotation operator is simply:

$$\mathbf{R}^{-1} = \mathbf{R}^T = \begin{bmatrix} \widehat{x}'_1 & \widehat{x}'_2 & \widehat{x}'_3 \\ \widehat{y}'_1 & \widehat{y}'_2 & \widehat{y}'_3 \\ \widehat{z}'_1 & \widehat{z}'_2 & \widehat{z}'_3 \end{bmatrix} \quad (8)$$

3.5.2 Convert Position Data to Scaled Integer

Point cloud position data in MIPC is stored as unsigned integers and translated to (x, y, z) positions in meters through linear translations of the form:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}_{L3DC} = \begin{bmatrix} offset_x \\ offset_y \\ offset_z \end{bmatrix} + \begin{bmatrix} scale_x * x_{SI} \\ scale_y * y_{SI} \\ scale_z * z_{SI} \end{bmatrix} \quad (9)$$

$$x_{SI} = \left\lfloor \frac{x' - offset_x}{scale_x} + 0.5 \right\rfloor \quad (10)$$

$$offset_x = \min_{i \in N} \{x'_i\} \quad (11)$$

and similarly for y and z . The SI subscript denotes the actual scaled integer value stored in the file. Notice that x_{SI} is the *floor* of the function in equation 10. N is the number of points in the tile, and,

$$\{x_{SI} | x_{SI} \in \mathbb{Z}, 0 \leq x_{SI} \leq 2^b - 1\} \quad (12)$$

$$\{y_{SI} | y_{SI} \in \mathbb{Z}, 0 \leq y_{SI} \leq 2^b - 1\} \quad (13)$$

$$\{z_{SI} | z_{SI} \in \mathbb{Z}, 0 \leq z_{SI} \leq 2^b - 1\} \quad (14)$$

where $b = 16, 32,$ or 64 ; depending on the bit depth chosen for storage.

The offset terms provide the intercept in a linear translation of the integer position values to meters so $offset_x$ should be set to the minimum value of \mathbf{x}' found in the tile, and similarly for $offset_y$ and $offset_z$.

The scale factor is the multiplier applied to the integer position values in the same linear translation. This factor's minimum value is the maximum tile dimension in X, Y, or Z (across all tiles in the channel) divided by 2^{16} or 2^{32} but should be set considering the precision of the data so that meaningful precision is not lost, but artificial precision is not introduced.

These parameters are stored in the following data objects:

$offset_x = \text{MODALITY_a.CLOUD_c.TILEMAP.Offsets(d, 0)}$
 $offset_y = \text{MODALITY_a.CLOUD_c.TILEMAP.Offsets(d, 1)}$
 $offset_z = \text{MODALITY_a.CLOUD_c.TILEMAP.Offsets(d, 2)}$

$scale_x = \text{MODALITY_a.CLOUD_c.TILEMAP.Scale(d, 0)}$
 $scale_y = \text{MODALITY_a.CLOUD_c.TILEMAP.Scale(d, 1)}$
 $scale_z = \text{MODALITY_a.CLOUD_c.TILEMAP.Scale(d, 2)}$

$x_{SI} = \text{MODALITY_a.CLOUD_c.POINTS.POSITIONS.Tile.d.(p, 0)}$
 $y_{SI} = \text{MODALITY_a.CLOUD_c.POINTS.POSITIONS.Tile.d.(p, 1)}$
 $z_{SI} = \text{MODALITY_a.CLOUD_c.POINTS.POSITIONS.Tile.d.(p, 2)}$

where \mathbf{d} is the tile index and \mathbf{p} is the point record index.

3.6 Error Data

Error in geopositioning data is stored based on the level of detail in the error knowledge. In all cases, error is only relevant to a single **CLOUD**. At this time, there is no error model across clouds.

3.6.1 Global Error

In the most simplistic case (aside from having no error information), global error for a specific **CLOUD** is stored in:

MODALITY_a.CLOUD_c.QUALITY.GeoUncertainty

as a 1 x 3 array where the columns are:

Circular Error 90% (CE90)

Linear Error 90% (LE90)

Spherical Error 90% (SE90)

respectively.

3.6.2 GPM

MIPC implements the **GPM** as a method for storing and calculating error in geopositioning. If a complete **GPM** implementation is desired, this information is stored in the groups and datasets under:

MODALITY_a.CLOUD_c.GPM

in accordance with the **GPM** standard.

3.6.3 Per Point Error

The next level of detail and efficiency is the storage of error per point as generated from the **GPM**. This method stores covariance matrices in:

MODALITY_a.CLOUD_c.GPM.PPE

This group has a dataset, **n_Records**, indicating the number of error records, and an array dataset, **Covariance**, storing different covariance matrices. The **Covariance** dataset is a $n \times 6$ array, where n is the error index, which allows for the storage of less errors than points assuming that some errors are used for multiple points. The 6 columns are the variance in x , y , z along with the covariance in xy , xz , yz .

The error data associated with each point is stored in

MODALITY_a.CLOUD_c.POINTS.ERROR

as a single integer index into the table of errors under **GPM.PPE**. The data under the **ERROR** group is tiled, named as **Tile_d**, where **d** is the integer index identical to the same tile under the **POSITIONS** group. Within a tile dataset, each row corresponds to the point at the same row in the same **Tile_d** in the **POSITIONS** group.

4 Extending MIPC

This section provides information on adding or extending upon the MIPC standard. Although HDF5 is a flexible *file format*, a *standard*, such as MIPC, is not flexible by definition. Blind flexibility leads to interoperability challenges. However, MIPC is tailorable in a standard manner to meet the requirements of the implementation.

MIPC was designed to be tailorable in the following manner. The file is hierarchical so that each branch is self-sufficient. The MIPC standard defines the minimum required content, and standardizes optional information. The current structure should not be changed without a formal process. However, there is room to *add* information in the forms of groups and datasets as described in the following sections. Additionally, HDF *attributes*, text that describes a group or dataset, were intentionally left out of the standard to be available to be used as desired by programs and systems.

If a change is warranted to an existing group or dataset, including the **REFERENCES** group, a Request for Change (RFC) should be submitted. The advantage of HDF5 is that significant changes to the standard should only require minor modifications to code that reads or writes MIPC data.

4.1 Extending Groups

There is no single **USERDEFINED** group because there was a design goal to have each group be self-sufficient. So it is therefore permissible to add a group anywhere within the hierarchy. This is not a risky practice if the software is written properly. If software is intended to acquire all groups within a branch of the file, it should use the appropriate HDF API for blindly asking for members by number, or query the member names and then ask. If software is intended to acquire specific groups or datasets defined in the MIPC standard, then it should be explicitly written in that manner using these specifications as guides and requesting only for those items of interest.

4.2 Extending Datasets

An existing dataset should never be extended. If additional information is desired as an array or table; then a new, separate, dataset should be added. Extending the rows or columns of an existing dataset could induce interoperability issues.

4.3 Adding Imagery

There is no explicit group for imagery within a MIPC file, because imagery is not the intention of MIPC. However, as with SIPC, the advantages to including context imagery are well recognized. As stated in § 4.1, new groups could be added as desired to include context imagery without the need to specify them within the MIPC standard. However, it is recommended that the images be placed in the appropriate section in the hierarchy. For example, if

the images are overviews related to a specific CLOUD, then it should be included under that CLOUD. If they are quicklooks of the source imagery from a specific sensor, they should be included under LOOKS. These example paths are given below:

MODALITY_0.CLOUD_0.CONTEXT.Dem

MODALITY_0.SENSOR_0.LOOKS.QUICKLOOKS.image_0

References

- [1] NGA, “Sensor Independent Point Cloud (SIPC) File Format Trade Study,” Report, Nov 2012. Ver. 2.0.

Glossary

application programming interface a library of software source code or executables with supporting documentation that simplifies and encapsulates the interface to third party components or data in order to streamline programming. 2

CamelCase a syntax where words are combined without spaces or underscores, but the first letter of each word is uppercase, and all other letters are lowercase. The first letter may or may not be uppercase depending on the meaning and convention. 6

channel a specific sensor configuration that modulates the intensity data in ground-space. This will typically indicate a combination of the spectral band and the polarization state. 25

data generally refers to the input to some process or system. For systems engineering models of NGA CONOPS, this term refers to GEOINT signal content that have been processed and are ready for analyst exploitation. This data can be used as input to various analysis tasks. iii

dataset a data component within HDF used to store an array of data. 5

group a data component within HDF used to organize data, similar to a directory in a file system. 5

metadata generally refers to data about data (signal). This is information that refers to the sensor specifics, or the manner in which the data was collected. This is sometimes referred to as *Narrow Band Data* or *Support Data*. iii

Acronyms

2-D two-dimensional. iii, 23

3-D three-dimensional. iii, 25, 27

API Application Programming Interface. 2, 4, 6, 7

CE90 Circular Error 90%. 31

CMMD Conceptual Model and Metadata Dictionary. 5

ECEF Earth-Centered, Earth-Fixed. 23, 25

ENU East-North-Up. 27

EO Electro-Optical. iii

GPM Generic Point-Cloud Model. 6, 31

GWG Geospatial Intelligence Standards Working Group. 4

HAE Height Above Ellipsoid. 23

HDF5 Hierarchical Data Format 5. iii, 1, 2, 4

IDL Interactive Data Language. 7, 14

CLV Key-Length-Value. 1

L3DC Local 3-D Cartesian. 26

LE90 Linear Error 90%. 31

LIDAR Light Detection and Ranging. iii

MIPC Modality Independent Point Cloud. iii, 1, 2, 8

MPI Message Passing Interface. 4

NASA National Aeronautics and Space Administration. 4

NCSA National Center for Supercomputing Applications. 4

NGA National Geospatial-Intelligence Agency. iii, 1, 4

NMF NSG Metadata Foundation. 5, 6

NSG National System for Geospatial-Intelligence. iii, 1

RADAR Radio Detection and Ranging. iii

RFC Request for Change. 32

SE90 Spherical Error 90%. 31

SICD Sensor Independent Complex Data. 1

SIPC Sensor Independent Point Cloud. iii, 1, 2, 8

TB Terabytes. 4

UTM Universal Transverse Mercator. 27

WAMI Wide Area Motion Imagery. iii