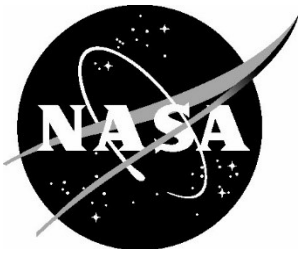
NASA/TM-2019-220402



Microphone Phased Array NetCDF / HDF5 Archival Files
Application Program Interface Reference

William M. Humphreys, Jr.
Langley Research Center, Hampton, Virginia

September 2019

NASA STI Program . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA scientific and technical information (STI) program plays a key part in helping NASA maintain this important role.

The NASA STI program operates under the auspices of the Agency Chief Information Officer. It collects, organizes, provides for archiving, and disseminates NASA's STI. The NASA STI program provides access to the NASA Aeronautics and Space Database and its public interface, the NASA Technical Report Server, thus providing one of the largest collections of aeronautical and space science STI in the world. Results are published in both non-NASA channels and by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services also include creating custom thesauri, building customized databases, and organizing and publishing research results.

For more information about the NASA STI program, see the following:

- Access the NASA STI program home page at *http://www.sti.nasa.gov*

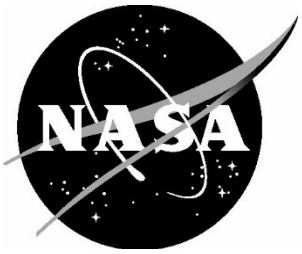
- E-mail your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at 443-757-5803

- Phone the NASA STI Help Desk at 443-757-5802

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7115 Standard Drive
  Hanover, MD 21076-1320

NASA/TM-2019-220402



Microphone Phased Array NetCDF / HDF5 Archival Files
Application Program Interface Reference

William M. Humphreys, Jr.
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

September 2019

Available from:

NASA Center for AeroSpace Information
7115 Standard Drive
Hanover, MD 21076-1320
443-757-5802

Contents

Contents

1.0 Abstract

An application program interface (API) has been developed for the creation and access of structured data files generated by microphone phased arrays utilized in aeroacoustics research. Two structured binary file formats are supported, namely NetCDF (Network Common Data Form) and HDF5 (Hierarchical Data Format) files. The API consists of a library of routines callable from C, Fortran or Matlab, with native versions of the API provided for each language. The libraries are divided into categories for file handling, file definition and initialization, data writing, data recovery, and error handling. The API is intended to provide a mechanism for generating self-describing binary files for long-term archiving of raw and processed data generated by phased array systems.

2.0 Introduction

Microphone phased arrays for aeroacoustics research generate copious amounts of raw time history and associated processed data. These data files traditionally have been stored in minimally structured (and minimally documented) binary files, requiring long-term knowledge of the file structure in order to retrieve the information contained within the files. Over time, there is a risk that knowledge of the file format may be lost, requiring possibly significant time and expense to be incurred to retrieve the information. Thus, it is desirable to store the data in files based on industry standards using self-describing formats. Self-describing file formats refer to files containing either separate header sections containing metadata describing the information in the file, or files exhibiting a hierarchical structure similar to that of a computer hard drive folder system.

There are two currently popular self-describing file formats that are well-suited for the long-term storage of data from phased arrays. These include the Network Common Data Form (NetCDF), developed by the University Corporation for Atmospheric Research (UCAR) [1], and the Hierarchical Data Format Version 5 (HDF5), originally developed by the National Center for Supercomputing Applications and now supported by the HDF Group [2]. Both file formats are extensively supported and utilized by researchers in academia and Government, especially in the atmospheric sciences. To provide maximum flexibility in the generation of future phased array datasets, it was decided to implement a custom application program interface (API) for both of these file formats. The API has been designed to be callable from C, Fortran or Matlab, with native implementations of the API provided for each language. This technical memorandum provides a reference manual for the API, describing how the data is stored in the files and showing the syntax of the API subroutine and function calls. Methods are shown for utilizing the routines to both create archival files and to store and retrieve data from them.

3.0 Implementation of the API

The API has been designed to be callable from C, Fortran 90, or Matlab. The following compiler environments and test platforms were used for the implementation:

Table 1. API Implementation Details

API Implementation	Compiler Environment	Platforms Tested
C	Intel XE Composer C++ Compiler	Windows 10, CentOS Linux
Fortran	Intel XE Composer Fortran Compiler	Windows 10, CentOS Linux
Matlab	Matlab Release R2019a	Windows 10

Additionally, the following NetCDF and HDF5 library versions were used for the builds:

Table 2. Library Versions Used in Build

Library	Version
NetCDF	3.4
HDF5	1.8

Although not tested, it is anticipated that the API source codes should be compatible with versions of the NetCDF and HDF5 libraries later than those shown in Table 2.

4.0 Overview of API Routines

The API has been developed to support four different types of archival data files that are associated with the use of phased arrays in either ground test or flight applications. These archival files are:

1. *Individual raw data file*, where the file contains a time history record for a single data channel in the array,
2. *Ensemble raw data file*, where the file contains time history records for all of the data channels in the array,
3. *Cross spectral matrix data file*, where the file contains the full cross spectral matrix generated from a set of time history data acquired with the array, and
4. *Static pressure data file*, where the file contains ancillary data regarding static pressures on models and structures acquired as part of an aeroacoustics study.

Detailed descriptions of the data structures for these various file types can be found in Appendices A, B, and C.

4.1 Data File Creation

Figure 1 depicts a flowchart outlining a typical process for creating one of the files listed in Section 4.0. The basic procedure is to populate the relevant data structures that will form the core of the data file, followed by executing calls to the API creating the file, populating the metadata in the file from the defined data structures, and then writing the relevant data arrays (time history, cross spectral matrix, or pressure data) to the file. The data structures that are used to populate the metadata in the file fall into three categories:

1. *Fixed header values* – these define common invariant parameters describing the data in the file (e.g., sensor information, calibration values, data acquisition parameters, etc.).
2. *Facility header values* – these define dynamic parameters associated with the ground test or flight facility that can change from file to file. Up to 200 unique facility header values can be defined in the file.
3. *Model header values* – these define dynamic parameters associated with the model under test that can change from file to file. Up to 200 unique model header values can be defined in the file.

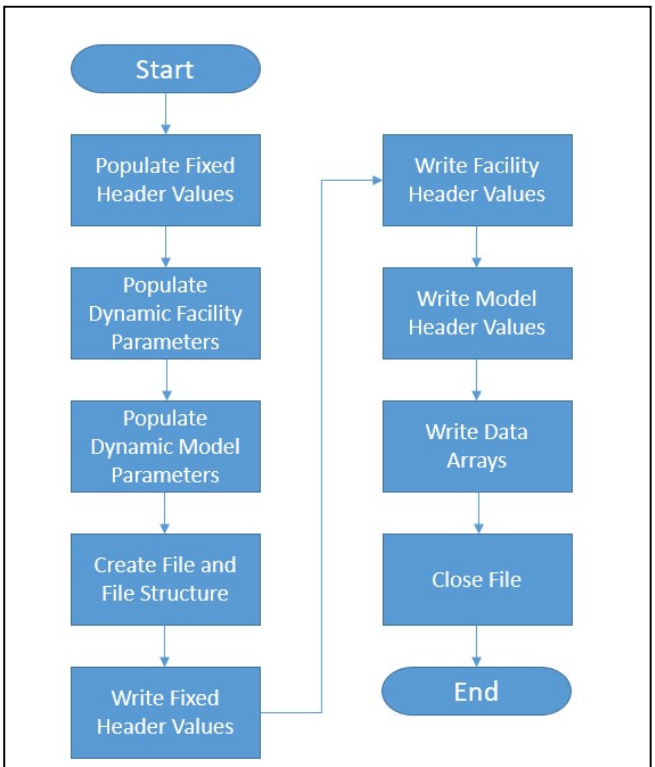


Figure 1. File Creation Process.

4.2 Data File Interrogation

Figure 2 depicts a flowchart outlining the typical process for opening and reading the data from a file. The basic procedure consists of executing calls to the API to open the file, read the metadata in the file, optionally transfer the metadata to external data structures if needed, and then read the relevant data arrays (time history, cross spectral matrix, or pressure data) from the file.

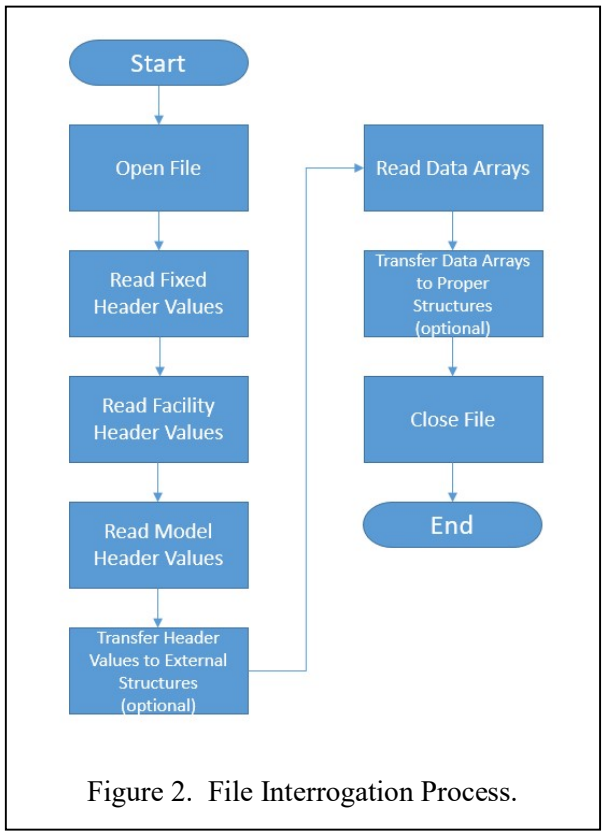


Figure 2. File Interrogation Process.

5.0 Adding Facility and Model Information

The metadata contained within the file headers are structured to allow the capability of dynamically adding and modifying facility- and model-specific information at run time as data files are created. There are two data structures included in the headers that are used for this purpose:

Data structure for facility-specific information:

facility_data.facility_parameter(*n*).long_name	Name of parameter
facility_data.facility_parameter(*n*).description	Detailed description of parameter
facility_data.facility_parameter(*n*).data_type	Parameter data type
facility_data.facility_parameter(*n*).units	Parameter units
facility_data.facility_parameter(*n*).value	Parameter value

where *n* is the parameter number

Data structure for model-specific information:

model_data.model_parameter(*n*).long_name	Name of parameter
model_data.model_parameter(*n*).description	Detailed description of parameter
model_data.model_parameter(*n*).data_type	Parameter data type
model_data.model_parameter(*n*).units	Parameter units
model_data.model_parameter(*n*).value	Parameter value

where *n* is the parameter number

A total of 200 separate facility and model parameter entries can be added to the header. For each parameter entry, there are five attributes that can be added supplying information regarding the parameter name, data type, units, and value. Each of the attributes is a character string with a maximum length of 80 characters.

As an example, to add a facility Mach number entry to a file as facility parameter #1, the data structure would be configured as follows (assuming Matlab syntax):

facility_data.facility_parameter(1).long_name = 'mach';
facility_data.facility_parameter(1).description = 'Facility Mach Number';
facility_data.facility_parameter(1).data_type = 'single precision real';
facility_data.facility_parameter(1).units = 'dimensionless';
facility_data.facility_parameter(1).value = '0.24';

As another example, to add a model angle of attack (AoA) entry to a file as model parameter #3, the data structure would be configured as follows (assuming Fortran 90 syntax):

model_data%model_parameter(3)%long_name = "AoA"
model_data%model_parameter(3)%description = "Model Angle of Attack"
model_data%model_parameter(3)%data_type = "single precision real"
model_data%model_parameter(3)%units = "degrees"
model_data%model_parameter(3)%value = "5.0"

Note that facility and model parameters are sequentially numbered from 1 to 200. Although not required, it is recommended that the parameters be listed contiguously in the file (i.e., numbered from 1 to the maximum number of parameters in the file without skipping over numbers).

6.0 Adding Static Tap Information

The manner for adding static tap information to pressure data files is similar to what is shown for adding facility and model parameter entries in Section 5.0. The data structure for pressure tap data is as follows:

static_data.static_tap_parameter(*n*).static_tap_designation	Static tap designation
static_data.static_tap_parameter(*n*).description	Static tap description
static_data.static_tap_parameter(*n*).units	Static tap pressure value units
static_data.static_tap_parameter(*n*).coefficient_name	Static tap coefficient name
static_data.static_tap_parameter(*n*).data_value	Static tap pressure value

where *n* is the static tap number

A total of 999 separate static tap values can be added to a pressure data file. For each static tap entry, there are five attributes that can be added supplying information regarding the tap designation, description, units, coefficient name, and value. Each of the attributes is a character string with a maximum length of 80 characters.

As an example, to add information for static pressure tap #55 to a pressure file, the data structure would be configured as follows (assuming Matlab syntax):

static_data.static_tap_parameter(55).static_tap_designation = 'ESP0215';
static_data.static_tap_parameter(55).description = 'ESP pressure port';
static_data.static_tap_parameter(55).units = 'psi';
static_data.static_tap_parameter(55).coefficient_name = 'ESP0215';
static_data.static_tap_parameter(55).value = '0.104827';

Note that static tap parameters are sequentially numbered from 1 to 999. Although not required, it is recommended that the parameters be listed contiguously in the file (i.e., numbered from 1 to the maximum number of static taps in the file without skipping over numbers).

7.0 API Unit Tests

Unit tests have been written in native C, Fortran, and Matlab in order to test each of the subroutines and functions comprising the API. The unit tests populate a given file format data structure with synthetic data, generate the appropriate data file, read the information from the file, and then compare what was read from the file with the original synthetic data fields that were generated. Any mismatches in the generated data and data returned from the file are flagged.

In addition to exercising the API subroutines and functions, the unit tests also provide the user with a series of detailed examples of how to use the API's to generate the various file formats.

8.0 NetCDF Function Reference

List of C / Matlab NetCDF API Routines by Category

Error Handling
- nc_issue_netcdf_error

File Handling
- nc_open_raw_file_v9
- nc_open_pressure_file_v9
- nc_open_csm_file_v9
- nc_close_raw_file_v9
- nc_close_pressure_file_v9
- nc_close_csm_file_v9

File Definition / Initialization
- nc_define_individual_raw_file_v9
- nc_define_ensemble_raw_file_v9
- nc_define_pressure_file_v9
- nc_define_csm_file_v9

Data Writing
- nc_write_individual_raw_fixed_header_v9
- nc_write_ensemble_raw_fixed_header_v9
- nc_write_pressure_fixed_header_v9
- nc_write_csm_fixed_header_v9
- nc_write_facility_header_v9
- nc_write_model_header_v9
- nc_write_individual_raw_data_v9
- nc_write_ensemble_raw_data_v9
- nc_write_pressure_data_v9
- nc_write_csm_data_v9

Data Reading
- nc_read_individual_raw_fixed_header_v9
- nc_read_ensemble_raw_fixed_header_v9
- nc_read_pressure_fixed_header_v9
- nc_read_csm_fixed_header_v9
- nc_read_facility_header_v9
- nc_read_model_header_v9
- nc_read_individual_raw_data_v9
- nc_read_ensemble_raw_data_v9
- nc_read_pressure_data_v9
- nc_read_csm_data_v9

List of Fortran NetCDF API Routines by Category

Error Handling
- nf_issue_netcdf_error

File Handling
- nf_open_raw_file_v9
- nf_open_pressure_file_v9
- nf_open_csm_file_v9
- nf_close_raw_file_v9
- nf_close_pressure_file_v9
- nf_close_csm_file_v9

File Definition / Initialization
- nf_define_individual_raw_file_v9
- nf_define_ensemble_raw_file_v9
- nf_define_pressure_file_v9
- nf_define_csm_file_v9

Data Writing
- nf_write_individual_raw_fixed_header_v9
- nf_write_ensemble_raw_fixed_header_v9
- nf_write_pressure_fixed_header_v9
- nf_write_csm_fixed_header_v9
- nf_write_facility_header_v9
- nf_write_model_header_v9
- nf_write_individual_raw_data_v9
- nf_write_ensemble_raw_data_v9
- nf_write_pressure_data_v9
- nf_write_csm_data_v9

Data Reading
- nf_read_individual_raw_fixed_header_v9
- nf_read_ensemble_raw_fixed_header_v9
- nf_read_pressure_fixed_header_v9
- nf_read_csm_fixed_header_v9
- nf_read_facility_header_v9
- nf_read_model_header_v9
- nf_read_individual_raw_data_v9
- nf_read_ensemble_raw_data_v9
- nf_read_pressure_data_v9
- nf_read_csm_data_v9

issue_netcdf_error

Purpose	This routine implements a general NetCDF error handler.
C Syntax	nc_issue_netcdf_error(*operation*,*status*)
Fortran Syntax	nf_issue_netcdf_error(*lu,operation,status*)
Matlab Syntax	nc_issue_netcdf_error(*operation,err*)
Input Arguments	*lu* Fortran logical unit number for output message (integer) *operation* NetCDF operation attempted (string) *status* Error status returned from operation (integer) *err* Error returned from operation (string)
Output Arguments	*none*
Remarks	Any output error messages are displayed on the console screen.

Example Use:

nc_issue_netcdf_error("open_raw_file",status);	*C*
call nf_issue_netcdf_error(lu,'open_raw_file',status)	*Fortran*
nc_issue_netcdf_error('open_raw_file','file not found');	*Matlab*

Purpose	This routine opens an existing NetCDF individual or ensemble raw data file for access.
C Syntax	*ncid* = nc_open_raw_file_v9(*name*, *mode*)
Fortran Syntax	nf_open_raw_file_v9(*name, ncid, mode*)
Matlab Syntax	*ncid* = nc_open_raw_file_v9(*name, mode*)
Input Arguments	*name* NetCDF file name to open (string) *mode* File access mode (string) 'r','R' = read only 'w','W' = read / write
Output Arguments	*ncid* NetCDF file ID number (integer)
Remarks	An error will be returned if the *mode* is 'r' and the file does not exist.

Example Use:

ncid = nc_open_raw_file_v9("T0001R0001P0001C0001.nc",'w');	*C*
call nf_open_raw_file_v9('T0001R0001P0001C0001.nc',ncid,'w')	*Fortran*
ncid = nc_open_raw_file_v9('T0001R0001P0001C0001.nc','w');	*Matlab*

open_pressure_file_v9

Purpose	This routine opens an existing NetCDF static pressure data file for access.
C Syntax	*ncid* = nc_open_pressure_file_v9(*name*, *mode*)
Fortran Syntax	nf_open_pressure_file_v9(*name, ncid, mode*)
Matlab Syntax	*ncid* = nc_open_pressure_file_v9(*name, mode*)
Input Arguments	*name* NetCDF file name to open (string) *mode* File access mode (string) 'r', 'R' = read only 'w', 'W' = read / write
Output Arguments	*ncid* NetCDF file ID number (integer)
Remarks	An error will be returned if the *mode* is 'r' and the file does not exist.

Example Use:

ncid = nc_open_pressure_file_v9("T0001R0001P0001_pressure.nc",'w');	*C*
call nf_open_pressure_file_v9('T0001R0001P0001_pressure.nc',ncid,'w')	*Fortran*
ncid = nc_open_pressure_file_v9('T0001R0001P0001_pressure.nc','w');	*Matlab*

Purpose	This routine opens an existing NetCDF cross spectral matrix data file for access.
C Syntax	*ncid* = nc_open_csm_file_v9(*name*, *mode*)
Fortran Syntax	nf_open_csm_file_v9(*name, ncid, mode*)
Matlab Syntax	*ncid* = nc_open_csm_file_v9(*name, mode*)
Input Arguments	*name* NetCDF file name to open (string) *mode* File access mode (string) 'r', 'R' = read only 'w'. 'W' = read / write
Output Arguments	*ncid* NetCDF file ID number (integer)
Remarks	An error will be returned if the *mode* is 'r' and the file does not exist.

Example Use:

ncid = nc_open_csm_file_v9("T0001R0001P0001_CSM.nc",'w');	*C*
call nf_open_csm_file_v9('T0001R0001P0001_CSM.nc',ncid,'w')	*Fortran*
ncid = nc_open_csm_file_v9('T0001R0001P0001_CSM.nc','w');	*Matlab*

close_raw_file_v9

Purpose	This routine closes an open NetCDF individual or ensemble raw data file.
C Syntax	nc_close_raw_file_v9(*ncid*)
Fortran Syntax	nf_close_raw_file_v9(*ncid*)
Matlab Syntax	nc_close_raw_file_v9(*ncid*)
Input Arguments	*ncid* NetCDF file ID number (integer)
Output Arguments	none
Remarks	An error will be returned if the file is not currently open.

Example Use:

nc_close_raw_file_v9(ncid);	*C*
call nf_close_raw_file_v9(ncid)	*Fortran*
nc_close_raw_file_v9(ncid);	*Matlab*

close_pressure_file_v9

Purpose	This routine closes an open NetCDF static pressure data file.
C Syntax	nc_close_pressure_file_v9(*ncid*)
Fortran Syntax	nf_close_pressure_file_v9(*ncid*)
Matlab Syntax	nc_close_pressure_file_v9(*ncid*)
Input Arguments	*ncid* NetCDF file ID number (integer)
Output Arguments	none
Remarks	An error will be returned if the file is not currently open.

Example Use:

nc_close_pressure_file_v9(ncid);	*C*
call nf_close_pressure_file_v9(ncid)	*Fortran*
nc_close_pressure_file_v9(ncid);	*Matlab*

close_csm_file_v9

Purpose	This routine closes an open NetCDF cross spectral matrix data file.
C Syntax	nc_close_csm_file(*ncid*)
Fortran Syntax	nf_close_csm_file_v9(*ncid*)
Matlab Syntax	nc_close_csm_file_v9(*ncid*)
Input Arguments	*ncid* NetCDF file ID number (integer)
Output Arguments	none
Remarks	An error will be returned if the file is not currently open.

Example Use:

nc_close_csm_file_v9(ncid);	*C*
call nf_close_csm_file_v9(ncid)	*Fortran*
nc_close_csm_file_v9(ncid);	*Matlab*

define_individual_raw_file_v9

Purpose	This routine opens and defines a new NetCDF individual raw data file.
C Syntax	*ncid* = nc_define_individual_raw_file_v9(*name, number_of_sources, number_of_bins, number_of_inclinometers, number_of_samples*)
Fortran Syntax	nf_define_individual_raw_file_v9(*name, number_of_sources, number_of_bins, number_of_inclinometers, number_of_samples*, *ncid*)
Matlab Syntax	*ncid* = nc_define_individual_raw_file_v9(*name, number_of_sources, number_of_bins, number_of_inclinometers, number_of_samples*)
Input Arguments	*name* Name of NetCDF individual raw data file to create (string) *number_of_sources* Number of individual calibration sources used with array (integer) *number_of_bins* Number of cross spectral matrix frequency bins (integer) *number_of_inclinometers* Number of inclinometers incorporated into array (integer) *number_of_samples* Number of time series samples acquired in file (integer)
Output Arguments	*ncid* NetCDF file ID number (integer)
Remarks	An error will be returned if the file already exists.

Example Use:

C

```
ncid = nc_define_individual_raw_file_v9("T0001R0001P0001C0001.nc",8,4096,3,1500000);
```

Fortran

```
call nf_define_individual_raw_file_v9('T0001R0001P0001C0001.nc',8,4096,3,1500000,ncid)
```

Matlab

```
ncid = nc_define_individual_raw_file_v9('T0001R0001P0001C0001.nc',8,4096,3,1500000);
```

Purpose — This routine opens and defines a new NetCDF ensemble raw data file.

C Syntax — *ncid* = nc_define_ensemble_raw_file_v9(*name, number_of_channels, number_of_sources, number_of_cals, number_of_bins, number_of_inclinometers, number_of_samples*)

Fortran Syntax — nf_define_ensemble_raw_file_v9(*name, number_of_channels, number_of_sources, number_of_cals, number_of_bins, number_of_inclinometers, number_of_samples*, *ncid*)

Matlab Syntax — *ncid* = nc_define_ensemble_raw_file_v9(*name, number_of_channels, number_of_sources, number_of_cals, number_of_bins, number_of_inclinometers, number_of_samples*)

Input Arguments

name
Name of NetCDF ensemble raw data file to create (string)

number_of_channels
Number of channels stored in data file (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

number_of_samples
Number of time series samples acquired in file (integer)

Output Arguments	*ncid* NetCDF file ID number (integer)
Remarks	An error will be returned if the file already exists.

Example Use:

C

```
ncid = nc_define_ensemble_raw_file_v9("T0001R0001P0001.nc",97,8,10,4096,3,1500000);
```

Fortran

```
call nf_define_ensemble_raw_file_v9('T0001R0001P0001.nc',97,8,10,4096,3,1500000,ncid)
```

Matlab

```
ncid = nc_define_ensemble_raw_file_v9('T0001R0001P0001.nc',97,8,10,4096,3,1500000);
```

define_pressure_file_v9

Purpose	This routine opens and defines a new NetCDF static pressure tap raw data file.
C Syntax	*ncid* = nc_define_pressure_file_v9(*name*)
Fortran Syntax	nf_define_pressure_file_v9(*name, ncid*)
Matlab Syntax	*ncid* = nc_define_pressure_file_v9(*name*)
Input Arguments	*name* Name of NetCDF static pressure tap data file to create (string)
Output Arguments	*ncid* NetCDF file ID number (integer)
Remarks	An error will be returned if the file already exists.

Example Use:

C

```
ncid = nc_define_pressure_file_v9("T0001R0001P0001_pressure.nc");
```

Fortran

```
call nf_define_pressure_file_v9('T0001R0001P0001_pressure.nc',ncid)
```

Matlab

```
ncid = nc_define_pressure_file_v9('T0001R0001P0001_pressure.nc');
```

Purpose

This routine opens and defines a new NetCDF cross spectral matrix data file.

C Syntax

ncid = nf_define_csm_file_v9(*name, number_of_channels, number_of_bins, number_of_sources, number_of_cals, number_of_inclinometers, number_of_blocks*)

Fortran Syntax

nf_define_csm_file_v9(*name, number_of_channels, number_of_bins, number_of_sources, number_of_cals, number_of_inclinometers, number_of_blocks*, *ncid*)

Matlab Syntax

ncid = nf_define_csm_file_v9(*name, number_of_channels, number_of_bins, number_of_sources, number_of_cals, number_of_inclinometers, number_of_blocks*)

Input Arguments

name
Name of NetCDF cross spectral matrix data file to create (string)

number_of_channels
Number of channels stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in data file (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

number_of_blocks
Number of FFT blocks used to create cross spectral matrix (integer)

Output Arguments *ncid*
NetCDF file ID number (integer)

Remarks An error will be returned if the file already exists.

Example Use:

C

ncid = nc_define_csm_file_v9("T0001R0001P0001_CSM.nc",97,4096,8,10,3,300);

Fortran

call nf_define_csm_file_v9('T0001R0001P0001_CSM.nc',97,4096,8,10,3,300,ncid)

Matlab

ncid = nc_define_csm_file_v9('T0001R0001P0001_CSM.nc',97,4096,8,10,3,300);

Purpose — This routine populates an individual raw data file header with fixed (non-facility and non-model) entries.

C Syntax

```
nc_write_individual_raw_fixed_header_v9(ncid,
                                        version_data,
                                        test_data,
                                        channel_data,
                                        calibration_data,
                                        das_data,
                                        array_data,
                                        number_of_sources,
                                        number_of_bins,
                                        number_of_inclinometers)
```

Fortran Syntax

```
nf_write_individual_raw_fixed_header_v9(ncid,
                                        version_data,
                                        test_data,
                                        channel_data,
                                        calibration_data,
                                        das_data,
                                        array_data,
                                        number_of_sources,
                                        number_of_bins,
                                        number_of_inclinometers)
```

Matlab Syntax

```
nc_write_individual_raw_fixed_header_v9(ncid,
                                        version_data,
                                        test_data,
                                        channel_data,
                                        calibration_data,
                                        das_data,
                                        array_data,
                                        number_of_sources,
                                        number_of_bins,
                                        number_of_inclinometers)
```

Input Arguments

ncid
NetCDF file ID number (integer)

version_data
Data structure containing version data in header
(See Appendices A, B, and C for format)

test_data
Data structure containing test data in header

channel_data
Data structure containing channel data in header

calibration_data
Data structure containing sensor calibration data in header

das_data
Data structure containing DAS data in header

array_data
Data structure containing array data in header

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

Output Arguments none

Remarks The file must first be opened with a call to *open_raw_file_v9* before calling this routine.

Example Use:

C

```
nc_write_individual_raw_fixed_header_v9(ncid,version_data,test_data,channel_data,
calibration_data,das_data,array_data,8,4096,3);
```

Fortran

```
call nf_write_individual_raw_fixed_header_v9(ncid,version_data,test_data,channel_data,
calibration_data,das_data,array_data,8,4096,3)
```

Matlab

```
nc_write_individual_raw_fixed_header_v9(ncid,version_data,test_data,channel_data,
calibration_data,das_data,array_data,8,4096,3);
```

write_ensemble_raw_fixed_header_v9

Purpose	This subroutine populates an ensemble raw data file header with fixed (non-facility and non-model) entries.
C Syntax	nc_write_ensemble_raw_fixed_header_v9(*ncid, ensemble_version_data, ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data, ensemble_array_data, number_of_channels, number_of_sources, number_of_cals, number_of_bins, number_of_inclinometers*)
Fortran Syntax	nf_write_ensemble_raw_fixed_header_v9(*ncid, ensemble_version_data, ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data, ensemble_array_data, number_of_channels, number_of_sources, number_of_cals, number_of_bins, number_of_inclinometers*)
Matlab Syntax	nc_write_ensemble_raw_fixed_header_v9(*ncid, ensemble_version_data, ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data, ensemble_array_data, number_of_channels, number_of_sources, number_of_cals, number_of_bins, number_of_inclinometers*)

Input Arguments

ncid
NetCDF file ID number (integer)

ensemble_version_data
Data structure containing ensemble version data in header
(see Appendices A, B, and C for format)

ensemble_test_data
Data structure containing ensemble test data in header

ensemble_channel_data
Data structure containing ensemble channel data in header

ensemble_calibration_data
Data structure containing ensemble sensor calibration data in header

ensemble_das_data
Data structure containing ensemble DAS data in header

ensemble_array_data
Data structure containing ensemble array data in header

number_of_channels
Number of channels stored in data file (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

Output Arguments

none

Remarks

The file must first be opened with a call to *open_raw_file_v9* before calling this routine.

Example Use:

C

nc_write_ensemble_raw_fixed_header_v9(ncid, ensemble_version_data, ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data, ensemble_array_data, 97, 8, 10, 4096, 3);

Fortran

call nf_write_ensemble_raw_fixed_header_v9(ncid, ensemble_version_data, ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data, ensemble_array_data, 97, 8, 10, 4096, 3)

Matlab

nc_write_ensemble_raw_fixed_header_v9(ncid, ensemble_version_data, ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data, ensemble_array_data, 97, 8, 10, 4096, 3);

Purpose	This subroutine populates a static pressure data file header with fixed (non-facility and non-model) entries.
C Syntax	nc_write_pressure_fixed_header_v9(*ncid, version_data, test_data, static_data*)
Fortran Syntax	nf_write_pressure_fixed_header_v9(*ncid, version_data, test_data, static_data*)
Matlab Syntax	nc_write_pressure_fixed_header_v9(*ncid, version_data, test_data, static_data*)
Input Arguments	*ncid* NetCDF file ID number (integer) *version_data* Data structure containing version data in header (see Appendices A, B, and C for format) *test_data* Data structure containing test data in header *static_data* Data structure containing static pressure header data
Output Arguments	none
Remarks	The file must first be opened with a call to *open_pressure_file_v9* before calling this routine.

Example Use:

C

```
nc_write_pressure_fixed_header_v9(ncid,version_data,test_data,static_data);
```

Fortran

```
call nf_write_pressure_fixed_header_v9(ncid,version_data,test_data,static_data)
```

Matlab

```
nc_write_pressure_fixed_header_v9(ncid,version_data,test_data,static_data);
```

Purpose — This subroutine populates a cross spectral matrix data file header with fixed (non-facility and non-model) entries.

C Syntax

```
nc_write_csm_fixed_header_v9(ncid,
                             csm_version_data,
                             csm_test_data,
                             csm_channel_data,
                             csm_calibration_data,
                             csm_das_data,
                             csm_array_data,
                             csm_data,
                             number_of_channels,
                             number_of_bins,
                             number_of_sources,
                             number_of_cals,
                             number_of_inclinometers,
                             number_of_blocks)
```

Fortran Syntax

```
nf_write_csm_fixed_header_v9(ncid,
                             csm_version_data,
                             csm_test_data,
                             csm_channel_data,
                             csm_calibration_data,
                             csm_das_data,
                             csm_array_data,
                             csm_data,
                             number_of_channels,
                             number_of_bins,
                             number_of_sources,
                             number_of_cals,
                             number_of_inclinometers,
                             number_of_blocks)
```

Matlab Syntax

nc_write_csm_fixed_header_v9(*ncid,*
csm_version_data,
csm_test_data,
csm_channel_data,
csm_calibration_data,
csm_das_data,
csm_array_data,
csm_data,
number_of_channels,
number_of_bins,
number_of_sources,
number_of_cals,
number_of_inclinometers,
number_of_blocks)

Input Arguments

ncid
NetCDF file ID number (integer)

csm_version_data
Data structure containing version data in header
(see Appendices A, B, and C for format)

csm_test_data
Data structure containing test data in header

csm_channel_data
Data structure containing channel data in header

csm_calibration_data
Data structure containing sensor calibration data in header

csm_das_data
Data structure containing DAS data in header

csm_array_data
Data structure containing array data in header

csm_data
Data structure containing csm data in header

number_of_channels
Number of channels stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in file (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

number_of_blocks
Number of FFT blocks used to create cross spectral matrix (integer)

Output Arguments none

Remarks The file must first be opened with a call to *open_csm_file_v9* before calling this routine.

Example Use:

C

```
nc_write_csm_fixed_header_v9(ncid,csm_version_data,csm_test_data,csm_channel_data,
csm_calibration_data,csm_das_data,csm_array_data,csm_data,97,4096,8,10,3,100);
```

Fortran

```
call nf_write_csm_fixed_header_v9(ncid,csm_version_data,csm_test_data,csm_channel_data,
csm_calibration_data,csm_das_data,csm_array_data,csm_data,97,4096,8,10,3,100)
```

Matlab

```
nc_write_csm_fixed_header_v9(ncid,csm_version_data,csm_test_data,csm_channel_data,
csm_calibration_data,csm_das_data,csm_array_data,csm_data,97,4096,8,10,3,100);
```

Purpose	This subroutine populates an individual raw / ensemble raw / static pressure / cross spectral matrix data file header with facility-specific entries.
C Syntax	nc_write_facility_header_v9(*ncid, facility_data*)
Fortran Syntax	nf_write_facility_header_v9(*ncid, facility_data*)
Matlab Syntax	nc_write_facility_header_v9(*ncid, facility_data*)
Input Arguments	*ncid* NetCDF file ID number (integer) *facility_data* Data structure containing facility data in header (see Appendices A, B, and C for format)
Output Arguments	none
Remarks	The file must first be opened with a call to *open_raw_file_v9, open_pressure_file_v9,* or *open_csm_file_v9* before calling this routine.

Example Use:

nc_write_facility_header_v9(ncid, facility_data);	*C*
call nf_write_facility_header_v9(ncid, facility_data)	*Fortran*
nc_write_facility_header_v9(ncid, facility_data);	*Matlab*

Purpose	This subroutine populates an individual raw / ensemble raw / static pressure / cross spectral matrix data file header with model-specific entries.
C Syntax	nc_write_model_header_v9(*ncid, model_data*)
Fortran Syntax	nf_write_model_header_v9(*ncid, model_data*)
Matlab Syntax	nc_write_model_header_v9(*ncid, model_data*)
Input Arguments	*ncid* NetCDF file ID number (integer) *model_data* Data structure containing model data in header (see Appendices A, B, and C for format)
Output Arguments	none
Remarks	The file must first be opened with a call to *open_raw_file_v9, open_pressure_file_v9,* or *open_csm_file_v9* before calling this routine.

Example Use:

nc_write_model_header_v9(ncid, model_data);	*C*
call nf_write_model_header_v9(ncid, model_data)	*Fortran*
nc_write_model_header_v9(ncid, model_data);	*Matlab*

write_individual_raw_data_v9

Purpose	This subroutine writes a block of time history data to an individual raw data file.
C Syntax	nc_write_individual_raw_data_v9(*ncid, raw_data, number_of_samples*)
Fortran Syntax	nf_write_individual_raw_data_v9(*ncid, raw_data, number_of_samples*)
Matlab Syntax	nc_write_individual_raw_data_v9(*ncid, raw_data, number_of_samples*)
Input Arguments	*ncid* NetCDF file ID number (integer) *raw_data* Time history data (1D floating point array) *number_of_samples* Number of time history data samples to write (integer)
Output Arguments	none
Remarks	The file must first be opened with a call to *open_raw_file_v9* before calling this subroutine.

Example Use:

nc_write_individual_raw_data_v9(ncid, raw_data, 1500000);	*C*
call nf_write_individual_raw_data_v9(ncid, raw_data, 1500000)	*Fortran*
nc_write_individual_raw_data_v9(ncid, raw_data, 1500000);	*Matlab*

write_ensemble_raw_data_v9

Purpose	This subroutine writes a full set of time history data to an ensemble raw data file.
C Syntax	nc_write_ensemble_raw_data_v9(*ncid, raw_data, number_of_channels, number_of_samples*)
Fortran Syntax	nf_write_ensemble_raw_data_v9(*ncid, raw_data, number_of_channels, number_of_samples*)
Matlab Syntax	nc_write_ensemble_raw_data_v9(*ncid, raw_data, number_of_channels, number_of_samples*)
Input Arguments	*ncid* NetCDF file ID number (integer) *raw_data* Time history data (2D floating point array) *number_of_channels* Number of channels stored in data file (integer) *number_of_samples* Number of samples per channel stored in data file (integer)
Output Arguments	none
Remarks	The file must first be opened with a call to *open_raw_file_v9* before calling this routine.

Example Use:

nc_write_ensemble_raw_data_v9(ncid, raw_data, 97, 1500000);	*C*
call nf_write_ensemble_raw_data_v9(ncid, raw_data, 97, 1500000)	*Fortran*
nc_write_ensemble_raw_data_v9(ncid, raw_data, 97, 1500000);	*Matlab*

write_pressure_data_v9

Purpose	This subroutine writes a set of pressure data to a static pressure data file.
C Syntax	nc_write_pressure_data_v9(*ncid, version_data, test_data, static_data*)
Fortran Syntax	nf_write_pressure_data_v9(*ncid, version_data, test_data, static_data*)
Matlab Syntax	nc_write_pressure_data_v9(*ncid, version_data, test_data, static_data*)
Input Arguments	*ncid* NetCDF file ID number (integer) *version_data* Data structure containing version data in header (see Appendices A, B, and C for format) *test_data* Data structure containing test data in header *static_data* Data structure containing static pressure tap data in header
Output Arguments	none
Remarks	The file must first be opened with a call to *open_pressure_file_v9* before calling this routine.

Example Use:

C

```
nc_write_pressure_data_v9(ncid, version_data, test_data, static_data);
```

Fortran

```
call nf_write_pressure_data_v9(ncid, version_data, test_data, static_data)
```

Matlab

```
nc_write_pressure_data_v9(ncid, version_data, test_data, static_data);
```

Purpose	This routine writes a set of cross spectral matrix data to a cross spectral matrix file.
C Syntax	nc_write_csm_data_v9(*ncid, csm_real_data, csm_imag_data, number_of_channels, number_of_bins*)
Fortran Syntax	nc_write_csm_data_v9(*ncid, csm_real_data, csm_imag_data, number_of_channels, number_of_bins*)
Matlab Syntax	nc_write_csm_data_v9(*ncid, csm_real_data, csm_imag_data, number_of_channels, number_of_bins*)
Input Arguments	*ncid* NetCDF file ID number (integer) *csm_real_data* 3D floating point array containing real component of cross spectral matrix, dimension is (num_channels, num_channels, num_bins) *csm_real_data* 3D floating point array containing imaginary component of cross spectral matrix, dimension is (num_channels, num_channels, num_bins) *number_of_channels* Number of channels stored in cross spectral matrix (integer) *number_of_bins* Number of frequency bins stored in cross spectral matrix (integer)
Output Arguments	none
Remarks	The file must first be opened with a call to *open_csm_file_v9* before calling this routine.

Example Use:

C

nc_write_csm_data_v9(ncid, csm_real_data, csm_imag_data, 97, 4096);

Fortran

call nf_write_csm_data_v9(ncid, csm_real_data, csm_imag_data, 97, 4096)

Matlab

nc_write_csm_data_v9(ncid, csm_real_data, csm_imag_data, 97, 4096);

Purpose
This routine returns fixed (non-facility and non-model) header entries from an open individual raw data file

C Syntax
nc_read_individual_raw_fixed_header_v9(*ncid,*
version_data,
test_data,
channel_data,
calibration_data,
das_data,
array_data,
number_of_sources,
number_of_bins,
number_of_inclinometers)

Fortran Syntax
nf_read_individual_raw_fixed_header_v9(*ncid,*
version_data,
test_data,
channel_data,
calibration_data,
das_data,
array_data,
number_of_sources,
number_of_bins,
number_of_inclinometers)

Matlab Syntax
[*version_data*, *test_data*, *channel_data*, *calibration_data*, *das_data*, *array_data*] = nc_read_individual_raw_fixed_header_v9(*ncid, number_of_sources, number_of_bins, number_of_inclinometers*)

Input Arguments
ncid
NetCDF file ID number (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

Output Arguments

version_data
Data structure containing version data in header
(see Appendices A, B, and C for format)

test_data
Data structure containing test data in header

channel_data
Data structure containing channel data in header

calibration_data
Data structure containing sensor calibration data in header

das_data
Data structure containing DAS data in header

array_data
Data structure containing array data in header

Remarks

The file must first be opened with a call to *open_raw_file_v9* before calling this routine.

Example Use:

C

```
nc_read_individual_raw_fixed_header_v9(ncid,version_data,test_data,channel_data,
calibration_data,das_data,array_data,8,4096,3);
```

Fortran

```
call nf_read_individual_raw_fixed_header_v9(ncid,version_data,test_data,channel_data,
calibration_data,das_data,array_data,8,4096,3)
```

Matlab

```
[version_data, test_data, channel_data, calibration_data, das_data, array_data] =
nc_read_individual_raw_fixed_header_v9(ncid, 8, 4096, 3);
```

Purpose

This routine returns fixed (non-facility and non-model) header entries from an open ensemble raw data file

C Syntax

nc_read_ensemble_raw_fixed_header_v9(*ncid,*
ensemble_version_data,
ensemble_test_data,
ensemble_channel_data,
ensemble_calibration_data,
ensemble_das_data,
ensemble_array_data,
number_of_channels,
number_of_sources,
number_of_cals,
number_of_bins,
number_of_inclinometers)

Fortran Syntax

nf_read_ensemble_raw_fixed_header_v9(*ncid,*
ensemble_version_data,
ensemble_test_data,
ensemble_channel_data,
ensemble_calibration_data,
ensemble_das_data,
ensemble_array_data,
number_of_channels,
number_of_sources,
number_of_cals,
number_of_bins,
number_of_inclinometers)

Matlab Syntax

[*ensemble_version_data*, *ensemble_test_data*, *ensemble_channel_data*, *ensemble_calibration_data*, *ensemble_das_data*, *ensemble_array_data*] = nf_read_ensemble_raw_fixed_header_v9(*ncid,*
number_of_channels,
number_of_sources,
number_of_cals,
number_of_bins,
number_of_inclinometers)

Input Arguments

ncid
NetCDF file ID number (integer)

number_of_channels
Number of data channels stored in file (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

Output Arguments

version_data
Data structure containing version data in header
(see Appendices A, B, and C for format)

test_data
Data structure containing test data in header

channel_data
Data structure containing channel data in header

calibration_data
Data structure containing sensor calibration data in header

das_data
Data structure containing DAS data in header

array_data
Data structure containing array data in header

Remarks

The file must first be opened with a call to *open_raw_file_v9* before calling this routine.

Example Use:

C

```
nc_read_ensemble_raw_fixed_header_v9(ncid, ensemble_version_data, ensemble_test_data,
ensemble_channel_data, ensemble_calibration_data, ensemble_das_data, ensemble_array_data,
97, 8, 10, 4096, 3);
```

Fortran

```
call nf_read_ensemble_raw_fixed_header_v9(ncid, ensemble_version_data, ensemble_test_data,
ensemble_channel_data, ensemble_calibration_data, ensemble_das_data, ensemble_array_data,
97, 8, 10, 4096, 3)
```

Matlab

```
[ensemble_version_data, ensemble_test_data, ensemble_channel_data,
ensemble_calibration_data, ensemble_das_data, ensemble_array_data] =
nc_read_ensemble_raw_fixed_header_v9(ncid, 97, 8, 10, 4096, 3);
```

Purpose	This routine returns fixed (non-facility and non-model) header entries from an open static pressure data file
C Syntax	nc_read_pressure_fixed_header_v9(*ncid, version_data, test_data, static_data)*
Fortran Syntax	nf_read_pressure_fixed_header_v9(*ncid, version_data, test_data, static_data)*
Matlab Syntax	[*version_data*, *test_data*, *static_data*] = nc_read_pressure_fixed_header_v9(*ncid*)
Input Arguments	*ncid* NetCDF file ID number (integer)
Output Arguments	*version_data* Data structure containing version data in header (see Appendices A, B, and C for format) *test_data* Data structure containing test data in header *static_data* Data structure containing static pressure data in header
Remarks	The file must first be opened with a call to *open_pressure_file_v9* before calling this routine.

Example Use:

C

```
nc_read_pressure_fixed_header_v9(ncid, version_data, test_data, static data);
```

Fortran

```
call nf_read_pressure_fixed_header_v9(ncid, version_data, test_data, static data)
```

Matlab

```
[version_data, test_data, static_data] = nc_read_pressure_fixed_header_v9(ncid);
```

Purpose — This routine returns fixed (non-facility and non-model) header entries from an open cross spectral matrix data file

C Syntax

```
nc_read_csm_fixed_header_v9(ncid,
                            csm_version_data,
                            csm_test_data,
                            csm_channel_data,
                            csm_calibration_data,
                            csm_das_data,
                            csm_array_data,
                            csm_data,
                            number_of_channels,
                            number_of_bins,
                            number_of_sources,
                            number_of_cals,
                            number_of_inclinometers,
                            number_of_blocks)
```

Fortran Syntax

```
nf_read_csm_fixed_header_v9(ncid,
                            csm_version_data,
                            csm_test_data,
                            csm_channel_data,
                            csm_calibration_data,
                            csm_das_data,
                            csm_array_data,
                            csm_data,
                            number_of_channels,
                            number_of_bins,
                            number_of_sources,
                            number_of_cals,
                            number_of_inclinometers,
                            number_of_blocks)
```

Matlab Syntax

```
[csm_version_data, csm_test_data, csm_channel_data,
 csm_calibration_data, csm_das_data, csm_array_data,
 csm_data] = nc_read_csm_fixed_header_v9(ncid,
                                         number_of_channels,
                                         number_of_bins,
                                         number_of_sources,
                                         number_of_cals,
                                         number_of_inclinometers,
                                         number_of_blocks)
```

Input Arguments

ncid
NetCDF file ID number (integer)

number_of_channels
Number of channels stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in file (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

number_of_blocks
Number of FFT blocks used to create cross spectral matrix (integer)

Output Arguments

csm_version_data
Data structure containing version data in header
(see Appendices A, B, and C for format)

csm_test_data
Data structure containing test data in header

csm_channel_data
Data structure containing channel data in header

csm_calibration_data
Data structure containing sensor calibration data in header

csm_das_data
Data structure containing DAS data in header

csm_array_data
Data structure containing array data in header

csm_data
Data structure containing csm data in header

Remarks

The file must first be opened with a call to *open_csm_file_v9* before calling this routine.

Example Use:

C

```
nc_read_csm_fixed_header_v9(ncid,csm_version_data,csm_test_data,csm_channel_data,
csm_calibration_data,csm_das_data,csm_array_data,csm_data,97,4096,8,10,3,100);
```

Fortran

```
call nf_read_csm_fixed_header_v9(ncid,csm_version_data,csm_test_data,csm_channel_data,
csm_calibration_data,csm_das_data,csm_array_data,csm_data,97,4096,8,10,3,100)
```

Matlab

```
[csm_version_data, csm_test_data, csm_channel_data, csm_calibration_data, csm_das_data,
csm_array_data,csm_data] = nc_read_csm_fixed_header_v9(ncid, 97, 4096, 8, 10, 3, 100);
```

Purpose	This routine returns facility-specific header entries from an individual raw / ensemble raw / static pressure / cross spectral matrix data file.
C Syntax	nc_read_facility_header_v9(*ncid, facility_data*)
Fortran Syntax	nf_read_facility_header_v9(*ncid, facility_data*)
Matlab Syntax	[*facility_data*] = nc_read_facility_header_v9(*ncid*)
Input Arguments	*ncid* NetCDF file ID number (integer)
Output Arguments	*facility_data* Data structure containing facility data in header (see Appendices A, B, and C for format)
Remarks	The file must first be opened with a call to *open_raw_file_v9, open_pressure_file_v9, or open_csm_file_v9* before calling this routine.

Example Use:	nc_read_facility_header_v9(ncid, facility_data);	*C*
	call nf_read_facility_header_v9(ncid, facility_data)	*Fortran*
	[facility_data] = nc_read_facility_header_v9(ncid);	*Matlab*

Purpose	This subroutine returns model-specific header entries from an individual raw / ensemble raw / static pressure / cross spectral matrix data file.
C Syntax	nc_read_model_header_v9(*ncid, model_data*)
Fortran Syntax	nf_read_model_header_v9(*ncid, model_data*)
Matlab Syntax	[*model_data*] = nc_read_facility_header_v9(*ncid*)
Input Arguments	*ncid* NetCDF file ID number (integer)
Output Arguments	*model_data* Fortran structure containing facility data in header (see Appendices A, B, and C for format)
Remarks	The file must first be opened with a call to *nf_open_raw_file_v9, nf_open_pressure_file_v9, or nf_open_csm_file_v9* before calling this routine.

Example Use:	nc_read_model_header_v9(ncid,model_data);	*C*
	call nf_read_model_header_v9(ncid, model_data)	*Fortran*
	[model_data] = nc_read_model_header_v9(ncid);	*Matlab*

Purpose	This subroutine reads a block of time history data from an individual raw data file.
C Syntax	nc_read_individual_raw_data_v9(*ncid, raw_data, number_of_samples*)
Fortran Syntax	nf_read_individual_raw_data_v9(*ncid, raw_data, number_of_samples*)
Matlab Syntax	[*raw_data*] = nc_read_individual_raw_data_v9(*ncid, number_of_samples*)
Input Arguments	*ncid* NetCDF file ID number (integer) *number_of_samples* Number of time history data samples to read (integer)
Output Arguments	*raw_data* Time history data (1D array of reals)
Remarks	The file must first be opened with a call to *nf_open_raw_file_v9* before calling this routine.

Example Use:

C

```
nc_read_individual_raw_data_v9(ncid, raw_data, 1500000);
```

Fortran

```
call nf_read_individual_raw_data_v9(ncid, raw_data, 1500000)
```

Matlab

```
[raw_data] = nc_read_individual_raw_data_v9(ncid, 1500000);
```

Purpose	This subroutine reads a set of time history data from an ensemble raw data file.
C Syntax	nc_read_ensemble_raw_data_v9(*ncid, raw_data, number_of_channels, number_of_samples*)
Fortran Syntax	nf_read_ensemble_raw_data_v9(*ncid, raw_data, number_of_channels, number_of_samples*)
Matlab Syntax	[*raw_data*] = nc_read_ensemble_raw_data_v9(*ncid, number_of_channels, number_of_samples*)
Input Arguments	*ncid* NetCDF file ID number (integer) *number_of_channels* Number of data channels stored in file (integer) *number_of_samples* Number of samples per data channel stored in file (integer)
Output Arguments	*raw_data* Time history data (2D array of reals)
Remarks	The file must first be opened with a call to *nf_open_raw_file_v9* before calling this subroutine.

Example Use:

C

```
nc_read_ensemble_raw_data_v9(ncid,raw_data, 97, 1500000);
```

Fortran

```
call nf_read_ensemble_raw_data_v9(ncid, raw_data, 97, 1500000)
```

Matlab

```
[raw_data] = nc_read_ensemble_raw_data_v9(ncid, 97, 1500000);
```

read_pressure_data_v9

Purpose	This subroutine reads a set of pressure data from a static pressure data file.
C Syntax	nc_read_pressure_data_v9(*ncid, static_data*)
Fortran Syntax	nf_read_pressure_data_v9(*ncid, static_data*)
Matlab Syntax	[*static_data*] = nc_read_pressure_data_v9(*ncid*)
Input Arguments	*ncid* NetCDF file ID number (integer)
Output Arguments	*static_data* Fortran structure containing static pressure tap data in header (see Appendices A, B, and C for format)
Remarks	The file must first be opened with a call to *nf_open_pressure_file_v9* before calling this subroutine.

Example Use:	nc_read_pressure_data_v9(ncid, static_data);	*C*
	call nf_read_pressure_data_v9(ncid, static_data)	*Fortran*
	[static_data] = nc_read_pressure_data_v9(ncid);	*Matlab*

Purpose	This subroutine reads a set of cross spectral matrix data from a cross spectral matrix file.
C Syntax	nc_read_csm_data_v9(*ncid, csm_real_data, csm_imag_data, number_of_channels, number_of_bins*)
Fortran Syntax	nf_read_csm_data_v9(*ncid, csm_real_data, csm_imag_data, number_of_channels, number_of_bins*)
Matlab Syntax	[*csm_real_data, csm_imag_data*] = nc_read_csm_data_v9(*ncid, number_of_channels, number_of_bins*)
Input Arguments	*ncid* NetCDF file ID number (integer) *number_of_channels* Number of channels stored in cross spectral matrix (integer) *number_of_bins* Number of frequency bins stored in cross spectral matrix (integer)
Output Arguments	*csm_real_data* 3D array containing real component of cross spectral matrix *csm_imag_data* 3D array containing imaginary component of cross spectral matrix
Remarks	The file must first be opened with a call to *nf_open_csm_file_v9* before calling this subroutine.

Example Use:

C

```
nc_read_csm_data_v9(ncid, csm_real_data, csm_imag_data, 97, 4096);
```

Fortran

```
call nf_read_csm_data_v9(ncid, csm_real_data, csm_imag_data, 97, 4096)
```

Matlab

```
[csm_real_data, csm_imag_data] = nc_read_csm_data_v9(ncid, 97, 4096);
```

9.0 HDF5 Function Reference

List of C / Fortran / Matlab HDF5 API Routines by Category

File Handling

- hdf5_open_raw_file_v9
- hdf5_open_pressure_file_v9
- hdf5_open_csm_file_v9
- hdf5_close_raw_file_v9
- hdf5_close_pressure_file_v9
- hdf5_close_csm_file_v9

File Definition / Initialization

- hdf5_define_individual_raw_file_v9
- hdr5_define_ensemble_raw_file_v9
- hdf5_define_pressure_file_v9
- hdf5_define_csm_file_v9

Data Writing

- hdf5_write_individual_raw_fixed_header_v9
- hdf5_write_ensemble_raw_fixed_header_v9
- hdf5_write_pressure_fixed_header_v9
- hdf5_write_csm_fixed_header_v9
- hdf5_write_facility_header_v9
- hdf5_write_model_header_v9
- hdf5_write_individual_raw_data_v9
- hdf5_write_ensemble_raw_data_v9
- hdf5_write_pressure_data_v9
- hdf5_write_csm_data_v9

Data Reading

- hdf5_read_individual_raw_fixed_header_v9
- hdf5_read_ensemble_raw_fixed_header_v9
- hdf5_read_pressure_fixed_header_v9
- hdf5_read_csm_fixed_header_v9
- hdf5_read_facility_header_v9
- hdf5_read_model_header_v9
- hdf5_read_individual_raw_data_v9
- hdf5_read_ensemble_raw_data_v9
- hdf5_read_pressure_data_v9
- hdf5_read_csm_data_v9

hdf5_open_raw_file_v9

Purpose	This routine opens an existing HDF5 individual or ensemble raw data file for access.
C Syntax	*file_id* = hdf5_open_raw_file_v9(*name*, *mode*)
Fortran Syntax	hdf5_open_raw_file_v9(*name, file_id, mode*)
Matlab Syntax	*file_id* = hdf5_open_raw_file_v9(*name*, *mode*)
Input Arguments	*name* HDF5 file name to open (string) *mode* File access mode (string) 'r', 'R' = read only 'w', 'W' = read / write
Output Arguments	*file_id* HDF5 file ID number (integer)
Remarks	An error will be returned if the *mode* is 'r' and the file does not exist.

Example Use:

```
file_id = hdf5_open_raw_file_v9("T0001R0001P0001C0001.h5",'w');          C
call hdf5_open_raw_file_v9('T0001R0001P0001C0001.h5',file_id,'w')        Fortran
file_id = hdf5_open_raw_file_v9('T0001R0001P0001C0001.h5','w');          Matlab
```

hdf5_open_pressure_file_v9

Purpose	This routine opens an existing HDF5 static pressure data file for access.
C Syntax	*file_id* = hdf5_open_pressure_file_v9(*name*,*mode*)
Fortran Syntax	hdf5_open_pressure_file_v9(*name, file_id, mode*)
Matlab Syntax	*file_id* = hdf5_open_pressure_file_v9(*name, mode*)
Input Arguments	*name* HDF5 file name to open (string) *mode* File access mode (string) 'r','R' = read only 'w','W' = read / write
Output Arguments	*file_id* HDF5 file ID number (integer)
Remarks	An error will be returned if the *mode* is 'r' and the file does not exist.

Example Use:

file_id = hdf5_open_pressure_file_v9("T0001R0001P0001_pressure.h5",'w'); *C*

call hdf5_open_pressure_file_v9('T0001R0001P0001_pressure.h5',file_id,'w') *Fortran*

file_id = hdf5_open_pressure_file_v9('T0001R0001P0001_pressure.h5','w'); *Matlab*

hdf5_open_csm_file_v9

Purpose	This routine opens an existing HDF5 cross spectral matrix data file for access.
C Syntax	*file_id* = hdf5_open_csm_file_v9(*name*,*mode*)
Fortran Syntax	hdf5_open_csm_file_v9(*name, file_id, mode*)
Matlab Syntax	*file_id* = hdf5_open_csm_file_v9(*name, mode*)
Input Arguments	*name* HDF5 file name to open (string) *mode* File access mode (string) 'r','R' = read only 'w','W' = read / write
Output Arguments	*file_id* HDF5 file ID number (integer)
Remarks	An error will be returned if the *mode* is 'r' and the file does not exist.

Example Use:

```
file_id = hdf5_open_csm_file_v9("T0001R0001P0001_CSM.nc",'w');          C
call hdf5_open_csm_file_v9('T0001R0001P0001_CSM.nc',file_id,'w')        Fortran
file_id = hdf5_open_csm_file_v9('T0001R0001P0001_CSM.nc','w');          Matlab
```

hdf5_close_raw_file_v9

Purpose	This routine closes an open HDF5 individual or ensemble raw data file.
C Syntax	hdf5_close_raw_file_v9(*file_id*)
Fortran Syntax	hdf5_close_raw_file_v9(*file_id*)
Matlab Syntax	hdf5_close_raw_file_v9(*file_id*)
Input Arguments	*file_id* HDF5 file ID number (integer)
Output Arguments	none
Remarks	An error will be returned if the file is not currently open.

Example Use:

```
hdf5_close_raw_file_v9(file_id);                    C
call hdf5_close_raw_file_v9(file_id)                Fortran
hdf5_close_raw_file_v9(file_id);                    Matlab
```

hdf5_close_pressure_file_v9

Purpose	This routine closes an open HDF5 static pressure data file.
C Syntax	hdf5_close_pressure_file_v9(*file_id*)
Fortran Syntax	hdf5_close_pressure_file_v9(*file_id*)
Matlab Syntax	hdf5_close_pressure_file_v9(*file_id*)
Input Arguments	*file_id* HDF5 file ID number (integer)
Output Arguments	none
Remarks	An error will be returned if the file is not currently open.

Example Use:

```
hdf5_close_pressure_file_v9(file_id);                    C
call hdf5_close_pressure_file_v9(file_id)                Fortran
hdf5_close_pressure_file_v9(file_id);                    Matlab
```

hdf5_close_csm_file_v9

Purpose	This routine closes an open HDF5 cross spectral matrix data file.
C Syntax	hdf5_close_csm_file_v9(*file_id*)
Fortran Syntax	hdf5_close_csm_file_v9(*file_id*)
Matlab Syntax	hdf5_close_csm_file_v9(*file_id*)
Input Arguments	*file_id* HDF5 file ID number (integer)
Output Arguments	none
Remarks	An error will be returned if the file is not currently open.

Example Use:

```
hdf5_close_csm_file_v9(file_id);                    C
call hdf5_close_csm_file_v9(file_id)                Fortran
hdf5_close_csm_file_v9(file_id);                    Matlab
```

hdf5_define_individual_raw_file_v9

Purpose	This routine opens and defines a new HDF5 individual raw data file.
C Syntax	hdf5_define_individual_raw_file_v9(*name, number_of_sources, number_of_bins, number_of_inclinometers, number_of_samples*)
Fortran Syntax	hdf5_define_individual_raw_file_v9(*name, number_of_sources, number_of_bins, number_of_inclinometers, number_of_samples*)
Matlab Syntax	hdf5_define_individual_raw_file_v9(*name, number_of_sources, number_of_bins, number_of_inclinometers, number_of_samples*)
Input Arguments	*name* Name of HDF5 individual raw data file to create (string) *number_of_sources* Number of individual calibration sources used with array (integer) *number_of_bins* Number of cross spectral matrix frequency bins (integer) *number_of_inclinometers* Number of inclinometers incorporated into array (integer) *number_of_samples* Number of time series samples acquired in file (integer)
Output Arguments	none
Remarks	An error will be returned if the file already exists.

Example Use:

C

hdf5_define_individual_raw_file_v9(“T0001R0001P0001C0001.h5”,8,4096,3,1500000);

Fortran

call hdf5_define_individual_raw_file_v9(‘T0001R0001P0001C0001.h5’,8,4096,3,1500000)

Matlab

hdf5_define_individual_raw_file_v9(‘T0001R0001P0001C0001.h5’,8,4096,3,1500000);

Purpose — This routine opens and defines a new HDF5 ensemble raw data file.

C Syntax — hdf5_define_ensemble_raw_file_v9(*name, number_of_channels, number_of_bins, number_of_sources, number_of_cals, number_of_inclinometers, number_of_samples*)

Fortran Syntax — hdf5_define_ensemble_raw_file_v9(*name, number_of_channels, number_of_bins, number_of_sources, number_of_cals, number_of_inclinometers, number_of_samples*)

Matlab Syntax — hdf5_define_ensemble_raw_file_v9(*name, number_of_channels, number_of_bins, number_of_sources, number_of_cals, number_of_inclinometers, number_of_samples*)

Input Arguments

name
Name of HDF5 ensemble raw data file to create (string)

number_of_channels
Number of channels stored in data file (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

number_of_samples
Number of time series samples acquired in file (integer)

Output Arguments none

Remarks An error will be returned if the file already exists.

Example Use:

C

hdf5_define_ensemble_raw_file_v9(“T0001R0001P0001.h5”,97,8,10,4096,3,1500000);

Fortran

call hdf5_define_ensemble_raw_file_v9(‘T0001R0001P0001.h5’,97,8,10,4096,3,1500000)

Matlab

hdf5_define_ensemble_raw_file_v9(‘T0001R0001P0001.h5’,97,8,10,4096,3,1500000);

hdf5_define_pressure_file_v9

Purpose	This routine opens and defines a new HDF5 static pressure tap raw data file.
C Syntax	hdf5_define_pressure file_v9(*name*)
Fortran Syntax	hdf5_define_pressure_file_v9(*name*)
Matlab Syntax	hdf5_define_pressure file_v9(*name*)
Input Arguments	*name* Name of HDF5 static pressure tap data file to create (string)
Output Arguments	none
Remarks	An error will be returned if the file already exists.

Example Use:

hdf5_define_pressure_file_v9("T0001R0001P0001_pressure.h5");	*C*
call hdf5_define_pressure_file_v9('T0001R0001P0001_pressure.h5')	*Fortran*
hdf5_define_pressure_file_v9('T0001R0001P0001_pressure.h5');	*Matlab*

hdf5_define_csm_file_v9

Purpose

This routine opens and defines a new HDF5 cross spectral matrix data file.

C Syntax

hdf5_define_csm_file_v9(*name, number_of_channels,*
number_of_bins,
number_of_sources,
number_of_cals,
number_of_inclinometers,
number_of_blocks)

Fortran Syntax

hdf5_define_csm_file_v9(*name, number_of_channels,*
number_of_bins,
number_of_sources,
number_of_cals,
number_of_inclinometers,
number_of_blocks)

Matlab Syntax

hdf5_define_csm_file_v9(*name, number_of_channels,*
number_of_bins,
number_of_sources,
number_of_cals,
number_of_inclinometers,
number_of_blocks)

Input Arguments

name
Name of HDF5 cross spectral matrix data file to create (string)

number_of_channels
Number of channels stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in data file (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

number_of_blocks
Number of FFT blocks used to create cross spectral matrix (integer)

Output Arguments none

Remarks An error will be returned if the file already exists.

Example Use:

C

hdf5_define_csm_file_v9(“T0001R0001P0001_CSM.nc”,97,4096,8,10,3,300);

Fortran

call hdf5_define_csm_file_v9(‘T0001R0001P0001_CSM.nc’,97,4096,8,10,3,300)

Matlab

hdf5_define_csm_file_v9(‘T0001R0001P0001_CSM.nc’,97,4096,8,10,3,300);

hdf5_write_individual_raw_fixed_header_v9

Purpose — This routine populates an individual raw data file header with fixed (non-facility and non-model) entries.

C Syntax

```
hdf5_write_individual_raw_fixed_header_v9(file_id,
                                          version_data,
                                          test_data,
                                          channel_data,
                                          calibration_data,
                                          das_data,
                                          array_data,
                                          number_of_sources,
                                          number_of_bins,
                                          number_of_inclinometers)
```

Fortran Syntax

```
hdf5_write_individual_raw_fixed_header_v9(file_id,
                                          version_data,
                                          test_data,
                                          channel_data,
                                          calibration_data,
                                          das_data,
                                          array_data,
                                          number_of_sources,
                                          number_of_bins,
                                          number_of_inclinometers)
```

Matlab Syntax

```
hdf5_write_individual_raw_fixed_header_v9(file_id,
                                          version_data,
                                          test_data,
                                          channel_data,
                                          calibration_data,
                                          das_data,
                                          array_data)
```

Input Arguments

file_id
HDF5 file ID number (integer)

version_data
Data structure containing version data in header
(see Appendices A, B, and C for format)

test_data
Data structure containing test data in header

channel_data
Data structure containing channel data in header

calibration_data
Data structure containing sensor calibration data in header

das_data
Data structure containing DAS data in header

array_data
Data structure containing array data in header

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

Output Arguments none

Remarks The file must first be opened with a call to *hdf5_open_raw_file_v9* before calling this routine.

Example Use:

C

```
hdf5_write_individual_raw_fixed_header_v9(file_id,version_data,test_data,channel_data,
calibration_data,das_data,array_data,8,4096,3);
```

Fortran

```
call hdf5_write_individual_raw_fixed_header_v9(file_id,version_data,test_data,channel_data,
calibration_data,das_data,array_data,8,4096,3)
```

Matlab

```
hdf5_write_individual_raw_fixed_header_v9(file_id,version_data,test_data,channel_data,
calibration_data,das_data,array_data);
```

Purpose

This routine populates an ensemble raw data file header with fixed (non-facility and non-model) entries.

C Syntax

hdf5_write_ensemble_raw_fixed_header_v9(*file_id,*
ensemble_version_data,
ensemble_test_data,
ensemble_channel_data,
ensemble_calibration_data,
ensemble_das_data,
ensemble_array_data,
number_of_channels,
number_of_bins,
number_of_sources,
number_of_cals,
number_of_inclinometers)

Fortran Syntax

hdf5_write_ensemble_raw_fixed_header_v9(*file_id,*
ensemble_version_data,
ensemble_test_data,
ensemble_channel_data,
ensemble_calibration_data,
ensemble_das_data,
ensemble_array_data,
number_of_channels,
number_of_bins,
number_of_sources,
number_of_cals,
number_of_inclinometers)

Matlab Syntax

hdf5_write_ensemble_raw_fixed_header_v9(*file_id,*
ensemble_version_data,
ensemble_test_data,
ensemble_channel_data,
ensemble_calibration_data,
ensemble_das_data,
ensemble_array_data)

Input Arguments

file_id
HDF5 file ID number (integer)

ensemble_version_data
Data structure containing ensemble version data in header
(see Appendices A, B, and C for format)

ensemble_test_data
Data structure containing ensemble test data in header

ensemble_channel_data
Data structure containing ensemble channel data in header

ensemble_calibration_data
Data structure containing ensemble sensor calibration data in header

ensemble_das_data
Data structure containing ensemble DAS data in header

ensemble_array_data
Data structure containing ensemble array data in header

number_of_channels
Number of channels stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in file (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

Output Arguments — none

Remarks — The file must first be opened with a call to *hdf5_open_raw_file_v9* before calling this subroutine.

Example Use:

C

```
hdf5_write_ensemble_raw_fixed_header_v9(file_id, ensemble_version_data,
ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data,
ensemble_array_data, 97, 4096, 8, 10, 3);
```

Fortran

```
call hdf5_write_ensemble_raw_fixed_header_v9(file_id, ensemble_version_data,
ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data,
ensemble_array_data, 97, 4096, 8, 10, 3)
```

Matlab

```
hdf5_write_ensemble_raw_fixed_header_v9(file_id, ensemble_version_data,
ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data,
ensemble_array_data);
```

hdf5_write_pressure_fixed_header_v9

Purpose	This routine populates a static pressure data file header with fixed (non-facility and non-model) entries.
C Syntax	hdf5_write_pressure_fixed_header_v9(*file_id*, *version_data, test_data, static_data*)
Fortran Syntax	hdf5_write_pressure_fixed_header_v9(*file_id*, *version_data, test_data, static_data*)
Matlab Syntax	hdf5_write_pressure_fixed_header_v9(*file_id*, *version_data*, *test_data*, *static_data)*
Input Arguments	*file_id* HDF5 file ID number (integer) *version_data* Data structure containing version data in header (see Appendices A, B, and C for format) *test_data* Data structure containing test data in header *static_data* Data structure containing static pressure header data
Output Arguments	none
Remarks	The file must first be opened with a call to *hdf5_open_pressure_file_v9* before calling this routine.

Example Use:

C

```
hdf5_write_pressure_fixed_header_v9(file_id, version_data, test_data, static_data);
```

Fortran

```
call hdf5_write_pressure_fixed_header_v9(file_id, version_data, test_data, static_data)
```

Matlab

```
hdf5_write_pressure_fixed_header_v9(file_id, version_data, test_data, static_data);
```

Purpose — This routine populates a cross spectral matrix data file header with fixed (non-facility and non-model) entries.

C Syntax

```
hdf5_write_csm_fixed_header_v9(file_id,
                               csm_version_data,
                               csm_test_data,
                               csm_channel_data,
                               csm_calibration_data,
                               csm_das_data,
                               csm_array_data,
                               csm_data,
                               number_of_channels,
                               number_of_bins,
                               number_of_sources,
                               number_of_cals,
                               number_of_inclinometers,
                               number_of_blocks)
```

Fortran Syntax

```
hdf5_write_csm_fixed_header_v9(file_id,
                               csm_version_data,
                               csm_test_data,
                               csm_channel_data,
                               csm_calibration_data,
                               csm_das_data,
                               csm_array_data,
                               csm_data,
                               number_of_channels,
                               number_of_bins,
                               number_of_sources,
                               number_of_cals,
                               number_of_inclinometers,
                               number_of_blocks)
```

Matlab Syntax

```
hdf5_write_csm_fixed_header_v9(file_id,
                               csm_version_data,
                               csm_test_data,
                               csm_channel_data,
                               csm_calibration_data,
                               csm_das_data,
                               csm_array_data,
                               csm_data)
```

Input Arguments

file_id
HDF5 file ID number (integer)

csm_version_data
Data structure containing version data in header
(see Appendices A, B, and C for format)

csm_test_data
Data structure containing test data in header

csm_channel_data
Data structure containing channel data in header

csm_calibration_data
Data structure containing sensor calibration data in header

csm_das_data
Data structure containing DAS data in header

csm_array_data
Data structure containing array data in header

csm_data
Data structure containing csm data in header

number_of_channels
Number of channels stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in file (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

number_of_blocks
Number of FFT blocks used to create cross spectral matrix (integer)

Output Arguments	none
Remarks	The file must first be opened with a call to *hdf5_open_csm_file_v9* before calling this routine.

Example Use:

C

```
hdf5_write_csm_fixed_header_v9(file_id, csm_version_data, csm_test_data, csm_channel_data,
csm_calibration_data, csm_das_data, csm_array_data, csm_data,
97, 4096, 8, 10, 3, 100);
```

Fortran

```
call hdf5_write_csm_fixed_header_v9(file_id, csm_version_data, csm_test_data,
csm_channel_data, csm_calibration_data, csm_das_data, csm_array_data, csm_data,
97, 4096, 8, 10, 3, 100)
```

Matlab

```
hdf5_write_csm_fixed_header_v9(file_id, csm_version_data, csm_test_data, csm_channel_data,
csm_calibration_data, csm_das_data, csm_array_data, csm_data);
```

hdf5_write_facility_header_v9

Purpose	This routine populates an individual raw / ensemble raw / static pressure / cross spectral matrix data file header with facility-specific entries.
C Syntax	hdf5_write_facility_header_v9(*file_id, facility_data*)
Fortran Syntax	hdf5_write_facility_header_v9(*file_id, facility_data*)
Matlab Syntax	hdf5_write_facility_header_v9(*file_id, facility_data*)
Input Arguments	*file_id* HDF5 file ID number (integer) *facility_data* Data structure containing facility data in header (see Appendices A, B, and C for format)
Output Arguments	none
Remarks	The file must first be opened with a call to *hdf5_open_raw_file_v9, hdf5_open_pressure_file_v9, or hdf5_open_csm_file_v9* before calling this routine.

Example Use:	hdf5_write_facility_header_v9(file_id, facility_data);	*C*
	call hdf5_write_facility_header_v9(file_id, facility_data)	*Fortran*
	hdf5_write_facility_header_v9(file_id, facility_data);	*Matlab*

hdf5_write_model_header_v9

Purpose	This routine populates an individual raw / ensemble raw / static pressure / cross spectral matrix data file header with model-specific entries.
C Syntax	hdf5_write_model_header_v9(*file_id, model_data*)
Fortran Syntax	hdf5_write_model_header_v9(*file_id, model_data*)
Matlab Syntax	hdf5_write_model_header_v9(*file_id, model_data*)
Input Arguments	*file_id* HDF5 file ID number (integer) *model_data* Data structure containing model data in header (see Appendices A, B, and C for format)
Output Arguments	none
Remarks	The file must first be opened with a call to *hdf5_open_raw_file_v9, hdf5_open_pressure_file_v9, or hdf5_open_csm_file_v9* before calling this subroutine.

Example Use:	hdf5_write_model_header_v9(file_id, model_data);	*C*
	call hdf5_write_model_header_v9(file_id, model_data)	*Fortran*
	hdf5_write_model_header_v9(file_id, model_data);	*Matlab*

hdf5_write_individual_raw_data_v9

Purpose	This routine writes a block of time history data to an individual raw data file.
C Syntax	hdf5_write_individual_raw_data_v9(*file_id, raw_data*)
Fortran Syntax	hdf5_write_individual_raw_data_v9(*file_id, number_of_samples, raw_data*)
Matlab Syntax	hdf5_write_individual_raw_data_v9(*file_id, raw_data*)
Input Arguments	*file_id* HDF5 file ID number (integer) *number_of_samples* Number of time history data samples to write (integer) *raw_data* Time history data (1D array of reals)
Output Arguments	none
Remarks	The file must first be opened with a call to *hdf5_open_raw_file_v9* before calling this routine.

Example Use:

C

```
hdf5_write_individual_raw_data_v9(file_id, raw_data);
```

Fortran

```
call hdf5_write_individual_raw_data_v9(file_id, 1500000, raw_data)
```

Matlab

```
hdf5_write_individual_raw_data_v9(file_id, raw_data);
```

Purpose	This routine writes a complete set of time history data to an ensemble raw data file.
C Syntax	hdf5_write_ensemble_raw_data_v9(*file_id, ensemble_raw_data*)
Fortran Syntax	hdf5_write_ensemble_raw_data_v9(*file_id, number_of_channels, number_of_samples, ensemble_raw_data*)
Matlab Syntax	hdf5_write_ensemble_raw_data_v9(*file_id, ensemble_raw_data*)
Input Arguments	*file_id* HDF5 file ID number (integer) *number_of_channels* Number of data channels to write (integer) *number_of_samples* Number of time history data samples to write per channel (integer) *ensemble_raw_data* Ensemble time history data (2D array of reals)
Output Arguments	none
Remarks	The file must first be opened with a call to *hdf5_open_raw_file_v9* before calling this routine.

Example Use:

C

```
hdf5_write_ensemble_raw_data_v9(file_id, ensemble_raw_data);
```

Fortran

```
call hdf5_write_ensemble_raw_data_v9(file_id, 97, 1500000, ensemble_raw_data)
```

Matlab

```
hdf5_write_ensemble_raw_data_v9(file_id, ensemble_raw_data);
```

hdf5_write_pressure_data_v9

Purpose	This routine writes static pressure tap data to a pressure data file.
C Syntax	hdf5_write_pressure_data_v9(*file_id, static_data*)
Fortran Syntax	hdf5_write_pressure_data_v9(*file_id, static_data*)
Matlab Syntax	hdf5_write_pressure_data_v9(*file_id, static_data*)
Input Arguments	*file_id* HDF5 file ID number (integer) *static_data* Data structure containing static pressure tap data (see Appendices A, B, and C for format)
Output Arguments	none
Remarks	The file must first be opened with a call to *hdf5_open_pressure_file_v9* before calling this routine.

Example Use:

hdf5_write_pressure_data_v9(file_id, static_data);	*C*
call hdf5_write_pressure_data_v9(file_id, static_data)	*Fortran*
hdf5_write_pressure_data_v9(file_id, static_data);	*Matlab*

hdf5_write_csm_data_v9

Purpose	This routine writes cross spectral matrix data to a CSM data file.
C Syntax	hdf5_write_csm_data_v9(*file_id, csm_real_data,csm_imag_data*)
Fortran Syntax	hdf5_write_csm_data_v9(*file_id, number_of_channels, number_of_bins, csm_real_data, csm_imag_data*)
Matlab Syntax	hdf5_write_csm_data_v9(*file_id, csm_real_data,csm_imag_data*)
Input Arguments	*file_id* HDF5 file ID number (integer) *number_of_channels* Number of channels contained in cross spectral matrix (integer) *number_of_bins* Number of frequency bins contained in cross spectral matrix (integer) *csm_real_data* 3D floating point array containing real component of cross spectral matrix, dimension of array is (num_channels, num_channels, num_bins) *csm_imag_data* 3D floating point array containing imaginary component of cross spectral matrix, dimension of array is (num_channels, num_channels, num_bins)
Output Arguments	none
Remarks	The file must first be opened with a call to *hdf5_open_csm_file_v9* before calling this routine.

Example Use:

```
hdf5_write_csm_data_v9(file_id, csm_real_data,csm_imag_data);                C
call hdf5_write_csm_data_v9(file_id, 97, 4096, csm_real_data, csm_imag_data) Fortran
hdf5_write_csm_data_v9(file_id, csm_real_data, csm_imag_data);               Matlab
```

hdf5_read_individual_raw_fixed_header_v9

Purpose

This routine retrieves a set of fixed (non-facility and non-model) entries from an open individual raw data file.

C Syntax

hdf5_read_individual_raw_fixed_header_v9(*file_id,*
version_data,
test_data,
channel_data,
calibration_data,
das_data,
array_data,
number_of_sources,
number_of_bins,
number_of_inclinometers)

Fortran Syntax

hdf5_read_individual_raw_fixed_header_v9(*file_id,*
version_data,
test_data,
channel_data,
calibration_data,
das_data,
array_data,
number_of_sources,
number_of_bins,
number_of_inclinometers)

Matlab Syntax

[*version_data*,
test_data,
channel_data,
calibration_data,
das_data,
array_data] = hdf5_read_individual_raw_fixed_header_v9(*file_id)*

Input Arguments

file_id
HDF5 file ID number (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

Output Arguments

version_data
Data structure containing version data in header
(see Appendices A, B, and C for format)

test_data
Data structure containing test data in header

channel_data
Data structure containing channel data in header

calibration_data
Data structure containing sensor calibration data in header

das_data
Data structure containing DAS data in header

array_data
Data structure containing array data in header

Remarks

The file must first be opened with a call to *hdf5_open_raw_file_v9* before calling this routine.

Example Use:

C

```
hdf5_read_individual_raw_fixed_header_v9(file_id, version_data, test_data, channel_data,
calibration_data, das_data, array_data, 8, 4096, 3);
```

Fortran

```
call hdf5_read_individual_raw_fixed_header_v9(file_id, version_data, test_data, channel_data,
calibration_data, das_data, array_data, 8, 4096, 3)
```

Matlab

```
[version_data, test_data, channel_data, calibration_data, das_data, array_data] =
hdf5_read_individual_raw_fixed_header_v9(file_id);
```

Purpose

This routine retrieves a set of fixed (non-facility and non-model) entries from an open ensemble raw data file.

C Syntax

hdf5_read_ensemble_raw_fixed_header_v9(*file_id,*
ensemble_version_data,
ensemble_test_data,
ensemble_channel_data,
ensemble_calibration_data,
ensemble_das_data,
ensemble_array_data,
number_of_channels,
number_of_bins,
number_of_sources,
number_of_cals,
number_of_inclinometers)

Fortran Syntax

hdf5_read_ensemble_raw_fixed_header_v9(*file_id,*
ensemble_version_data,
ensemble_test_data,
ensemble_channel_data,
ensemble_calibration_data,
ensemble_das_data,
ensemble_array_data,
number_of_channels,
number_of_bins,
number_of_sources,
number_of_cals,
number_of_inclinometers)

Matlab Syntax

[*ensemble_version_data*,
ensemble_test_data,
ensemble_channel_data,
ensemble_calibration_data,
ensemble_das_data,
ensemble_array_data] =
hdf5_read_ensemble_raw_fixed_header_v9(*file_id)*

Input Arguments

file_id
HDF5 file ID number (integer)

number_of_channels
Number of channels stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in file (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

Output Arguments

ensemble_version_data
Data structure containing ensemble version data in header
(see Appendices A, B, and C for format)

ensemble_test_data
Data structure containing ensemble test data in header

ensemble_channel_data
Data structure containing ensemble channel data in header

ensemble_calibration_data
Data structure containing ensemble sensor calibration data in header

ensemble_das_data
Data structure containing ensemble DAS data in header

ensemble_array_data
Data structure containing ensemble array data in header

Remarks

The file must first be opened with a call to *hdf5_open_raw_file_v9* before calling this subroutine.

Example Use:

C

```
hdf5_read_ensemble_raw_fixed_header_v9(file_id, ensemble_version_data, ensemble_test_data,
ensemble_channel_data, ensemble_calibration_data, ensemble_das_data, ensemble_array_data,
97, 4096, 8, 10, 3);
```

Fortran

```
call hdf5_read_ensemble_raw_fixed_header_v9(file_id, ensemble_version_data,
ensemble_test_data, ensemble_channel_data, ensemble_calibration_data, ensemble_das_data,
ensemble_array_data, 97, 4096, 8, 10, 3)
```

Matlab

```
[ensemble_version_data, ensemble_test_data, ensemble_channel_data,
 ensemble_calibration_data, ensemble_das_data, ensemble_array_data] =
 hdf5_read_ensemble_raw_fixed_header_v9(file_id);
```

hdf5_read_pressure_fixed_header_v9

Purpose	This routine retrieves a set of fixed (non-facility and non-model) entries from an open pressure data file.
C Syntax	hdf5_read_pressure_fixed_header_v9(*file_id*, *version_data, test_data, static_data*)
Fortran Syntax	hdf5_read_pressure_fixed_header_v9(*file_id*, *version_data, test_data, static_data*)
Matlab Syntax	[*version_data*, *test_data*, *static_data*] = hdf5_write_pressure_fixed_header_v9(*file_id)*
Input Arguments	*file_id* HDF5 file ID number (integer) *version_data* Data structure containing version data in header (see Appendices A, B, and C for format) *test_data* Data structure containing test data in header *static_data* Data structure containing static pressure header data
Output Arguments	none
Remarks	The file must first be opened with a call to *hdf5_open_pressure_file_v9* before calling this routine.

Example Use:

C

```
hdf5_read_pressure_fixed_header_v9(file_id, version_data, test_data, static_data);
```

Fortran

```
call hdf5_read_pressure_fixed_header_v9(file_id, version_data, test_data, static_data)
```

Matlab

```
[version_data, test_data, static_data] = hdf5_read_pressure_fixed_header_v9(file_id);
```

Purpose

This routine retrieves a set of fixed (non-facility and non-model) entries from an open cross spectral matrix data file.

C Syntax

```
hdf5_read_csm_fixed_header_v9(file_id,
                              csm_version_data,
                              csm_test_data,
                              csm_channel_data,
                              csm_calibration_data,
                              csm_das_data,
                              csm_array_data,
                              csm_data,
                              number_of_channels,
                              number_of_bins,
                              number_of_sources,
                              number_of_cals,
                              number_of_inclinometers,
                              number_of_blocks)
```

Fortran Syntax

```
hdf5_read_csm_fixed_header_v9(file_id,
                              csm_version_data,
                              csm_test_data,
                              csm_channel_data,
                              csm_calibration_data,
                              csm_das_data,
                              csm_array_data,
                              csm_data,
                              number_of_channels,
                              number_of_bins,
                              number_of_sources,
                              number_of_cals,
                              number_of_inclinometers,
                              number_of_blocks)
```

Matlab Syntax

```
[csm_version_data,
 csm_test_data,
 csm_channel_data,
 csm_calibration_data,
 csm_das_data,
 csm_array_data,
 csm_data] = hdf5_read_csm_fixed_header_v9(file_id)
```

Input Arguments

file_id
HDF5 file ID number (integer)

number_of_channels
Number of channels stored in data file (integer)

number_of_bins
Number of cross spectral matrix frequency bins (integer)

number_of_sources
Number of individual calibration sources used with array (integer)

number_of_cals
Number of individual sets of calibration data stored in file (integer)

number_of_inclinometers
Number of inclinometers incorporated into array (integer)

number_of_blocks
Number of FFT blocks used to create cross spectral matrix (integer)

Output Arguments

csm_version_data
Data structure containing version data in header
(see Appendices A, B, and C for format)

csm_test_data
Data structure containing test data in header

csm_channel_data
Data structure containing channel data in header

csm_calibration_data
Data structure containing sensor calibration data in header

csm_das_data
Data structure containing DAS data in header

csm_array_data
Data structure containing array data in header

csm_data
Data structure containing csm data in header

Remarks

The file must first be opened with a call to *hdf5_open_csm_file_v9* before calling this routine.

Example Use:

C

```
hdf5_read_csm_fixed_header_v9(file_id, csm_version_data, csm_test_data, csm_channel_data,
csm_calibration_data, csm_das_data, csm_array_data, csm_data,
97, 4096, 8, 10, 3, 100);
```

Fortran

```
call hdf5_read_csm_fixed_header_v9(file_id, csm_version_data, csm_test_data,
csm_channel_data, csm_calibration_data, csm_das_data, csm_array_data, csm_data,
97, 4096, 8, 10, 3, 100)
```

Matlab

```
[csm_version_data, csm_test_data, csm_channel_data, csm_calibration_data, csm_das_data,
 csm_array_data, csm_data] = hdf5_read_csm_fixed_header_v9(file_id);
```

hdf5_read_facility_header_v9

Purpose	This routine returns facility-specific header entries from an individual raw / ensemble raw / static pressure / cross spectral matrix data file.
C Syntax	hdf5_read_facility_header_v9(*file_id, facility_data*)
Fortran Syntax	hdf5_read_facility_header_v9(*file_id, facility_data*)
Matlab Syntax	[*facility_data*] = hdf5_read_facility_header_v9(*file_id*)
Input Arguments	*file_id* HDF5 file ID number (integer)
Output Arguments	*facility_data* Data structure containing facility data in header (see Appendices A, B, and C for format)
Remarks	The file must first be opened with a call to *hdf5_open_raw_file_v9, hdf5_open_pressure_file_v9, or hdf5_open_csm_file_v9* before calling this routine.

Example Use:

```
nf_read_facility_header_v9(file_id, facility_data);                  C
call nf_read_facility_header_v9(file_id, facility_data)              Fortran
[facility_data] = hdf5_read_facility_header_v9(file_id);             Matlab
```

hdf5_read_model_header_v9

Purpose	This routine returns model-specific header entries from an individual raw / ensemble raw / static pressure / cross spectral matrix data file.
C Syntax	hdf5_read_model_header_v9(*file_id, model_data*)
Fortran Syntax	hdf5_read_model_header_v9(*file_id, model_data*)
Matlab Syntax	[*model_data*] = hdf5_read_model_header_v9(*file_id*)
Input Arguments	*file_id* HDF5 file ID number (integer)
Output Arguments	*model_data* Data structure containing facility data in header (see Appendices A, B, and C for format)
Remarks	The file must first be opened with a call to *hdf5_open_raw_file_v9, hdf5_open_pressure_file_v9, or hdf5_open_csm_file_v9* before calling this routine.

Example Use:	hdf5_read_model_header_v9(file_id, model_data);	*C*
	call hdf5_read_model_header_v9(file_id, model_data)	*Fortran*
	[model_data] = hdf5_read_model_header_v9(file_id);	*Matlab*

hdf5_read_individual_raw_data_v9

Purpose	This routine reads a block of time history data from an individual raw data file.
C Syntax	hdf5_read_individual_raw_data_v9(*file_id, raw_data*)
Fortran Syntax	hdf5_read_individual_raw_data_v9(*file_id, number_of_samples, raw_data*)
Matlab Syntax	[*raw_data*] = hdf5_read_individual_raw_data_v9(*file_id*)
Input Arguments	*file_id* HDF5 file ID number (integer) *number_of_samples* Number of time history data samples to read (integer)
Output Arguments	*raw_data* Time history data (1D floating point array)
Remarks	The file must first be opened with a call to *hdf5_open_raw_file_v9* before calling this routine.

Example Use:

C

```
hdf5_read_individual_raw_data_v9(file_id, raw_data);
```

Fortran

```
call hdf5_read_individual_raw_data_v9(file_id, 1500000, raw_data)
```

Matlab

```
[raw_data] = hdf5_read_individual_raw_data_v9(file_id);
```

Purpose	This routine reads a complete set of time history data from an ensemble raw data file.
C Syntax	hdf5_read_ensemble_raw_data_v9(*file_id, ensemble_raw_data*)
Fortran Syntax	hdf5_read_ensemble_raw_data_v9(*file_id, number_of_channels, number_of_samples, ensemble_raw_data*)
Matlab Syntax	[*ensemble_raw_data*] = hdf5_write_ensemble_raw_data_v9(*file_id*)
Input Arguments	*file_id* HDF5 file ID number (integer) *number_of_channels* Number of data channels to read (integer) *number_of_samples* Number of time history data samples to read per channel (integer)
Output Arguments	*ensemble_raw_data* Ensemble time history data (2D floating point array), dimension is (num_channels, num_samples)
Remarks	The file must first be opened with a call to *hdf5_open_raw_file_v9* before calling this routine.

Example Use:

C

```
hdf5_read_ensemble_raw_data_v9(file_id, 97, ensemble_raw_data);
```

Fortran

```
call hdf5_read_ensemble_raw_data_v9(file_id, 97, 1500000, ensemble_raw_data)
```

Matlab

```
[ensemble_raw_data] = hdf5_write_ensemble_raw_data_v9(file_id);
```

hdf5_read_pressure_data_v9

Purpose	This routine reads static pressure tap data from a pressure data file.
C Syntax	hdf5_read_pressure_data_v9(*file_id, static_data*)
Fortran Syntax	hdf5_read_pressure_data_v9(*file_id, static_data*)
Matlab Syntax	[static_data] = hdf5_read_pressure_data_v9(*file_id, static_data*)
Input Arguments	*file_id* HDF5 file ID number (integer) *static_data* Data structure containing static pressure tap data (see Appendices A, B, and C for format)
Output Arguments	*static_data* Data structure containing static pressure tap data
Remarks	The file must first be opened with a call to *hdf5_open_pressure_file_v9* before calling this routine.

Example Use:

```
hdf5_read_pressure_data_v9(file_id, static_data);                          C
call hdf5_read_pressure_data_v9(file_id, static_data)                      Fortran
[static_data] = hdf5_read_pressure_data_v9(file_id, static_data);          Matlab
```

hdf5_read_csm_data_v9

Purpose	This routine reads cross spectral matrix data from a CSM data file.
C Syntax	hdf5_read_csm_data_v9(*file_id, csm_real_data, csm_imag_data*)
Fortran Syntax	hdf5_read_csm_data_v9(*file_id, number_of_channels, number_of_bins, csm_real_data, csm_imag_data*)
Matlab Syntax	[*csm_real_data, csm_imag_data*] = hdf5_read_csm_data_v9(*file_id*)
Input Arguments	*file_id* HDF5 file ID number (integer) *number_of_channels* Number of channels contained in cross spectral matrix (integer) *number_of_bins* Number of frequency bins contained in cross spectral matrix (integer)
Output Arguments	*csm_real_data* 3D floating point array containing real component of cross spectral matrix, dimension is (num_channels, num_channels, num_bins) *csm_imag_data* 3D floating point array containing imaginary component of cross spectral matrix, dimension is (num_channels, num_channels, num_bins)
Remarks	The file must first be opened with a call to *hdf5_open_csm_file_v9* before calling this routine.

Example Use:

hdf5_read_csm_data_v9(file_id, csm_real_data, csm_imag_data);	*C*
call hdf5_read_csm_data_v9(file_id, 97, 4096, csm_real_data, csm_imag_data)	*Fortran*
[csm_real_data, csm_imag_data] = hdf5_read_csm_data_v9(file_id);	*Matlab*

10.0 References

[1] Information on the NetCDF data format and library can be found at https://www.unidata.ucar.edu/software/netcdf/

[2] Information on the HDF5 data format and library can be found at https://portal.hdfgroup.org/display/support

Appendix A.1 – C Data Structure for NetCDF / HDF5 Individual Raw Data Files (assumes pointer access to structure)

Variable Name	*Description*	*Data Type*
Version variables		
version_data->netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar float
version_data->hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar float
version_data->hdr_version	Header Version Number	scalar float
Test variables		
test_data->testno	Test Number	scalar long
test_data->runno	Run Number	scalar long
test_data->pointno	Point Number	scalar long
test_data->date_time	Acquistion Date and Time	string, length 17
test_data->GPS_start_time	Acquisition UTC Start Time	scalar float
test_data->GPS_stop_time	Acquisition UTC Stop Time	scalar float
test_data->test_engineer	Test Engineer Name	string, length 80
Channel variables		
channel_data->channo	Channel Number	scalar long
channel_data->nam_chan	Channel Name	string, length 80
channel_data->chan_description	Channel Description	string, length 80
channel_data->sensor_sn	Sensor Serial Number	scalar long
channel_data->preamplifier_sn	Preamplifier Serial Number	scalar long
channel_data->sensor_type	Sensor Type	scalar long
channel_data->preamplifier_type	Preamplifier Type	scalar long
channel_data->sensor_description	Sensor Description	string, length 80
channel_data->preamplifier_description	Preamplifier Description	string, length 80
channel_data->excitation_voltage	Excitation Voltage	scalar float
channel_data->excitation_current	Excitation Current	scalar float
channel_data->sensor_x	Sensor x Location	scalar float
channel_data->sensor_y	Sensor y Location	scalar float
channel_data->sensor_z	Sensor z Location	scalar float
channel_data->sensor_coord_ref	Sensor Coordinate Reference	string, length 12

Appendix A.1 – C Data Structure for NetCDF / HDF5 Individual Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Calibration variables		
calibration_data->sensitivity	Calibrated Sensor Sensitivity	scalar float
calibration_data->number_of_cal_sources	# of On-Board Calibration Sources	scalar long
calibration_data->cal_source_coordinates	X,Y,Z Coordinates of Sources	2D array of floats, dimension[num_sources][3]
calibration_data->cal_source_coord_ref	Coordinate System for Sources	string, length = 12
calibration_data->cal_source_baseline_array	Calibration Source Baseline Levels	1D array of floats, dimension[num_sources]
calibration_data->cal_source_level_array	Calibration Source Levels	1D array of floats, dimension[num_sources]
calibration_data->injection_cal_baseline	Injection Calibration Baseline Level	scalar float
calibration_data->injection_cal_level	Injection Calibration Recorded Level	scalar float
calibration_data->calibration_frequency	Frequency Response Frequency Array	1D array of floats, dimension[num_bins]
calibration_data->calibration_magnitude	Frequency Response Magnitude Array	1D array of floats, dimension[num_bins]
calibration_data->calibration_phase	Frequency Response Phase Array	1D array of floats, dimension[num_bins]
DAS variables		
das_data->total_channels	Total Channels in System	scalar long
das_data->adbits	A/D Resolution for Channel	scalar long
das_data->coupling	AC Coupling Indicator	scalar long
das_data->full_scale_range	Channel Full Scale Range	scalar float
das_data->trigger_mode	Trigger Mode for Channel	scalar long
das_data->clock_mode	Clock Mode for Channel	scalar long
das_data->sample_count	Channel Sample Count	scalar long
das_data->sample_period	Channel Sample Period	scalar float
das_data->lpf_cutoff_freq	Low Pass Filter Cutoff Frequency	scalar float
das_data->lpf_pregain	Low Pass Filter Pregain Setting	scalar float
das_data->lpf_postgain	Low Pass Filter Postgain Setting	scalar float
das_data->hpf_cutoff_freq	High Pass Filter Cutoff Frequency	scalar float
das_data->hpf_pregain	High Pass Filter Pregain Setting	scalar float
das_data->hpf_postgain	High Pass Filter Postgain Setting	scalar float
das_data->external_gain_offset	Sensor External Gain Offset	scalar float
das_data->on_board_LP_filter	On-board LP Filter Switch	scalar long

Appendix A.1 – C Data Structure for NetCDF / HDF5 Individual Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Array variables		
array_data->total_array_sensors	Number of Array Sensors	scalar long
array_data->traverse_configuration	Traverse Configuration Description	string, length 80
array_data->sideline_traverse_x	Sideline Traverse X Location	scalar float
array_data->sideline_traverse_y	Sideline Traverse Y Location	scalar float
array_data->sideline_traverse_z	Sideline Traverse Z Location	scalar float
array_data->sideline_traverse_coord_ref	Coordinate System for Sideline Traverse	string, length 12
array_data->array_traverse_x	Array Traverse X Location	scalar float
array_data->array_traverse_y	Array Traverse Y Location	scalar float
array_data->array_traverse_z	Array Traverse Z Location	scalar float
array_data->array_traverse_coord_ref	Coordinate System for Array Traverse	string, length 12
array_data->array_elevation	Array Face Elevation Angle	scalar float
array_data->array_azimuth	Array Face Azimuth Angle	scalar float
array_data->number_of_inclinometers	Number of Array Inclinometers	scalar long
array_data->inclinometer_coordinates	X,Y,Z Coordinates of Inclinometers	2D array of floats, dimension[num_inclinometers][3]
array_data->inclinometer_coord_ref	Coordinate System for Inclinometers	string, length 12
array_data->inclinometer_phi_array	Array Inclinometer Phi Array	1D array of floats, dimension[num_inclinometers]
array_data->inclinometer_theta_array	Array Inclinometer Theta Array	1D array of floats, dimension[num_inclinometers]
array_data->inclinometer_rotation_array	Array Inclinometer Rotation Array	1D array of floats, dimension[num_inclinometers]
array_data->inclinometer_temperature_array	Array Inclinometer Temperature Array	1D array of floats, dimension[num_inclinometers]
array_data->photogrammetric_x_location	Array X Location from Photogrammetry	scalar float
array_data->photogrammetric_y_location	Array Y Location from Photogrammetry	scalar float
array_data->photogrammetric_z_location	Array Z Location from Photogrammetry	scalar float
array_data->photogrammetric_yaw_angle	Array Yaw Angle from Photogrammetry	scalar float
array_data->photogrammetric_pitch_angle	Array Pitch Angle from Photogrammetry	scalar float
array_data->photogrammetric_roll_angle	Array Roll Angle from Photogrammetry	scalar float
array_data->photogrammetric_number_targets	Number of Photogrammetry Targets	scalar long
array_data->photogrammetric_RMS_error	Photogrammetry RMS Error	scalar float
Facility variables		
facility_data->facility_name	Name of Ground Test Facility	string, length 80
facility_data->facility_description	Description of Ground Test Facility	string, length 80
facility_data->total_facility_parameters	Total Number of Facility Parameters	scalar long
facility_data->facility_parameter_001	Test-specific Facility Parameter	string structure (see below)
.		
.		
facility_data->facility_parameter_200	Test-specific Facility Parameter	string structure

Appendix A.1 – C Data Structure for NetCDF / HDF5 Individual Raw Data Files (concluded)

Variable Name	*Description*	*Data Type*
Model variables		
model_data->model_id	Model Identification Code	scalar long
model_data->model_description	Description of Test Model	string, length 80
model_data->total_model_parameters	Total Number of Model Parameters	scalar long
model_data->model_parameter_001	Test-specific Model Parameter	string structure (see below)
.		
.		
model_data->model_parameter_200	Test-specific Model Parameter	string structure
Raw Data		
raw_data->data_values	Raw Data Values	1D array of floats, dimension[num_samples]

The string structures for the facility and model parameters are as follows:

facility_data->facility_parameter[*n*].long_name	Long name of parameter	string, length = 80
facility_data->facility_parameter[*n*].description	Detailed description of parameter	string, length = 80
facility_data->facility_parameter[*n*].data_type	Parameter data type	string, length = 80
facility_data->facility_parameter[*n*].units	Parameter units	string, length = 80
facility_data->facility_parameter[*n*].value	Parameter value	string, length = 80
model_data->model_parameter[*n*].long_name	Long name of parameter	string, length = 80
model_data->model_parameter[*n*].description	Detailed description of parameter	string, length = 80
model_data->model_parameter[*n*].data_type	Parameter data type	string, length = 80
model_data->model_parameter[*n*].units	Parameter units	string, length = 80
model_data->model_parameter[*n*].value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 0).

Appendix A.2 - C Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (assumes pointer access to structure)

Variable Name	*Description*	*Data Type*
Version variables		
ensemble_version_data->netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar float
ensemble_version_data->hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar float
ensemble_version_data->hdr_version	Header Version Number	scalar float
Test variables		
ensemble_test_data->testno	Test Number	scalar long
ensemble_test_data->runno	Run Number	scalar long
ensemble_test_data->pointno	Point Number	scalar long
ensemble_test_data->identifier_prefix	Acquisition Type Prefix Character	character
ensemble_test_data->date_time	Acquistion Date and Time	string, length 17
ensemble_test_data->GPS_start_time	Acquisition UTC Start Time	scalar float
ensemble_test_data->GPS_stop_time	Acquisition UTC Stop Time	scalar float
ensemble_test_data->test_engineer	Test Engineer Name	string, length 80
Channel variables		
ensemble_channel_data->channo_array	Channel Number Array	1D array of longs, dimension[num_channels]
ensemble_channel_data->sensor_sn_array	Sensor Serial Number Array	1D array of longs, dimension[num_channels]
ensemble_channel_data->preamplifier_sn_array	Preamplifier Serial Number Array	1D array of longs, dimension[num_channels]
ensemble_channel_data->sensor_type_array	Sensor Type Array	1D array of longs, dimension[num_channels]
ensemble_channel_data->preamplifier_type_array	Preamplifier Type Array	1D array of longs, dimension[num_channels]
ensemble_channel_data->excitation_voltage_array	Sensor Excitation Voltage Array	1D array of floats, dimension[num_channels]
ensemble_channel_data->excitation_current_array	Sensor Excitation Current Array	1D array of floats, dimension[num_channels]
ensemble_channel_data->sensor_coord_array	Array of Sensor X,Y,Z Locations	2D array of floats, dimension[num_channels][3]
ensemble_channel_data->sensor_coord_ref_array	Array Sensor Coordinate References	1D array of strings, length 12 dimension[num_channels]

Appendix A.2 - C Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Calibration variables		
ensemble_calibration_data->sensitivity_array	Sensor Sensitivity from Calibration	1D array of floats, dimension[num_channels]
ensemble_calibration_data->filter_response	Signal Conditioning Bandpass Filter Response	2D array of floats, dimension[num_bins][3]
ensemble_calibration_data->number_of_cal_sources	Number of On-Board Calibration Sources Used	scalar long
ensemble_calibration_data->cal_source_coordinates	X,Y,Z Coordinates of Calibration Sources	2D array of floats, dimension[num_sources][3]
ensemble_calibration_data->cal_source_coord_ref_array	Coordinate System for Calibration Source Locations	1D array of strings, length 12 dimension[num_sources]
ensemble_calibration_data->number_of_cal_measurements	Number of calibration measurements completed	scalar long
ensemble_calibration_data->date_time_of_cal	Array of source calibration dates and times	2D array of strings, length 17 dimension[num_sources][num_cals]
ensemble_calibration_data->cal_source_baseline_array	On-board Calibration Source Baseline Level Array	3D array of floats dimension[channels][sources][cals]
ensemble_calibration_data->cal_source_level_array	On-board Calibration Source Level Array	3D array of floats dimension[channels][sources][cals]
ensemble_calibration_data->injection_cal_baseline_array	Injection Calibration Baseline Level	1D array of floats, dimension[num_channels]
ensemble_calibration_data->injection_cal_level_array	Injection Calibration Recorded Level	1D array of floats, dimension[num_channels]
ensemble_calibration_data->freq_calibration_array	Sensor Frequency Response Frequency Array	3D array of floats, dimension[channels][bins][3]
DAS variables		
ensemble_das_data->total_channels	Total Number of Channels in System	scalar long
ensemble_das_data->adbits	A/D Resolution for Channel	scalar long
ensemble_das_data->coupling_array	AC Coupling Indicator Array	1D array of longs, dimension[num_channels]
ensemble_das_data->full_scale_range_array	Channel Full Scale Range Array	1D array of floats, dimension[num_channels]
ensemble_das_data->trigger_mode	Trigger Mode for Channel	scalar long
ensemble_das_data->clock_mode	Clock Mode for Channel	scalar long
ensemble_das_data->sample_count	Channel Sample Count	scalar long
ensemble_das_data->sample_period	Channel Sample Period	scalar float
ensemble_das_data->lpf_cutoff_freq_array	Low Pass Filter Cutoff Frequency Array	1D array of floats, dimension[num_channels]
ensemble_das_data->lpf_pregain_array	Low Pass Filter Pregain Setting Array	1D array of floats, dimension[num_channels]
ensemble_das_data->lpf_postgain_array	Low Pass Filter Postgain Setting Array	1D array of floats, dimension[num_channels]
ensemble_das_data->hpf_cutoff_freq_array	High Pass Filter Cutoff Frequency Array	1D array of floats, dimension[num_channels]
ensemble_das_data->hpf_pregain_array	High Pass Filter Pregain Setting Array	1D array of floats, dimension[num_channels]
ensemble_das_data->hpf_postgain_array	High Pass Filter Postgain Setting Array	1D array of floats, dimension[num_channels]
ensemble_das_data->external_gain_offset_array	Sensor Power Supply External Gain Offset Array	1D array of floats, dimension[num_channels]
ensemble_das_data->on_board_LP_filter_array	On-board LP Filter Boolean Switch Array	1D array of longs, dimension[num_channels]

Appendix A.2 - C Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Array variables		
ensemble_array_data->total_array_sensors	Total Number of Array Sensors	scalar long
ensemble_array_data->traverse_configuration	Traverse Configuration Description	string, length 80
ensemble_array_data->sideline_traverse_x	Sideline Traverse X Location	scalar float
ensemble_array_data->sideline_traverse_y	Sideline Traverse Y Location	scalar float
ensemble_array_data->sideline_traverse_z	Sideline Traverse Z Location	scalar float
ensemble_array_data->sideline_traverse_coord_ref	Coordinate System for Sideline Traverse	string, length 12
ensemble_array_data->array_traverse_x	Array Traverse X Location	scalar float
ensemble_array_data->array_traverse_y	Array Traverse Y Location	scalar float
ensemble_array_data->array_traverse_z	Array Traverse Z Location	scalar float
ensemble_array_data->array_traverse_coord_ref	Coordinate System for Array Traverse	string, length 12
ensemble_array_data->array_elevation	Array Face Elevation Angle	scalar float
ensemble_array_data->array_azimuth	Array Face Azimuth Angle	scalar float
ensemble_array_data->number_of_inclinometers	Number of Inclinometers Incorporated Into Array	scalar long
ensemble_array_data->inclinometer_coordinates	X,Y,Z Coordinates of Inclinometers	2D array of floats, dimension[# inclinometers][3]
ensemble_array_data->inclinometer_coord_ref	Coordinate System for Inclinometer Locations	string, length 12
ensemble_array_data->inclinometer_phi_array	Array Inclinometer Phi Reading Array	1D array of floats, dimension[# inclinometers]
ensemble_array_data->inclinometer_theta_array	Array Inclinometer Theta Reading Array	1D array of floats, dimension[# inclinometers]
ensemble_array_data->inclinometer_rotation_array	Array Inclinometer Rotation Reading Array	1D array of floats, dimension[# inclinometers]
ensemble_array_data->inclinometer_temperature_array	Array Inclinometer Temperature Reading Array	1D array of floats, dimension[# inclinometers]
ensemble_array_data->photogrammetric_x_location	Array X Location from Photogrammetry Measurements	scalar float
ensemble_array_data->photogrammetric_y_location	Array Y Location from Photogrammetry Measurements	scalar float
ensemble_array_data->photogrammetric_z_location	Array Z Location from Photogrammetry Measurements	scalar float
ensemble_array_data->photogrammetric_yaw_angle	Array Yaw Angle from Photogrammetry Measurements	scalar float
ensemble_array_data->photogrammetric_pitch_angle	Array Pitch Angle from Photogrammetry Measurements	scalar float
ensemble_array_data->photogrammetric_roll_angle	Array Roll Angle from Photogrammetry Measurements	scalar float
ensemble_array_data->photogrammetric_number_targets	Array Panel Number of Photogrammetry Targets Used	scalar long
ensemble_array_data->photogrammetric_RMS_error	Array Panel Photogrammetry RMS Error	scalar float
Facility variables		
facility_data->facility_name	Name of Ground Test Facility	string, length 80
facility_data->facility_description	Description of Ground Test Facility	string, length 80
facility_data->total_facility_parameters	Total Number of Facility Parameters	scalar long
facility_data->facility_parameter_001	Test-specific Facility Parameter	string structure
.		
.		
facility_data->facility_parameter_200	Test-specific Facility Parameter	string structure

Appendix A.2 - C Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (concluded)

Variable Name	*Description*	*Data Type*
Model variables		
model_data->model_id	Model Identification Code	scalar long
model_data->model_description	Description of Test Model	string, length 80
model_data->total_model_parameters	Total Number of Model Parameters	scalar long
model_data->model_parameter_001	Test-specific Model Parameter	string structure
.		
.		
model_data->model_parameter_200	Test-specific Model Parameter	string structure
Raw Data		
ensemble_raw_data->data_values	Raw Data Values	2D array of floats, dimension[channels][samples]

The string structures for the facility and model parameters are as follows:

facility_data->facility_parameter[*n*].long_name	Long name of parameter	string, length = 80
facility_data->facility_parameter[*n*].description	Detailed description of parameter	string, length = 80
facility_data->facility_parameter[*n*].data_type	Parameter data type	string, length = 80
facility_data->facility_parameter[*n*].units	Parameter units	string, length = 80
facility_data->facility_parameter[*n*].value	Parameter value	string, length = 80
model_data->model_parameter[*n*].long_name	Long name of parameter	string, length = 80
model_data->model_parameter[*n*].description	Detailed description of parameter	string, length = 80
model_data->model_parameter[*n*].data_type	Parameter data type	string, length = 80
model_data->model_parameter[*n*].units	Parameter units	string, length = 80
model_data->model_parameter[*n*].value	Parameter value	string, length = 80

where *n* is the parameter number (n starts at 0).

Appendix A.3 – C Data Structure for NetCDF / HDF5 Static Pressure Data Files (assumes pointer access to structure)

Variable Name	*Description*	*Data Type*
Version variables		
version_data->netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar float
version_data->hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar float
version_data->hdr_version	Header Version Number	scalar float
Test variables		
test_data->testno	Test Number	scalar long
test_data->runno	Run Number	scalar long
test_data->pointno	Point Number	scalar long
test_data->date_time	Acquistion Date and Time	string, length 17
test_data->GPS_start_time	Acquisition UTC Start Time	scalar float
test_data->GPS_stop_time	Acquisition UTC Stop Time	scalar float
test_data->test_engineer	Test Engineer Name	string, length 80
Facility variables		
facility_data->facility_name	Name of Ground Test Facility	string, length 80
facility_data->facility_description	Description of Ground Test Facility	string, length 80
facility_data->total_facility_parameters	Total Number of Facility Parameters	scalar long
facility_data->facility_parameter_001	Test-specific Facility Parameter	string structure (see below)
.		
.		
facility_data->facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data->model_id	Model Identification Code	scalar long
model_data->model_description	Description of Test Model	string, length 80
model_data->total_model_parameters	Total Number of Model Parameters	scalar long
model_data->model_parameter_001	Test-specific Model Parameter	string structure (see below)
.		
.		
model_data->model_parameter_200	Test-specific Model Parameter	string structure

Appendix A.3 – C Data Structure for NetCDF / HDF5 Static Pressure Data Files (concluded)

Variable Name	*Description*	*Data Type*
Static tap variables		
static_data->total_static_taps	Number of Static Taps in File	scalar long
static_data->static_tap_parameter_001	Static Tap Data Entry	string structure (see below)
.		
.		
static_data->static_tap_parameter_500	Static Tap Data Entry	string structure

The string structures for the facility and model parameters are as follows:

facility_data->facility_parameter[*n*].long_name	Long name of parameter	string, length = 80
facility_data->facility_parameter[*n*].description	Detailed description of parameter	string, length = 80
facility_data->facility_parameter[*n*].data_type	Parameter data type	string, length = 80
facility_data->facility_parameter[*n*].units	Parameter units	string, length = 80
facility_data->facility_parameter[*n*].value	Parameter value	string, length = 80
model_data->model_parameter[*n*].long_name	Long name of parameter	string, length = 80
model_data->model_parameter[*n*].description	Detailed description of parameter	string, length = 80
model_data->model_parameter[*n*].data_type	Parameter data type	string, length = 80
model_data->model_parameter[*n*].units	Parameter units	string, length = 80
model_data->model_parameter[*n*].value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 0).

The string structure for the static tap data is as follows:

static_data->static_tap_parameter[*n*].static_tap_designation	Static tap designation	string, length = 80
static_data->static_tap_parameter[*n*].description	Static tap description	string, length = 80
static_data->static_tap_parameter[*n*].units	Static tap pressure value units	string, length = 80
static_data->static_tap_parameter[*n*].coefficient_name	Static tap coefficient name	string, length = 80
static_data->static_tap_parameter[*n*].data_value	Static tap pressure value	string, length = 80

where *n* is the static tap number (*n* starts at 0).

Appendix A.4 - C Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (assumes pointer access to structure)

Variable Name	*Description*	*Data Type*
Version variables		
csm_version_data->netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar float
csm_version_data->hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar float
csm_version_data->hdr_version	Header Version Number	scalar float
Test variables		
csm_test_data->testno	Test Number	scalar long
csm_test_data->runno	Run Number	scalar long
csm_test_data->pointno	Point Number	scalar long
csm_test_data->date_time	Acquistion Date and Time	string, length 17
csm_test_data->GPS_start_time	Acquisition UTC Start Time	scalar float
csm_test_data->GPS_stop_time	Acquisition UTC Stop Time	scalar float
Channel variables		
csm_channel_data->sensor_sn_array	Sensor Serial Number Array	1D array of longs, dimension[num_channels]
csm_channel_data->preamplifier_sn_array	Preamplifier Serial Number Array	1D array of longs, dimension[num_channels]
csm_channel_data->sensor_type_array	Sensor Type Array	1D array of longs, dimension[num_channels]
csm_channel_data->preamplifier_type_array	Preamplifier Type Array	1D array of longs, dimension[num_channels]
csm_channel_data->excitation_voltage_array	Excitation Voltage Array	1D array of floats, dimension[num_channels]
csm_channel_data->excitation_current_array	Excitation Current Array	1D array of floats, dimension[num_channels]
csm_channel_data->sensor_coord_array	Array of Sensor X,Y,Z Locations	2D array of floats, dimension[num_channels][3]
csm_channel_data->array_sensor_coord_ref	Array Sensor Coordinate Reference	string, length 12
csm_channel_data->sideline_sensor_coord_ref	Sideline Sensor Coordinate Reference	string, length 12
csm_channel_data->kulite_sensor_coord_ref	Kulite Sensor Coordinate Reference	string, length 12
Calibration variables		
csm_calibration_data->sensitivity_array	Sensor Sensitivity from Calibration	1D array of floats, dimension[num_channels]
csm_calibration_data->freq_calibration_array	Sensor Frequency Response Frequency Array	3D array of floats, dimension[channels][bins][3]
csm_calibration_data->filter_response	Signal Conditioning Bandpass Filter Response	2D array of singles, dimension[num_bins][3]
csm_calibration_data->number_of_cal_sources	Number of On-Board Calibration Sources Used	scalar long
csm_calibration_data->cal_source_coordinates	X,Y,Z Coordinates of Calibration Sources	2D array of floats, dimension[num_channels][3]
csm_calibration_data->cal_source_coord_ref_array	Array of Coordinate Systems for Calibration Speaker Locations	1D array of strings, dimension[sources][12]
csm_calibration_data->number_of_cal_measurements	Number of On-Board Calibration Runs Performed	scalar long
csm_calibration_data->date_time_of_cal	Array of Dates and Times On-Board Calibration Runs Performed	2D array of strings, dimension[sources][cals][17]

Appendix A.4 - C Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
csm_calibration_data->cal_source_baseline_array	On-board Calibration Source Baseline Level Array	3D array of floats, dimension[channels][sources][cals]
csm_calibration_data->cal_source_level_array	On-board Calibration Source Level Array	3D array of floats, dimension[channels][sources][cals]
csm_calibration_data->injection_cal_baseline_array	Injection Calibration Baseline Level	1D array of floats, dimension[num_channels]
csm_calibration_data->injection_cal_level_array	Injection Calibration Recorded Level	1D array of floats, dimension[num_channels]
csm_calibration_data->sensitivity_correction_switch	Indicator for Sensitivity Adjustment Due to Source Calibration	scalar long
csm_calibration_data->temperature_correction_switch	Indicator for Sensitivity Adjustment Due to Temperature	scalar long
csm_calibration_data->6db_correction_switch	Indicator for 6 dB Spectral Subtraction for Sensor Flush Mounting	scalar long
DAS variables		
csm_das_data->total_channels	Total Number of Channels in System	scalar long
csm_das_data->sample_count	Channel Sample Count	scalar long
csm_das_data->sample_period	Channel Sample Period	scalar float
csm_das_data->full_scale_range_array	Channel Full Scale Range Array	1D array of floats, dimension[num_channels]
csm_das_data->lpf_cutoff_freq_array	Low Pass Filter Cutoff Frequency Array	1D array of floats, dimension[num_channels]
csm_das_data->lpf_pregain_array	Low Pass Filter Pregain Setting Array	1D array of floats, dimension[num_channels]
csm_das_data->lpf_postgain_array	Low Pass Filter Postgain Setting Array	1D array of floats, dimension[num_channels]
csm_das_data->hpf_cutoff_freq_array	High Pass Filter Cutoff Frequency Array	1D array of floats, dimension[num_channels]
csm_das_data->hpf_pregain_array	High Pass Filter Pregain Setting Array	1D array of floats, dimension[num_channels]
csm_das_data->hpf_postgain_array	High Pass Filter Postgain Setting Array	1D array of floats, dimension[num_channels]
csm_das_data->external_gain_offset_array	Sensor Power Supply External Gain Offset Array	1D array of floats, dimension[num_channels]
Array variables		
csm_array_data->total_array_sensors	Total Number of Array Sensors	scalar long
csm_array_data->traverse_configuration	Traverse Configuration Description	string, length 80
csm_array_data->sideline_traverse_x	Sideline Traverse X Location	scalar float
csm_array_data->sideline_traverse_y	Sideline Traverse Y Location	scalar float
csm_array_data->sideline_traverse_z	Sideline Traverse Z Location	scalar float
csm_array_data->sideline_traverse_coord_ref	Coordinate System Reference for Sideline Traverse	string, length 12
csm_array_data->array_traverse_x	Array Traverse X Location	scalar float
csm_array_data->array_traverse_y	Array Traverse Y Location	scalar float
csm_array_data->array_traverse_z	Array Traverse Z Location	scalar float
csm_array_data->array_traverse_coord_ref	Coordinate System Reference for Array Traverse	string, length 12
csm_array_data->array_elevation	Array Face Elevation Angle	scalar float
csm_array_data->array_azimuth	Array Face Azimuth Angle	scalar float
csm_array_data->number_of_inclinometers	Number of Inclinometers Incorporated Into Array	scalar long

Appendix A.4 - C Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
csm_array_data->inclinometer_coordinates	X,Y,Z Coordinates of Inclinometers	2D array of floats, dimension[num_inclinometers][3]
csm_array_data->inclinometer_coord_ref	Coordinate System Reference for Inclinometer Locations	string, length 12
csm_array_data->inclinometer_phi_array	Array Inclinometer Phi Reading Array	1D array of floats, dimension(num_inclinometers]
csm_array_data->inclinometer_theta_array	Array Inclinometer Theta Reading Array	1D array of floats, dimension[num_inclinometers]
csm_array_data->inclinometer_rotation_array	Array Inclinometer Rotation Reading Array	1D array of floats, dimension[num_inclinometers]
csm_array_data->inclinometer_temperature_array	Array Inclinometer Temperature Reading Array	1D array of floats, dimension[num_inclinometers]
csm_array_data->photogrammetric_x_location	Array Panel X Location from Photogrammetry Measurements	scalar float
csm_array_data->photogrammetric_y_location	Array Panel Y Location from Photogrammetry Measurements	scalar float
csm_array_data->photogrammetric_z_location	Array Panel Z Location from Photogrammetry Measurements	scalar float
csm_array_data->photogrammetric_yaw_angle	Array Panel Yaw Angle from Photogrammetry Measurements	scalar float
csm_array_data->photogrammetric_pitch_angle	Array Panel Pitch Angle from Photogrammetry Measurements	scalar float
csm_array_data->photogrammetric_roll_angle	Array Panel Roll Angle from Photogrammetry Measurements	scalar float
csm_array_data->photogrammetric_number_targets	Array Panel Number of Photogrammetry Targets Used	scalar long
csm_array_data->photogrammetric_RMS_error	Array Panel Photogrammetry RMS Error	scalar float
Facility variables		
facility_data->facility_name	Name of Ground Test Facility	string, length 80
facility_data->facility_description	Description of Ground Test Facility	string, length 80
facility_data->total_facility_parameters	Total Number of Facility Parameters	scalar long
facility_data->facility_parameter_001	Test-specific Facility Parameter	string structure (see below)
.		
.		
facility_data->facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data->model_id	Model Identification Code	scalar long
model_data->model_description	Description of Test Model	string, length 80
model_data->total_model_parameters	Total Number of Model Parameters	scalar long
model_data->model_parameter_001	Test-specific Model Parameter	string structure (see below)
.		
.		
model_data->model_parameter_200	Test-specific Model Parameter	string structure

Appendix A.4 - C Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (concluded)

Variable Name	*Description*	*Data Type*
CSM variables		
csm_data->fft_length	FFT Block Length	scalar long
csm_data->overlap	FFT Block Overlap	scalar long
csm_data->num_blocks	Total Number of FFT Blocks	scalar long
csm_data->num_blocks_used	Total Number of FFT Blocks Actually Used for Form CSM	scalar long
csm_data->block_table	Block Usage Table for CSM Generation	2D array of longs, dimension[num_blocks][num_channels]
csm_data->cslength	Saved Cross Spectral Length	scalar long
csm_data->winfunc	FFT Window Function Name	string, length 11
csm_data->beta	Kaiser Window Function Beta Value	scalar float
csm_data->CSM_units	Units for Cross Spectral Matrix Elements	scalar long
csm_data->csm_real	Cross Spectral Matrix Real Data Matrix	3D array of floats dimension[channels][channels][num_bins]
csm_data->csm_imag	Cross Spectral Matrix Imaginary Data Matrix	3D array of floats dimension[channels][channels][num_bins]

The string structures for the facility and model parameters are as follows:

facility_data->facility_parameter[*n*].long_name	Long name of parameter	string, length = 80
facility_data->facility_parameter[*n*].description	Detailed description of parameter	string, length = 80
facility_data->facility_parameter[*n*].data_type	Parameter data type	string, length = 80
facility_data->facility_parameter[*n*].units	Parameter units	string, length = 80
facility_data->facility_parameter[*n*].value	Parameter value	string, length = 80
model_data->model_parameter[*n*].long_name	Long name of parameter	string, length = 80
model_data->model_parameter[*n*].description	Detailed description of parameter	string, length = 80
model_data->model_parameter[*n*].data_type	Parameter data type	string, length = 80
model_data->model_parameter[*n*].units	Parameter units	string, length = 80
model_data->model_parameter[*n*].value	Parameter value	string, length = 80

where *n* is the parameter number (*n* start at 0).

Appendix B.1 – Fortran Data Structure for NetCDF / HDF5 Individual Raw Data Files

Variable Name	*Description*	*Data Type*
Version variables		
version_data%netcdf_version	NetCDF Library Version Number (NetCDF Files)	scalar real
version_data%hdf5_version	HDF5 Library Version Number (HDF5 Files)	scalar real
version_data%hdr_version	Header Version Number	scalar real
Test variables		
test_data%testno	Test Number	scalar integer
test_data%runno	Run Number	scalar integer
test_data%pointno	Point Number	scalar integer
test_data%date_time	Acquistion Date and Time	string, length = 16
test_data%GPS_start_time	Acquisition UTC Start Time	scalar real
test_data%GPS_stop_time	Acquisition UTC Stop Time	scalar real
test_data%test_engineer	Test Engineer Name	string, length = 80
Channel variables		
channel_data%channo	Channel Number	scalar integer
channel_data%nam_chan	Channel Name	string, length = 80
channel_data%chan_description	Channel Description	string, length = 80
channel_data%sensor_sn	Sensor Serial Number	scalar integer
channel_data%preamplifier_sn	Preamplifier Serial Number	scalar integer
channel_data%sensor_type	Sensor Type	scalar integer
channel_data%preamplifier_type	Preamplifier Type	scalar integer
channel_data%sensor_description	Sensor Description	string, length = 80
channel_data%preamplifier_description	Preamplifier Description	string, length = 80
channel_data%excitation_voltage	Excitation Voltage	scalar real
channel_data%excitation_current	Excitation Current	scalar real
channel_data%sensor_x	Sensor x Location	scalar real
channel_data%sensor_y	Sensor y Location	scalar real
channel_data%sensor_z	Sensor z Location	scalar real
channel_data%sensor_coord_ref	Sensor Coordinate Reference	string, length = 12

Appendix B.1 – Fortran Data Structure for NetCDF / HDF5 Individual Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Calibration variables		
calibration_data%sensitivity	Sensor Sensitivity from Calibration	scalar real
calibration_data%number_of_cal_sources	Number of On-Board Calibration Sources Used	scalar integer
calibration_data%cal_source_coordinates	X,Y,Z Coordinates of Calibration Sources	2D real array, dimension(num_sources, 3)
calibration_data%cal_source_coord_ref	Coordinate System for Calibration Source Locations	string, length = 12
calibration_data%cal_source_baseline_array	On-board Calibration Source Baseline Level Array	1D real array, dimension(num_sources)
calibration_data%cal_source_level_array	On-board Calibration Source Level Array	1D real array, dimension(num_sources)
calibration_data%injection_cal_baseline	Injection Calibration Baseline Level	scalar real
calibration_data%injection_cal_level	Injection Calibration Recorded Level	scalar real
calibration_data%calibration_frequency	Sensor Frequency Response Frequency Array	1D real array, dimension(num_bins)
calibration_data%calibration_magnitude	Sensor Frequency Response Magnitude Array	1D real array, dimension(num_bins)
calibration_data%calibration_phase	Sensor Frequency Response Phase Array	1D real array, dimension(num_bins)
DAS variables		
DAS_data%total_channels	Total Number of Channels in System	scalar integer
DAS_data%adbits	A/D Resolution for Channel	scalar integer
DAS_data%coupling	AC Coupling Indicator	scalar integer
DAS_data%full_scale_range	Channel Full Scale Range	scalar real
DAS_data%trigger_mode	Trigger Mode for Channel	scalar integer
DAS_data%clock_mode	Clock Mode for Channel	scalar integer
DAS_data%sample_count	Channel Sample Count	scalar integer
DAS_data%sample_period	Channel Sample Period	scalar real
DAS_data%lpf_cutoff_freq	Low Pass Filter Cutoff Frequency	scalar real
DAS_data%lpf_pregain	Low Pass Filter Pregain Setting	scalar real
DAS_data%lpf_postgain	Low Pass Filter Postgain Setting	scalar real
DAS_data%hpf_cutoff_freq	High Pass Filter Cutoff Frequency	scalar real
DAS_data%hpf_pregain	High Pass Filter Pregain Setting	scalar real
DAS_data%hpf_postgain	High Pass Filter Postgain Setting	scalar real
DAS_data%external_gain_offset	Sensor Power Supply External Gain Offset	scalar real
DAS_data%on_board_LP_filter	On-board LP Filter Boolean Switch	scalar integer

Appendix B.1 – Fortran Data Structure for NetCDF / HDF5 Individual Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Array variables		
array_data%total_array_sensors	Total Number of Array Sensors	scalar integer
array_data%traverse_configuration	Traverse Configuration Description	string, length = 80
array_data%sideline_traverse_x	Sideline Traverse X Location	scalar real
array_data%sideline_traverse_y	Sideline Traverse Y Location	scalar real
array_data%sideline_traverse_z	Sideline Traverse Z Location	scalar real
array_data%sideline_traverse_coord_ref	Coordinate System for Sideline Traverse	string, length = 80
array_data%array_traverse_x	Array Traverse X Location	scalar real
array_data%array_traverse_y	Array Traverse Y Location	scalar real
array_data%array_traverse_z	Array Traverse Z Location	scalar real
array_data%array_traverse_coord_ref	Coordinate System for Array Traverse	string, length = 12
array_data%array_elevation	Array Face Elevation Angle	scalar real
array_data%array_azimuth	Array Face Azimuth Angle	scalar real
array_data%number_of_inclinometers	Number of Inclinometers Incorporated Into Array	scalar integer
array_data%inclinometer_coordinates	X,Y,Z Coordinates of Inclinometers	real array, dimension(num_inclinometers, 3)
array_data%inclinometer_coord_ref	Coordinate System for Inclinometer Locations	string, length = 12
array_data%inclinometer_phi_array	Array Inclinometer Phi Reading Array	real array, dimension(num_inclinometers)
array_data%inclinometer_theta_array	Array Inclinometer Theta Reading Array	real array, dimension(num_inclinometers)
array_data%inclinometer_rotation_array	Array Inclinometer Rotation Reading Array	real array, dimension(num_inclinometers)
array_data%inclinometer_temperature_array	Array Inclinometer Temperature Reading Array	real array, dimension(num_inclinometers)
array_data%photogrammetric_x_location	Array Panel X Location from Photogrammetry	scalar real
array_data%photogrammetric_y_location	Array Panel Y Location from Photogrammetry	scalar real
array_data%photogrammetric_z_location	Array Panel Z Location from Photogrammetry	scalar real
array_data%photogrammetric_yaw_angle	Array Panel Yaw Angle from Photogrammetry	scalar real
array_data%photogrammetric_pitch_angle	Array Panel Pitch Angle from Photogrammetry	scalar real
array_data%photogrammetric_roll_angle	Array Panel Roll Angle from Photogrammetry	scalar real
array_data%photogrammetric_number_targets	Array Panel Number of Photogrammetry Targets	scalar integer
array_data%photogrammetric_RMS_error	Array Panel Photogrammetry RMS Error	scalar real

Appendix B.1 – Fortran Data Structure for NetCDF / HDF5 Individual Raw Data Files (concluded)

Variable Name	*Description*	*Data Type*
Facility variables		
facility_data%facility_name	Name of Ground Test Facility	string, length = 80
facility_data%facility_description	Description of Ground Test Facility	string, length = 80
facility_data%total_facility_parameters	Total Number of Facility Parameters	scalar integer
facility_data%facility_parameter_001	Test-specific Facility Parameter	string structure (see below)
.		
.		
facility_data%facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data%model_id	Model Identification Code	scalar integer
model_data%model_description	Description of Test Model	string, length = 80
model_data%total_model_parameters	Total Number of Model Parameters	scalar integer
model_data%model_parameter_001	Test-specific Model Parameter	string structure (see below)
.		
.		
model_data%model_parameter_200	Test-specific Model Parameter	string structure
Raw Data		
raw_data%data_values	Raw Data Values	real array, dimension(num_samples)

The string structures for the facility and model parameters are as follows:

facility_data%facility_parameter(*n*)%long_name	Long name of parameter	string, length = 80
facility_data%facility_parameter(*n*)%description	Detailed description of parameter	string, length = 80
facility_data%facility_parameter(*n*)%data_type	Parameter data type	string, length = 80
facility_data%facility_parameter(*n*)%units	Parameter units	string, length = 80
facility_data%facility_parameter(*n*)%value	Parameter value	string, length = 80
model_data%model_parameter(*n*)%long_name	Long name of parameter	string, length = 80
model_data%model_parameter(*n*)%description	Detailed description of parameter	string, length = 80
model_data%model_parameter(*n*)%data_type	Parameter data type	string, length = 80
model_data%model_parameter(*n*)%units	Parameter units	string, length = 80
model_data%model_parameter(*n*)%value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 1).

Appendix B.2 - Fortran Data Structure for NetCDF / HDF5 Ensemble Raw Data Files

Variable Name	*Description*	*Data Type*
Version variables		
ensemble_version_data%netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar real
ensemble_version_data%hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar real
ensemble_version_data%hdr_version	Header Version Number	scalar real
Test variables		
ensemble_test_data%testno	Test Number	scalar integer
ensemble_test_data%runno	Run Number	scalar integer
ensemble_test_data%pointno	Point Number	scalar integer
ensemble_test_data%identifier_prefix	Acquisition Type Prefix Character	character
ensemble_test_data%date_time	Acquistion Date and Time	string, length = 17
ensemble_test_data%GPS_start_time	Acquisition UTC Start Time	scalar real
ensemble_test_data%GPS_stop_time	Acquisition UTC Stop Time	scalar real
ensemble_test_data%test_engineer	Test Engineer Name	string, length = 80
Channel variables		
ensemble_channel_data%channo_array	Channel Number Array	1D integer array, dimension(num_channels)
ensemble_channel_data%sensor_sn_array	Sensor Serial Number Array	1D integer array, dimension(num_channels)
ensemble_channel_data%preamplifier_sn_array	Preamplifier Serial Number Array	1D integer array, dimension(num_channels)
ensemble_channel_data%sensor_type_array	Sensor Type Array	1D integer array, dimension(num_channels)
ensemble_channel_data%preamplifier_type_array	Preamplifier Type Array	1D integer array, dimension(num_channels)
ensemble_channel_data%excitation_voltage_array	Excitation Voltage Array	1D real array, dimension(num_channels)
ensemble_channel_data%excitation_current_array	Excitation Current Array	1D real array, dimension(num_channels)
ensemble_channel_data%sensor_coord_array	Array of Sensor X,Y,Z Locations	2D real array, dimension(num_channels, 3)
ensemble_channel_data%sensor_coord_ref_array	Array Sensor Coordinate References	1D string array, length = 12, dimension(num_channels)

Appendix B.2 - Fortran Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Calibration variables		
ensemble_calibration_data%sensitivity_array	Sensor Sensitivity from Calibration	1D real array, dimension(num_channels)
ensemble_calibration_data%filter_response	Signal Conditioning Bandpass Filter Response	2D real array, dimension(num_bins, 3)
ensemble_calibration_data%number_of_cal_sources	Number of On-Board Calibration Sources Used	scalar integer
ensemble_calibration_data%cal_source_coordinates	X,Y,Z Coordinates of Calibration Sources	real array, dimension(num_sources, 3)
ensemble_calibration_data%cal_source_coord_ref_array	Coordinate System for Calibration Source Locations	1D string array, length = 12, dimension(num_sources)
ensemble_calibration_data%number_of_cal_measurements	Number of calibration measurements completed	scalar integer
ensemble_calibration_data%date_time_of_cal	Array of source calibration dates and times	2D string array, length = 16, dimension(sources,calibrations)
ensemble_calibration_data%cal_source_baseline_array	On-board Calibration Source Baseline Level Array	3D real array, dimension(channels,sources,cals)
ensemble_calibration_data%cal_source_level_array	On-board Calibration Source Level Array	3D real array, dimension(channels,sources,cals)
ensemble_calibration_data%injection_cal_baseline_array	Injection Calibration Baseline Level	1D real array, dimension(num_channels)
ensemble_calibration_data%injection_cal_level_array	Injection Calibration Recorded Level	1D real array, dimension(num_channels)
ensemble_calibration_data%freq_calibration_array	Sensor Frequency Response Frequency Array	3D real array dimension(num_channels,num_bins,3)

Appendix B.2 - Fortran Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
DAS variables		
ensemble_DAS_data%total_channels	Total Number of Channels in System	scalar integer
ensemble_DAS_data%adbits	A/D Resolution for Channel	scalar integer
ensemble_DAS_data%coupling_array	AC Coupling Indicator Array	1D integer array, dimension(num_channels)
ensemble_DAS_data%full_scale_range_array	Channel Full Scale Range Array	1D real array, dimension(num_channels)
ensemble_DAS_data%trigger_mode	Trigger Mode for Channel	scalar integer
ensemble_DAS_data%clock_mode	Clock Mode for Channel	scalar integer
ensemble_DAS_data%sample_count	Channel Sample Count	scalar integer
ensemble_DAS_data%sample_period	Channel Sample Period	scalar real
ensemble_DAS_data%lpf_cutoff_freq_array	Low Pass Filter Cutoff Frequency Array	1D real array, dimension(num_channels)
ensemble_DAS_data%lpf_pregain_array	Low Pass Filter Pregain Setting Array	1D real array, dimension(num_channels)
ensemble_DAS_data%lpf_postgain_array	Low Pass Filter Postgain Setting Array	1D real array, dimension(num_channels)
ensemble_DAS_data%hpf_cutoff_freq_array	High Pass Filter Cutoff Frequency Array	1D real array, dimension(num_channels)
ensemble_DAS_data%hpf_pregain_array	High Pass Filter Pregain Setting Array	1D real array, dimension(num_channels)
ensemble_DAS_data%hpf_postgain_array	High Pass Filter Postgain Setting Array	1D real array, dimension(num_channels)
ensemble_DAS_data%external_gain_offset_array	Sensor Power Supply External Gain Offset Array	1D real array, dimension(num_channels)
ensemble_DAS_data%on_board_LP_filter_array	On-board LP Filter Boolean Switch Array	1D integer array, dimension(num_channels)

Appendix B.2 - Fortran Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Array variables		
ensemble_array_data%total_array_sensors	Total Number of Array Sensors	scalar integer
ensemble_array_data%traverse_configuration	Traverse Configuration Description	string, length = 80
ensemble_array_data%sideline_traverse_x	Sideline Traverse X Location	scalar real
ensemble_array_data%sideline_traverse_y	Sideline Traverse Y Location	scalar real
ensemble_array_data%sideline_traverse_z	Sideline Traverse Z Location	scalar real
ensemble_array_data%sideline_traverse_coord_ref	Coordinate System for Sideline Traverse	string, length = 12
ensemble_array_data%array_traverse_x	Array Traverse X Location	scalar real
ensemble_array_data%array_traverse_y	Array Traverse Y Location	scalar real
ensemble_array_data%array_traverse_z	Array Traverse Z Location	scalar real
ensemble_array_data%array_traverse_coord_ref	Coordinate System for Array Traverse	string, length = 12
ensemble_array_data%array_elevation	Array Face Elevation Angle	scalar real
ensemble_array_data%array_azimuth	Array Face Azimuth Angle	scalar real
ensemble_array_data%number_of_inclinometers	Number of Inclinometers Incorporated Into Array	scalar integer
ensemble_array_data%inclinometer_coordinates	X,Y,Z Coordinates of Inclinometers	2D real array, dimension(num_inclinometers,3)
ensemble_array_data%inclinometer_coord_ref	Coordinate System for Inclinometer Locations	string, length = 12
ensemble_array_data%inclinometer_phi_array	Array Inclinometer Phi Reading Array	1D real array, dimension(num_inclinometers)
ensemble_array_data%inclinometer_theta_array	Array Inclinometer Theta Reading Array	1D real array, dimension(num_inclinometers)
ensemble_array_data%inclinometer_rotation_array	Array Inclinometer Rotation Reading Array	1D real array, dimension(num_inclinometers)
ensemble_array_data%inclinometer_temperature_array	Array Inclinometer Temperature Reading Array	1D real array, dimension(num_inclinometers)
ensemble_array_data%photogrammetric_x_location	Array Panel X Location from Photogrammetry	scalar real
ensemble_array_data%photogrammetric_y_location	Array Panel Y Location from Photogrammetry	scalar real
ensemble_array_data%photogrammetric_z_location	Array Panel Z Location from Photogrammetry	scalar real
ensemble_array_data%photogrammetric_yaw_angle	Array Panel Yaw Angle from Photogrammetry	scalar real
ensemble_array_data%photogrammetric_pitch_angle	Array Panel Pitch Angle from Photogrammetry	scalar real
ensemble_array_data%photogrammetric_roll_angle	Array Panel Roll Angle from Photogrammetry	scalar real
ensemble_array_data%photogrammetric_number_targets	Array Panel Number of Photogrammetry Targets Used	scalar integer
ensemble_array_data%photogrammetric_RMS_error	Array Panel Photogrammetry RMS Error	scalar real

Appendix B.2 - Fortran Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (concluded)

Variable Name	*Description*	*Data Type*
Facility variables		
facility_data%facility_name	Name of Ground Test Facility	string, length = 80
facility_data%facility_description	Description of Ground Test Facility	string, length = 80
facility_data%total_facility_parameters	Total Number of Facility Parameters	scalar integer
facility_data%facility_parameter_001	Test-specific Facility Parameter	string structure
.		
.		
facility_data%facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data%model_id	Model Identification Code	scalar integer
model_data%model_description	Description of Test Model	string, length = 80
model_data%total_model_parameters	Total Number of Model Parameters	scalar integer
model_data%model_parameter_001	Test-specific Model Parameter	string structure
.		
.		
model_data%model_parameter_200	Test-specific Model Parameter	string structure
Raw Data		
ensemble_raw_data%data_values	Raw Data Values	2D real array, dimension(channels,samples)

The string structures for the facility and model parameters are as follows:

facility_data%facility_parameter(*n*)%long_name	Long name of parameter	string, length = 80
facility_data%facility_parameter(*n*)%description	Detailed description of parameter	string, length = 80
facility_data%facility_parameter(*n*)%data_type	Parameter data type	string, length = 80
facility_data%facility_parameter(*n*)%units	Parameter units	string, length = 80
facility_data%facility_parameter(*n*)%value	Parameter value	string, length = 80
model_data%model_parameter(*n*)%long_name	Long name of parameter	string, length = 80
model_data%model_parameter(*n*)%description	Detailed description of parameter	string, length = 80
model_data%model_parameter(*n*)%data_type	Parameter data type	string, length = 80
model_data%model_parameter(*n*)%units	Parameter units	string, length = 80
model_data%model_parameter(*n*)%value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 1).

Appendix B.3 – Fortran Data Structure for NetCDF / HDF5 Static Pressure Data Files

Variable Name	*Description*	*Data Type*
Version variables		
version_data%netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar real
version_data%hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar real
version_data%hdr_version	Header Version Number	scalar real
Test variables		
test_data%testno	Test Number	scalar integer
test_data%runno	Run Number	scalar integer
test_data%pointno	Point Number	scalar integer
test_data%date_time	Acquistion Date and Time	string, length 17
test_data%GPS_start_time	Acquisition UTC Start Time	scalar real
test_data%GPS_stop_time	Acquisition UTC Stop Time	scalar real
test_data%test_engineer	Test Engineer Name	string, length 80
Facility variables		
facility_data%facility_name	Name of Ground Test Facility	string, length 80
facility_data%facility_description	Description of Ground Test Facility	string, length 80
facility_data%total_facility_parameters	Total Number of Facility Parameters	scalar integer
facility_data%facility_parameter_001	Test-specific Facility Parameter	string structure
.		
.		
facility_data%facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data%model_id	Model Identification Code	scalar integer
model_data%model_description	Description of Test Model	string, length 80
model_data%total_model_parameters	Total Number of Model Parameters	scalar integer
model_data%model_parameter_001	Test-specific Model Parameter	string structure
.		
.		
model_data%model_parameter_200	Test-specific Model Parameter	string structure

Appendix B.3 – Fortran Data Structure for NetCDF / HDF5 Static Pressure Data Files (concluded)

Variable Name	*Description*	*Data Type*
Static tap variables		
static_data%total_static_taps	Number of Static Taps in File	scalar integer
static_data%static_tap_parameter_001	Static Tap Data Entry	string structure
.		
.		
static_data%static_tap_parameter_500	Static Tap Data Entry	string structure

The string structures for the facility and model parameters are as follows:

facility_data%facility_parameter(*n*)%long_name	Long name of parameter	string, length = 80
facility_data%facility_parameter(*n*)%description	Detailed description of parameter	string, length = 80
facility_data%facility_parameter(*n*)%data_type	Parameter data type	string, length = 80
facility_data%facility_parameter(*n*)%units	Parameter units	string, length = 80
facility_data%facility_parameter(*n*)%value	Parameter value	string, length = 80
model_data%model_parameter(*n*)%long_name	Long name of parameter	string, length = 80
model_data%model_parameter(*n*)%description	Detailed description of parameter	string, length = 80
model_data%model_parameter(*n*)%data_type	Parameter data type	string, length = 80
model_data%model_parameter(*n*)%units	Parameter units	string, length = 80
model_data%model_parameter(*n*)%value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 1).

The string structure for the static tap data is as follows:

static_data%static_tap_parameter(*n*)%static_tap_designation	Static tap designation	string, length = 80
static_data%static_tap_parameter(*n*)%description	Static tap description	string, length = 80
static_data%static_tap_parameter(*n*)%units	Static tap pressure value units	string, length = 80
static_data%static_tap_parameter(*n*)%coefficient_name	Static tap coefficient name	string, length = 80
static_data%static_tap_parameter(*n*)%data_value	Static tap pressure value	string, length = 80

where *n* is the static tap number (*n* starts at 1).

Appendix B.4 - Fortran Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files

Variable Name	*Description*	*Data Type*
Version variables		
csm_version_data%netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar real
csm_version_data%hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar real
csm_version_data%hdr_version	Header Version Number	scalar real
Test variables		
csm_test_data%testno	Test Number	scalar integer
csm_test_data%runno	Run Number	scalar integer
csm_test_data%pointno	Point Number	scalar integer
csm_test_data%date_time	Acquisition Date and Time	string, length = 16
csm_test_data%GPS_start_time	Acquisition UTC Start Time	scalar real
csm_test_data%GPS_stop_time	Acquisition UTC Stop Time	scalar real
Channel variables		
csm_channel_data%sensor_sn_array	Sensor Serial Number Array	1D integer array, dimension(num_channels)
csm_channel_data%preamplifier_sn_array	Preamplifier Serial Number Array	1D integer array, dimension(num_channels)
csm_channel_data%sensor_type_array	Sensor Type Array	1D integer array, dimension(num_channels)
csm_channel_data%preamplifier_type_array	Preamplifier Type Array	1D integer array, dimension(num_channels)
csm_channel_data%excitation_voltage_array	Excitation Voltage Array	1D real array, dimension(num_channels)
csm_channel_data%excitation_current_array	Excitation Current Array	1D real array, dimension(num_channels)
csm_channel_data%sensor_coord_array	Array of Sensor X,Y,Z Locations	2D real array, dimension(num_channels,3)
csm_channel_data%array_sensor_coord_ref	Array Sensor Coordinate Reference	string, length = 12
csm_channel_data%sideline_sensor_coord_ref	Sideline Sensor Coordinate Reference	string, length = 12
csm_channel_data%kulite_sensor_coord_ref	Kulite Sensor Coordinate Reference	string, length = 12

Appendix B.4 - Fortran Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
Calibration variables		
csm_calibration_data%sensitivity_array	Sensor Sensitivity from Calibration	1D real array, dimension(num_channels)
csm_calibration_data%freq_calibration_array	Sensor Frequency Response Frequency Array	3D real array. dimension(num_channels,num_bins,3)
csm_calibration_data%filter_response	Signal Conditioning Bandpass Filter Response	2D real array, dimension(num_bins,3)
csm_calibration_data%number_of_cal_sources	Number of On-Board Calibration Sources Used	scalar integer
csm_calibration_data%cal_source_coordinates	X,Y,Z Coordinates of Calibration Sources	2D real array, dimension(num_sources,3)
csm_calibration_data%cal_source_coord_ref_array	Array of Coordinate Systems for Calibration Speaker Locations	1D string array, length = 12, dimension(num_sources)
csm_calibration_data%number_of_cal_measurements	Number of On-Board Calibration Runs Performed	scalar integer
csm_calibration_data%date_time_of_cal	Array of Dates and Times On-Board Calibration Runs Performed	1D string array, length = 16, dimension(num_cals)
csm_calibration_data%cal_source_baseline_array	On-board Calibration Source Baseline Level Array	3D real array, dimension(channels,sources,cals)
csm_calibration_data%cal_source_level_array	On-board Calibration Source Level Array	3D real array, dimension(channels,sources,cals)
csm_calibration_data%injection_cal_baseline_array	Injection Calibration Baseline Level	1D real array, dimension(num_channels)
csm_calibration_data%injection_cal_level_array	Injection Calibration Recorded Level	1D real array, dimension(num_channels)
csm_calibration_data%sensitivity_correction_switch	Indicator for Sensitivity Adjustment Due to Source Calibration	scalar integer
csm_calibration_data%temperature_correction_switch	Indicator for Sensitivity Adjustment Due to Temperature	scalar integer
csm_calibration_data%6db_correction_switch	Indicator for 6 dB Subtraction from Spectra for Sensor Flush Mounting	scalar integer

Appendix B.4 - Fortran Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
DAS variables		
csm_DAS_data%total_channels	Total Number of Channels in System	scalar integer
csm_DAS_data%sample_count	Channel Sample Count	scalar integer
csm_DAS_data%sample_period	Channel Sample Period	scalar real
csm_DAS_data%full_scale_range_array	Channel Full Scale Range Array	1D real array, dimension(num_channels)
csm_DAS_data%lpf_cutoff_freq_array	Low Pass Filter Cutoff Frequency Array	1D real array, dimension(number_of_channels))
csm_DAS_data%lpf_pregain_array	Low Pass Filter Pregain Setting Array	1D real array, dimension(num_channels)
csm_DAS_data%lpf_postgain_array	Low Pass Filter Postgain Setting Array	1D real array. dimension(num_channels)
csm_DAS_data%hpf_cutoff_freq_array	High Pass Filter Cutoff Frequency Array	1D real array, dimension(num_channels)
csm_DAS_data%hpf_pregain_array	High Pass Filter Pregain Setting Array	1D real array, dimension(num_channels)
csm_DAS_data%hpf_postgain_array	High Pass Filter Postgain Setting Array	1D real array, dimension(num_channels)
csm_DAS_data%external_gain_offset_array	Sensor Power Supply External Gain Offset Array	1D real array, dimension(num_channels)

Appendix B.4 - Fortran Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
Array variables		
csm_array_data%total_array_sensors	Total Number of Array Sensors	scalar integer
csm_array_data%traverse_configuration	Traverse Configuration Description	string, length = 80
csm_array_data%sideline_traverse_x	Sideline Traverse X Location	scalar real
csm_array_data%sideline_traverse_y	Sideline Traverse Y Location	scalar real
csm_array_data%sideline_traverse_z	Sideline Traverse Z Location	scalar real
csm_array_data%sideline_traverse_coord_ref	Coordinate System for Sideline Traverse	string, length = 12
csm_array_data%array_traverse_x	Array Traverse X Location	scalar real
csm_array_data%array_traverse_y	Array Traverse Y Location	scalar real
csm_array_data%array_traverse_z	Array Traverse Z Location	scalar real
csm_array_data%array_traverse_coord_ref	Coordinate System for Array Traverse	string, length = 12
csm_array_data%array_elevation	Array Face Elevation Angle	scalar real
csm_array_data%array_azimuth	Array Face Azimuth Angle	scalar real
csm_array_data%number_of_inclinometers	Number of Inclinometers Incorporated Into Array	scalar integer
csm_array_data%inclinometer_coordinates	X,Y,Z Coordinates of Inclinometers	2D real array, dimension(num_inclinometers,3)
csm_array_data%inclinometer_coord_ref	Coordinate System for Inclinometer Locations	string, length = 12
csm_array_data%inclinometer_phi_array	Array Inclinometer Phi Reading Array	1D real array, dimension(num_inclinometers)
csm_array_data%inclinometer_theta_array	Array Inclinometer Theta Reading Array	1D real array, dimension(num_inclinometers)
csm_array_data%inclinometer_rotation_array	Array Inclinometer Rotation Reading Array	1D real array, dimension(num_inclinometers)
csm_array_data%inclinometer_temperature_array	Array Inclinometer Temperature Reading Array	1D real array, dimension(num_inclinometers)
csm_array_data%photogrammetric_x_location	Array Panel X Location from Photogrammetry Measurements	scalar real
csm_array_data%photogrammetric_y_location	Array Panel Y Location from Photogrammetry Measurements	scalar real
csm_array_data%photogrammetric_z_location	Array Panel Z Location from Photogrammetry Measurements	scalar real
csm_array_data%photogrammetric_yaw_angle	Array Panel Yaw Angle from Photogrammetry Measurements	scalar real
csm_array_data%photogrammetric_pitch_angle	Array Panel Pitch Angle from Photogrammetry Measurements	scalar real
csm_array_data%photogrammetric_roll_angle	Array Panel Roll Angle from Photogrammetry Measurements	scalar real
csm_array_data%photogrammetric_number_targets	Array Panel Number of Photogrammetry Targets Used	scalar integer
csm_array_data%photogrammetric_RMS_error	Array Panel Photogrammetry RMS Error	scalar real

Appendix B.4 - Fortran Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
Facility variables		
facility_data%facility_name	Name of Ground Test Facility	string, length = 80
facility_data%facility_description	Description of Ground Test Facility	string, length = 80
facility_data%total_facility_parameters	Total Number of Facility Parameters	scalar integer
facility_data%facility_parameter_001	Test-specific Facility Parameter	string structure
.		
.		
facility_data%facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data%model_id	Model Identification Code	scalar integer
model_data%model_description	Description of Test Model	string, length = 80
model_data%total_model_parameters	Total Number of Model Parameters	scalar integer
model_data%model_parameter_001	Test-specific Model Parameter	string structure
.		
.		
model_data%model_parameter_200	Test-specific Model Parameter	string structure
CSM variables		
csm_data%fft_length	FFT Block Length	scalar integer
csm_data%overlap	FFT Block Overlap	scalar integer
csm_data%num_blocks	Total Number of FFT Blocks	scalar integer
csm_data%num_blocks_used	Total Number of FFT Blocks Actually Used for Form CSM	scalar integer
csm_data%block_table	Block Usage Table for CSM Generation	2D integer array, dimension(num_blocks,num_channels)
csm_data%cslength	Saved Cross Spectral Length	scalar integer
csm_data%winfunc	FFT Window Function Name	string, length = 12
csm_data%beta	Kaiser Window Function Beta Value	scalar real
csm_data%CSM_units	Units for Cross Spectral Matrix Elements	scalar integer
csm_data%csm_real	Cross Spectral Matrix Real Data Matrix	3D real array, dimension(channels,channels,bins)
csm_data%csm_imag	Cross Spectral Matrix Imaginary Data Matrix	3D real array, dimension(channels,channels,bins)

Appendix B.4 - Fortran Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (concluded)

Variable Name	*Description*	*Data Type*
The string structures for the facility and model parameters are as follows:		
facility_data%facility_parameter(*n*)%long_name	Long name of parameter	string, length = 80
facility_data%facility_parameter(*n*)%description	Detailed description of parameter	string, length = 80
facility_data%facility_parameter(*n*)%data_type	Parameter data type	string, length = 80
facility_data%facility_parameter(*n*)%units	Parameter units	string, length = 80
facility_data%facility_parameter(*n*)%value	Parameter value	string, length = 80
model_data%model_parameter(*n*)%long_name	Long name of parameter	string, length = 80
model_data%model_parameter(*n*)%description	Detailed description of parameter	string, length = 80
model_data%model_parameter(*n*)%data_type	Parameter data type	string, length = 80
model_data%model_parameter(*n*)%units	Parameter units	string, length = 80
model_data%model_parameter(*n*)%value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 1).

Appendix C.1 - Matlab Data Structure for NetCDF / HDF5 Individual Raw Data Files

Variable Name	*Description*	*Data Type*
Version variables		
version_data.netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar single
version_data.hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar single
version_data.hdr_version	Header Version Number	scalar single
Test variables		
test_data.testno	Test Number	scalar int32
test_data.runno	Run Number	scalar int32
test_data.pointno	Point Number	scalar int32
test_data.date_time	Acquistion Date and Time	string, length = 17
test_data.GPS_start_time	Acquisition UTC Start Time	scalar single
test_data.GPS_stop_time	Acquisition UTC Stop Time	scalar single
test_data.test_engineer	Test Engineer Name	string, length = 80
Channel variables		
channel_data.channo	Channel Number	scalar int32
channel_data.nam_chan	Channel Name	string, length = 80
channel_data.chan_description	Channel Description	string, length = 80
channel_data.sensor_sn	Sensor Serial Number	scalar int32
channel_data.preamplifier_sn	Preamplifier Serial Number	scalar int32
channel_data.sensor_type	Sensor Type	scalar int32
channel_data.preamplifier_type	Preamplifier Type	scalar int32
channel_data.sensor_description	Sensor Description	string, length = 80
channel_data.preamplifier_description	Preamplifier Description	string, length = 80
channel_data.excitation_voltage	Excitation Voltage	scalar single
channel_data.excitation_current	Excitation Current	scalar single
channel_data.sensor_x	Sensor x Location	scalar single
channel_data.sensor_y	Sensor y Location	scalar single
channel_data.sensor_z	Sensor z Location	scalar single
channel_data.sensor_coord_ref	Sensor Coordinate Reference	string, length = 12

Appendix C.1 - Matlab Data Structure for NetCDF / HDF5 Individual Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Calibration variables		
calibration_data.sensitivity	Sensor Sensitivity from Calibration	scalar single
calibration_data.number_of_cal_sources	Number of On-Board Calibration Sources Used	scalar int32
calibration_data.cal_source_coordinates	X,Y,Z Coordinates of Calibration Sources	2D array of singles, dimension(num_sources,3)
calibration_data.cal_source_coord_ref	Coordinate System for Calibration Source Locations	string, length = 12
calibration_data.cal_source_baseline_array	On-board Calibration Source Baseline Level Array	1D array of singles, dimension(num_sources)
calibration_data.cal_source_level_array	On-board Calibration Source Level Array	1D array of singles, dimension(num_sources)
calibration_data.injection_cal_baseline	Injection Calibration Baseline Level	scalar single
calibration_data.injection_cal_level	Injection Calibration Recorded Level	scalar single
calibration_data.calibration_frequency	Sensor Frequency Response Frequency Array	1D array of singles, dimension(num_bins)
calibration_data.calibration_magnitude	Sensor Frequency Response Magnitude Array	1D array of singles, dimension(num_bins)
calibration_data.calibration_phase	Sensor Frequency Response Phase Array	1D array of singles, dimension(num_bins)
DAS variables		
das_data.total_channels	Total Number of Channels in System	scalar int32
das_data.adbits	A/D Resolution for Channel	scalar int32
das_data.coupling	AC Coupling Indicator	scalar int32
das_data.full_scale_range	Channel Full Scale Range	scalar single
das_data.trigger_mode	Trigger Mode for Channel	scalar int32
das_data.clock_mode	Clock Mode for Channel	scalar int32
das_data.sample_count	Channel Sample Count	scalar int32
das_data.sample_period	Channel Sample Period	scalar single
das_data.lpf_cutoff_freq	Low Pass Filter Cutoff Frequency	scalar single
das_data.lpf_pregain	Low Pass Filter Pregain Setting	scalar single
das_data.lpf_postgain	Low Pass Filter Postgain Setting	scalar single
das_data.hpf_cutoff_freq	High Pass Filter Cutoff Frequency	scalar single
das_data.hpf_pregain	High Pass Filter Pregain Setting	scalar single
das_data.hpf_postgain	High Pass Filter Postgain Setting	scalar single
das_data.external_gain_offset	Sensor Power Supply External Gain Offset	scalar single
das_data.on_board_LP_filter	On-board LP Filter Boolean Switch	scalar int32

Appendix C.1 - Matlab Data Structure for NetCDF / HDF5 Individual Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Array variables		
array_data.total_array_sensors	Total Number of Array Sensors	scalar int32
array_data.traverse_configuration	Traverse Configuration Description	string, length = 80
array_data.sideline_traverse_x	Sideline Traverse X Location	scalar single
array_data.sideline_traverse_y	Sideline Traverse Y Location	scalar single
array_data.sideline_traverse_z	Sideline Traverse Z Location	scalar single
array_data.sideline_traverse_coord_ref	Coordinate System for Sideline Traverse	string, length = 12
array_data.array_traverse_x	Array Traverse X Location	scalar single
array_data.array_traverse_y	Array Traverse Y Location	scalar single
array_data.array_traverse_z	Array Traverse Z Location	scalar single
array_data.array_traverse_coord_ref	Coordinate System for Array Traverse	string, length = 12
array_data.array_elevation	Array Face Elevation Angle	scalar single
array_data.array_azimuth	Array Face Azimuth Angle	scalar single
array_data.number_of_inclinometers	Number of Inclinometers Incorporated Into Array	scalar int32
array_data.inclinometer_coordinates	X,Y,Z Coordinates of Inclinometers	2D array of singles, dimension(num_inclinometers,3)
array_data.inclinometer_coord_ref	Coordinate System for Inclinometer Locations	string, length =12
array_data.inclinometer_phi_array	Array Inclinometer Phi Reading Array	1D array of singles, dimension(num_inclinometers)
array_data.inclinometer_theta_array	Array Inclinometer Theta Reading Array	1D array of singles, dimension(num_inclinometers)
array_data.inclinometer_rotation_array	Array Inclinometer Rotation Reading Array	1D array of singles, dimension(num_inclinometers)
array_data.inclinometer_temperature_array	Array Inclinometer Temperature Reading Array	1D array of singles, dimension(num_inclinometers)
array_data.photogrammetric_x_location	Array Panel X Location from Photogrammetry Measurements	scalar single
array_data.photogrammetric_y_location	Array Panel Y Location from Photogrammetry Measurements	scalar single
array_data.photogrammetric_z_location	Array Panel Z Location from Photogrammetry Measurements	scalar single
array_data.photogrammetric_yaw_angle	Array Panel Yaw Angle from Photogrammetry Measurements	scalar single
array_data.photogrammetric_pitch_angle	Array Panel Pitch Angle from Photogrammetry Measurements	scalar single
array_data.photogrammetric_roll_angle	Array Panel Roll Angle from Photogrammetry Measurements	scalar single
array_data.photogrammetric_number_targets	Array Panel Number of Photogrammetry Targets Used	scalar int32
array_data.photogrammetric_RMS_error	Array Panel Photogrammetry RMS Error	scalar single

Appendix C.1 - Matlab Data Structure for NetCDF / HDF5 Individual Raw Data Files (concluded)

Variable Name	*Description*	*Data Type*
Facility variables		
facility_data.facility_name	Name of Ground Test Facility	string, length = 80
facility_data.facility_description	Description of Ground Test Facility	string, length = 80
facility_data.total_facility_parameters	Total Number of Facility Parameters	scalar int32
facility_data.facility_parameter_001	Test-specific Facility Parameter	string structure
.		
.		
facility_data.facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data.model_id	Model Identification Code	scalar int32
model_data.model_description	Description of Test Model	string, length = 80
model_data.total_model_parameters	Total Number of Model Parameters	scalar int32
model_data.model_parameter_001	Test-specific Model Parameter	string structure
.		
.		
model_data.model_parameter_200	Test-specific Model Parameter	string structure
Raw Data		
raw_data.data_values	Raw Data Values	1D array of singles, dimension(num_samples)

The string structures for the facility and model parameters are as follows:

facility_data.facility_parameter(*n*).long_name	Long name of parameter	string, length = 80
facility_data.facility_parameter(*n*).description	Detailed description of parameter	string, length = 80
facility_data.facility_parameter(*n*).data_type	Parameter data type	string, length = 80
facility_data.facility_parameter(*n*).units	Parameter units	string, length = 80
facility_data.facility_parameter(*n*).value	Parameter value	string, length = 80
model_data.model_parameter(*n*).long_name	Long name of parameter	string, length = 80
model_data.model_parameter(*n*).description	Detailed description of parameter	string, length = 80
model_data.model_parameter(*n*).data_type	Parameter data type	string, length = 80
model_data.model_parameter(*n*).units	Parameter units	string, length = 80
model_data.model_parameter(*n*).value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 1).

Appendix C.2 - Matlab Data Structure for NetCDF / HDF5 Ensemble Raw Data Files

Variable Name	*Description*	*Data Type*
Version variables		
ensemble_version_data.netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar single
ensemble_version_data.hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar single
ensemble_version_data.hdr_version	Header Version Number	scalar single
Test variables		
ensemble_test_data.testno	Test Number	scalar int32
ensemble_test_data.runno	Run Number	scalar int32
ensemble_test_data.pointno	Point Number	scalar int32
ensemble_test_data.identifier_prefix	Acquisition Type Prefix Character	character
ensemble_test_data.date_time	Acquistion Date and Time	string, length = 17
ensemble_test_data.GPS_start_time	Acquisition UTC Start Time	scalar single
ensemble_test_data.GPS_stop_time	Acquisition UTC Stop Time	scalar single
ensemble_test_data.test_engineer	Test Engineer Name	scalar string
Channel variables		
ensemble_channel_data.channo_array	Channel Number Array	1D array of int32, dimension(num_channels)
ensemble_channel_data.sensor_sn_array	Sensor Serial Number Array	1D array of int32, dimension(num_channels)
ensemble_channel_data.preamplifier_sn_array	Preamplifier Serial Number Array	1D array of int32, dimension(num_channels)
ensemble_channel_data.sensor_type_array	Sensor Type Array	1D array of int32, dimension(num_channels)
ensemble_channel_data.preamplifier_type_array	Preamplifier Type Array	1D array of int32, dimension(num_channels)
ensemble_channel_data.excitation_voltage_array	Excitation Voltage Array	1D array of single, dimension(num_channels)
ensemble_channel_data.excitation_current_array	Excitation Current Array	1D array of single, dimension(num_channels)
ensemble_channel_data.sensor_coord_array	Array of Sensor X,Y,Z Locations	2D array of singles, dimension(num_channels,3)
ensemble_channel_data.sensor_coord_ref_array	Array Sensor Coordinate References	1D array of strings, dimension(num_channels)

Appendix C.2 - Matlab Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Calibration variables		
ensemble_calibration_data.sensitivity_array	Sensor Sensitivity from Calibration	1D array of singles, dimension(num_channels)
ensemble_calibration_data.filter_response	Signal Conditioning Bandpass Filter Response	2D array of singles, dimension(num_bins,3)
ensemble_calibration_data.number_of_cal_sources	Number of On-Board Calibration Sources Used	scalar int32
ensemble_calibration_data.cal_source_coordinates	X,Y,Z Coordinates of Calibration Sources	2D array of singles, dimension(num_sources,3)
ensemble_calibration_data.cal_source_coord_ref_array	Coordinate System for Calibration Source Locations	1D array of strings, length = 12, dimension(num_sources)
ensemble_calibration_data.number_of_cal_measurements	Number of calibration measurements completed	scalar int32
ensemble_calibration_data.date_time_of_cal	Array of source calibration dates and times	2D array of strings, length = 17, dimension(num_sources,num_cals)
ensemble_calibration_data.cal_source_baseline_array	On-board Calibration Source Baseline Level Array	3D array of singles, dimension(channels,sources,cals)
ensemble_calibration_data.cal_source_level_array	On-board Calibration Source Level Array	3D array of singles, dimension(channels,sources,cals)
ensemble_calibration_data.injection_cal_baseline_array	Injection Calibration Baseline Level	1D array of singles, dimension(num_channels)
ensemble_calibration_data.injection_cal_level_array	Injection Calibration Recorded Level	1D array of singles, dimension(num_channels)
ensemble_calibration_data.freq_calibration_array	Sensor Frequency Response Frequency Array	3D array of singles, dimension(num_channels,num_bins,3)

Appendix C.2 - Matlab Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
DAS variables		
ensemble_DAS_data.total_channels	Total Number of Channels in System	scalar int32
ensemble_DAS_data.adbits	A/D Resolution for Channel	scalar int32
ensemble_DAS_data.coupling_array	AC Coupling Indicator Array	1D array of int32, dimension(num_channels)
ensemble_DAS_data.full_scale_range_array	Channel Full Scale Range Array	1D array of singles, dimension(num_channels)
ensemble_DAS_data.trigger_mode	Trigger Mode for Channel	scalar int32
ensemble_DAS_data.clock_mode	Clock Mode for Channel	scalar int32
ensemble_DAS_data.sample_count	Channel Sample Count	scalar int32
ensemble_DAS_data.sample_period	Channel Sample Period	scalar single
ensemble_DAS_data.lpf_cutoff_freq_array	Low Pass Filter Cutoff Frequency Array	1D array of singles, dimension(num_channels)
ensemble_DAS_data.lpf_pregain_array	Low Pass Filter Pregain Setting Array	1D array of singles, dimension(num_channels)
ensemble_DAS_data.lpf_postgain_array	Low Pass Filter Postgain Setting Array	1D array of singles, dimension(num_channels)
ensemble_DAS_data.hpf_cutoff_freq_array	High Pass Filter Cutoff Frequency Array	1D array of singles, dimension(num_channels)
ensemble_DAS_data.hpf_pregain_array	High Pass Filter Pregain Setting Array	1D array of singles, dimension(num_channels)
ensemble_DAS_data.hpf_postgain_array	High Pass Filter Postgain Setting Array	1D array of singles, dimension(num_channels)
ensemble_DAS_data.external_gain_offset_array	Sensor Power Supply External Gain Offset Array	1D array of singles, dimension(num_channels)
ensemble_DAS_data.on_board_LP_filter_array	On-board LP Filter Boolean Switch Array	1D array of int32, dimension(num_channels)

Appendix C.2 - Matlab Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (continued)

Variable Name	*Description*	*Data Type*
Array variables		
ensemble_array_data.total_array_sensors	Total Number of Array Sensors	scalar int32
ensemble_array_data.traverse_configuration	Traverse Configuration Description	string, length = 80
ensemble_array_data.sideline_traverse_x	Sideline Traverse X Location	scalar single
ensemble_array_data.sideline_traverse_y	Sideline Traverse Y Location	scalar single
ensemble_array_data.sideline_traverse_z	Sideline Traverse Z Location	scalar single
ensemble_array_data.sideline_traverse_coord_ref	Coordinate System for Sideline Traverse	string, length = 12
ensemble_array_data.array_traverse_x	Array Traverse X Location	scalar single
ensemble_array_data.array_traverse_y	Array Traverse Y Location	scalar single
ensemble_array_data.array_traverse_z	Array Traverse Z Location	scalar single
ensemble_array_data.array_traverse_coord_ref	Coordinate System for Array Traverse	string, length = 12
ensemble_array_data.array_elevation	Array Face Elevation Angle	scalar single
ensemble_array_data.array_azimuth	Array Face Azimuth Angle	scalar single
ensemble_array_data.number_of_inclinometers	Number of Inclinometers Incorporated Into Array	scalar int32
ensemble_array_data.inclinometer_coordinates	X,Y,Z Coordinates of Inclinometers	2D array of singles, dimension(num_inclinometers,3)
ensemble_array_data.inclinometer_coord_ref	Coordinate System for Inclinometer Locations	string, length = 12
ensemble_array_data.inclinometer_phi_array	Array Inclinometer Phi Reading Array	1D array of singles, dimension(num_inclinometers)
ensemble_array_data.inclinometer_theta_array	Array Inclinometer Theta Reading Array	1D array of singles, dimension(num_inclinometers)
ensemble_array_data.inclinometer_rotation_array	Array Inclinometer Rotation Reading Array	1D array of singles, dimension(num_inclinometers)
ensemble_array_data.inclinometer_temperature_array	Array Inclinometer Temperature Reading Array	1D array of singles, dimension(num_inclinometers)
ensemble_array_data.photogrammetric_x_location	Array Panel X Location from Photogrammetry Measurements	scalar single
ensemble_array_data.photogrammetric_y_location	Array Panel Y Location from Photogrammetry Measurements	scalar single
ensemble_array_data.photogrammetric_z_location	Array Panel Z Location from Photogrammetry Measurements	scalar single
ensemble_array_data.photogrammetric_yaw_angle	Array Panel Yaw Angle from Photogrammetry Measurements	scalar single
ensemble_array_data.photogrammetric_pitch_angle	Array Panel Pitch Angle from Photogrammetry Measurements	scalar single
ensemble_array_data.photogrammetric_roll_angle	Array Panel Roll Angle from Photogrammetry Measurements	scalar single
ensemble_array_data.photogrammetric_number_targets	Array Panel Number of Photogrammetry Targets Used	scalar int32
ensemble_array_data.photogrammetric_RMS_error	Array Panel Photogrammetry RMS Error	scalar single

Appendix C.2 - Matlab Data Structure for NetCDF / HDF5 Ensemble Raw Data Files (concluded)

Variable Name	*Description*	*Data Type*
Facility variables		
facility_data.facility_name	Name of Ground Test Facility	string, length = 80
facility_data.facility_description	Description of Ground Test Facility	string, length = 80
facility_data.total_facility_parameters	Total Number of Facility Parameters	scalar int32
facility_data.facility_parameter_001	Test-specific Facility Parameter	string structure
.		
.		
facility_data.facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data.model_id	Model Identification Code	scalar int32
model_data.model_description	Description of Test Model	string, length = 80
model_data.total_model_parameters	Total Number of Model Parameters	scalar int32
model_data.model_parameter_001	Test-specific Model Parameter	string structure
.		
.		
model_data.model_parameter_200	Test-specific Model Parameter	string structure
Raw Data		
ensemble_raw_data.data_values	Raw Data Values	2D array of singles, dimension(num_channels,num_samples)

The string structures for the facility and model parameters are as follows:

facility_data.facility_parameter(*n*).long_name	Long name of parameter	string, length = 80
facility_data.facility_parameter(*n*).description	Detailed description of parameter	string, length = 80
facility_data.facility_parameter(*n*).data_type	Parameter data type	string, length = 80
facility_data.facility_parameter(*n*).units	Parameter units	string, length = 80
facility_data.facility_parameter(*n*).value	Parameter value	string, length = 80
model_data.model_parameter(*n*).long_name	Long name of parameter	string, length = 80
model_data.model_parameter(*n*).description	Detailed description of parameter	string, length = 80
model_data.model_parameter(*n*).data_type	Parameter data type	string, length = 80
model_data.model_parameter(*n*).units	Parameter units	string, length = 80
model_data.model_parameter(*n*).value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 1).

Appendix C.3 – Matlab Data Structure for NetCDF / HDF5 Static Pressure Data Files

Variable Name	*Description*	*Data Type*
Version variables		
version_data.netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar float
version_data.hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar float
version_data.hdr_version	Header Version Number	scalar float
Test variables		
test_data.testno	Test Number	scalar long
test_data.runno	Run Number	scalar long
test_data.pointno	Point Number	scalar long
test_data.date_time	Acquistion Date and Time	string, length 17
test_data.GPS_start_time	Acquisition UTC Start Time	scalar float
test_data.GPS_stop_time	Acquisition UTC Stop Time	scalar float
test_data.test_engineer	Test Engineer Name	string, length 80
Facility variables		
facility_data.facility_name	Name of Ground Test Facility	string, length 80
facility_data.facility_description	Description of Ground Test Facility	string, length 80
facility_data.total_facility_parameters	Total Number of Facility Parameters	scalar long
facility_data.facility_parameter_001	Test-specific Facility Parameter	string structure
.		
.		
facility_data.facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data.model_id	Model Identification Code	scalar long
model_data.model_description	Description of Test Model	string, length 80
model_data.total_model_parameters	Total Number of Model Parameters	scalar long
model_data.model_parameter_001	Test-specific Model Parameter	string structure
.		
.		
model_data.model_parameter_200	Test-specific Model Parameter	string structure

Appendix C.3 – Matlab Data Structure for NetCDF / HDF5 Static Pressure Data Files (concluded)

Variable Name	*Description*	*Data Type*
Static tap variables		
static_data.total_static_taps	Number of Static Taps in File	scalar long
static_data.static_tap_parameter_001	Static Tap Data Entry	string structure
.		
.		
static_data.static_tap_parameter_500	Static Tap Data Entry	string structure

The string structures for the facility and model parameters are as follows:

facility_data.facility_parameter(*n*).long_name	Long name of parameter	string, length = 80
facility_data.facility_parameter(*n*).description	Detailed description of parameter	string, length = 80
facility_data.facility_parameter(*n*).data_type	Parameter data type	string, length = 80
facility_data.facility_parameter(*n*).units	Parameter units	string, length = 80
facility_data.facility_parameter(*n*).value	Parameter value	string, length = 80
model_data.model_parameter(*n*).long_name	Long name of parameter	string, length = 80
model_data.model_parameter(*n*).description	Detailed description of parameter	string, length = 80
model_data.model_parameter(*n*).data_type	Parameter data type	string, length = 80
model_data.model_parameter(*n*).units	Parameter units	string, length = 80
model_data.model_parameter(*n*).value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 1).

The string structure for the static tap data is as follows:

static_data.static_tap_parameter(*n*).static_tap_designation	Static tap designation	string, length = 80
static_data.static_tap_parameter(*n*).description	Static tap description	string, length = 80
static_data.static_tap_parameter(*n*).units	Static tap pressure value units	string, length = 80
static_data.static_tap_parameter(*n*).coefficient_name	Static tap coefficient name	string, length = 80
static_data.static_tap_parameter(*n*).data_value	Static tap pressure value	string, length = 80

where *n* is the static tap number (*n* starts at 1).

Appendix C.4 - Matlab Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files

Variable Name	*Description*	*Data Type*
Version variables		
csm_version_data->netcdf_version	NetCDF Library Version Number (NetCDF files)	scalar single
csm_version_data->hdf5_version	HDF5 Library Version Number (HDF5 files)	scalar single
csm_version_data.hdr_version	Header Version Number	scalar single
Test variables		
csm_test_data.testno	Test Number	scalar int32
csm_test_data.runno	Run Number	scalar int32
csm_test_data.pointno	Point Number	scalar int32
csm_test_data.date_time	Acquistion Date and Time	string, length = 17
csm_test_data.GPS_start_time	Acquisition UTC Start Time	scalar single
csm_test_data.GPS_stop_time	Acquisition UTC Stop Time	scalar single
Channel variables		
csm_channel_data.sensor_sn_array	Sensor Serial Number Array	1D array of int32, dimension(num_channels)
csm_channel_data.preamplifier_sn_array	Preamplifier Serial Number Array	1D array of int32, dimension(num_channels)
csm_channel_data.sensor_type_array	Sensor Type Array	1D array of int32, dimension(num_channels)
csm_channel_data.preamplifier_type_array	Preamplifier Type Array	1D array of int32, dimension(num_channels)
csm_channel_data.excitation_voltage_array	Excitation Voltage Array	1D array of singles, dimension(num_channels)
csm_channel_data.excitation_current_array	Excitation Current Array	1D array of singles, dimension(num_channels)
csm_channel_data.sensor_coord_array	Array of Sensor X,Y,Z Locations	2D array of singles, dimension(num_channels,3)
csm_channel_data.array_sensor_coord_ref	Array Sensor Coordinate Reference	string, length = 12
csm_channel_data.sideline_sensor_coord_ref	Sideline Sensor Coordinate Reference	string, length = 12
csm_channel_data.kulite_sensor_coord_ref	Kulite Sensor Coordinate Reference	string, length = 12

Appendix C.4 - Matlab Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
Calibration variables		
csm_calibration_data.sensitivity_array	Sensor Sensitivity from Calibration	1D array of singles, dimension(num_channels)
csm_calibration_data.freq_calibration_array	Sensor Frequency Response Frequency Array	3D array of singles, dimension(num_channels,num_bins,3)
csm_calibration_data.filter_response	Signal Conditioning Bandpass Filter Response	2D array of singles, dimension(num_bins,3)
csm_calibration_data.number_of_cal_sources	Number of On-Board Calibration Sources Used	scalar int32
csm_calibration_data.cal_source_coordinates	X,Y,Z Coordinates of Calibration Sources	2D array of singles, dimension(num_sources,3)
csm_calibration_data.cal_source_coord_ref_array	Array of Coordinate Systems for Calibration Source Locations	1D array of strings, length 12, dimension(num_sources)
csm_calibration_data.number_of_cal_measurements	Number of On-Board Calibration Runs Performed	scalar int32
csm_calibration_data.date_time_of_cal	Array of Dates and Times On-Board Calibration Runs Performed	2D array of strings, length 17, dimension(num_sources,num_cals)
csm_calibration_data.cal_source_baseline_array	On-board Calibration Source Baseline Level Array	3D array of singles, dimension(channels,sources,cals)
csm_calibration_data.cal_source_level_array	On-board Calibration Source Level Array	3D array of singles, dimension(channels,sources,cals)
csm_calibration_data.injection_cal_baseline_array	Injection Calibration Baseline Level	1D array of singles, dimension(num_channels)
csm_calibration_data.injection_cal_level_array	Injection Calibration Recorded Level	1D array of singles, dimension(num_channels)
csm_calibration_data.sensitivity_correction_switch	Indicator for Sensitivity Adjustment Due to Source Calibration	scalar int32
csm_calibration_data.temperature_correction_switch	Indicator for Sensitivity Adjustment Due to Temperature	scalar int32
csm_calibration_data.6db_correction_switch	Indicator for 6 dB Subtraction from Spectra for Sensor Flush Mounting	scalar int32

Appendix C.4 - Matlab Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
DAS variables		
csm_das_data.total_channels	Total Number of Channels in System	scalar int32
csm_das_data.sample_count	Channel Sample Count	scalar int32
csm_das_data.sample_period	Channel Sample Period	scalar single
csm_das_data.full_scale_range_array	Channel Full Scale Range Array	1D array of singles, dimension(num_channels)
csm_das_data.lpf_cutoff_freq_array	Low Pass Filter Cutoff Frequency Array	1D array of singles, dimension(num_channels)
csm_das_data.lpf_pregain_array	Low Pass Filter Pregain Setting Array	1D array of singles, dimension(num_channels)
csm_das_data.lpf_postgain_array	Low Pass Filter Postgain Setting Array	1D array of singles, dimension(num_channels)
csm_das_data.hpf_cutoff_freq_array	High Pass Filter Cutoff Frequency Array	1D array of singles, dimension(num_channels)
csm_das_data.hpf_pregain_array	High Pass Filter Pregain Setting Array	1D array of singles, dimension(num_channels)
csm_das_data.hpf_postgain_array	High Pass Filter Postgain Setting Array	1D array of singles, dimension(num_channels)
csm_das_data.external_gain_offset_array	Sensor Power Supply External Gain Offset Array	1D array of singles, dimension(num_channels)

Appendix C.4 - Matlab Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
Array variables		
csm_array_data.total_array_sensors	Total Number of Array Sensors	scalar int32
csm_array_data.traverse_configuration	Traverse Configuration Description	string, length = 80
csm_array_data.sideline_traverse_x	Sideline Traverse X Location	scalar single
csm_array_data.sideline_traverse_y	Sideline Traverse Y Location	scalar single
csm_array_data.sideline_traverse_z	Sideline Traverse Z Location	scalar single
csm_array_data.sideline_traverse_coord_ref	Coordinate System for Sideline Traverse	string, length = 12
csm_array_data.array_traverse_x	Array Traverse X Location	scalar single
csm_array_data.array_traverse_y	Array Traverse Y Location	scalar single
csm_array_data.array_traverse_z	Array Traverse Z Location	scalar single
csm_array_data.array_traverse_coord_ref	Coordinate System for Array Traverse	string, length = 12
csm_array_data.array_elevation	Array Face Elevation Angle	scalar single
csm_array_data.array_azimuth	Array Face Azimuth Angle	scalar single
csm_array_data.number_of_inclinometers	Number of Inclinometers Incorporated Into Array	scalar int32
csm_array_data.inclinometer_coordinates	X,Y,Z Coordinates of Inclinometers	2D array of singles, dimension(num_inclinometers,3)
csm_array_data.inclinometer_coord_ref	Coordinate System for Inclinometer Locations	string, length = 12
csm_array_data.inclinometer_phi_array	Array Inclinometer Phi Reading Array	1D array of singles, dimension(num_inclinometers)
csm_array_data.inclinometer_theta_array	Array Inclinometer Theta Reading Array	1D array of singles, dimension(num_inclinometers)
csm_array_data.inclinometer_rotation_array	Array Inclinometer Rotation Reading Array	1D array of singles, dimension(num_inclinometers)
csm_array_data.inclinometer_temperature_array	Array Inclinometer Temperature Reading Array	1D array of singles, dimension(num_inclinometers)
csm_array_data.photogrammetric_x_location	Array Panel X Location from Photogrammetry Measurements	scalar single
csm_array_data.photogrammetric_y_location	Array Panel Y Location from Photogrammetry Measurements	scalar single
csm_array_data.photogrammetric_z_location	Array Panel Z Location from Photogrammetry Measurements	scalar single
csm_array_data.photogrammetric_yaw_angle	Array Panel Yaw Angle from Photogrammetry Measurements	scalar single
csm_array_data.photogrammetric_pitch_angle	Array Panel Pitch Angle from Photogrammetry Measurements	scalar single
csm_array_data.photogrammetric_roll_angle	Array Panel Roll Angle from Photogrammetry Measurements	scalar single
csm_array_data.photogrammetric_number_targets	Array Panel Number of Photogrammetry Targets Used	scalar int32
csm_array_data.photogrammetric_RMS_error	Array Panel Photogrammetry RMS Error	scalar single

Appendix C.4 - Matlab Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (continued)

Variable Name	*Description*	*Data Type*
Facility variables		
facility_data.facility_name	Name of Ground Test Facility	string, length = 80
facility_data.facility_description	Description of Ground Test Facility	string, length = 80
facility_data.total_facility_parameters	Total Number of Facility Parameters	scalar int32
facility_data.facility_parameter_001	Test-specific Facility Parameter	string structure (see below)
.		
.		
facility_data.facility_parameter_200	Test-specific Facility Parameter	string structure
Model variables		
model_data.model_id	Model Identification Code	scalar int32
model_data.model_description	Description of Test Model	string, length = 80
model_data.total_model_parameters	Total Number of Model Parameters	scalar int32
model_data.model_parameter_001	Test-specific Model Parameter	string structure (see below)
.		
.		
model_data.model_parameter_200	Test-specific Model Parameter	string structure
CSM variables		
csm_data.fft_length	FFT Block Length	scalar int32
csm_data.overlap	FFT Block Overlap	scalar int32
csm_data.num_blocks	Total Number of FFT Blocks	scalar int32
csm_data.num_blocks_used	Total Number of FFT Blocks Actually Used for Form CSM	scalar int32
csm_data.block_table	Block Usage Table for CSM Generation	2D array of int32, dimension(num_blocks,num_channels)
csm_data.cslength	Saved Cross Spectral Length	scalar int32
csm_data.winfunc	FFT Window Function Name	string, length = 11
csm_data.beta	Kaiser Window Function Beta Value	scalar single
csm_data.CSM_units	Units for Cross Spectral Matrix Elements	scalar int32
csm_data.csm_real	Cross Spectral Matrix Real Data Matrix	3D array of singles, dimension(channels,channels,bins)
csm_data.csm_imag	Cross Spectral Matrix Imaginary Data Matrix	3D array of singles, dimension(channels,channels,bins)

Appendix C.4 - Matlab Data Structure for NetCDF / HDF5 Cross Spectral Matrix Files (concluded)

Variable Name	*Description*	*Data Type*
The string structures for the facility and model parameters are as follows:		
facility_data.facility_parameter(*n*).long_name	Long name of parameter	string, length = 80
facility_data.facility_parameter(*n*).description	Detailed description of parameter	string, length = 80
facility_data.facility_parameter(*n*).data_type	Parameter data type	string, length = 80
facility_data.facility_parameter(*n*).units	Parameter units	string, length = 80
facility_data.facility_parameter(*n*).value	Parameter value	string, length = 80
model_data.model_parameter(*n*).long_name	Long name of parameter	string, length = 80
model_data.model_parameter(*n*).description	Detailed description of parameter	string, length = 80
model_data.model_parameter(*n*).data_type	Parameter data type	string, length = 80
model_data.model_parameter(*n*).units	Parameter units	string, length = 80
model_data.model_parameter(*n*).value	Parameter value	string, length = 80

where *n* is the parameter number (*n* starts at 1).

Appendix D.1 - Populating Header Values in Individual Raw Data Files
C Code Fragment (assumes arrow access to structure)

```
// Version data

version_data->netcdf_version = 3.4;           //If writing NetCDF files
version_data->hdf5_version   = 1.8;           //If writing HDF5 files
version_data->hdr_version    = 9.4;           //Note: Do not change these values

// Test data

test_data->testno          = 643;
test_data->runno           = 259;
test_data->pointno         = 3541;
test_data->GPS_start_time  = 28800.0;
test_data->GPS_stop_time   = 28830.0;
strcpy_and_pad(test_data->date_time,"12/20/2018 08:00",17);
strcpy_and_pad(test_data->test_engineer,"William Humphreys",80);

// Channel data

channel_data->channo                    = 1;
channel_data->sensor_sn                 = 2688098;
channel_data->preamplifier_sn           = 2688519;
channel_data->sensor_type               = 1;
channel_data->preamplifier_type         = 1;
channel_data->excitation_voltage        = 200.0;
channel_data->excitation_current        = 0.0;
channel_data->sensor_x                  = -0.041;
channel_data->sensor_y                  = 0.999;
channel_data->sensor_z                  = 0.0;
strcpy_and_pad(channel_data->sensor_coord_ref,"ARRAY",80);
strcpy_and_pad(channel_data->nam_chan,"Channel 001",80);
strcpy_and_pad(channel_data->chan_description,"Array Channel",80);
strcpy_and_pad(channel_data->sensor_description,"B&K 4938 Pressure Microphone",80);
strcpy_and_pad(channel_data->preamplifier_description,
                                        "B&K 2670 1/4-inch Preamplifier",80);

// Calibration data

calibration_data->sensitivity                   = 63.0;
calibration_data->number_of_cal_sources         = 3;
calibration_data->cal_source_coordinates[0][0] = 114.34;
calibration_data->cal_source_coordinates[0][1] = 22.85;
calibration_data->cal_source_coordinates[0][2] = -4.79;
calibration_data->cal_source_coordinates[1][0] = 137.56;
calibration_data->cal_source_coordinates[1][1] = 58.59;
calibration_data->cal_source_coordinates[1][2] = 0.99;
calibration_data->cal_source_coordinates[2][0] = 164.55;
calibration_data->cal_source_coordinates[2][1] = 99.85;
calibration_data->cal_source_coordinates[2][2] = 5.07;
calibration_data->cal_source_baseline_array[0] = 0.0;
calibration_data->cal_source_baseline_array[1] = 0.0;
calibration_data->cal_source_baseline_array[2] = 0.0;
calibration_data->cal_source_level_array[0]    = 0.0;
calibration_data->cal_source_level_array[1]    = 0.0;
calibration_data->cal_source_level_array[2]    = 0.0;
calibration_data->injection_cal_baseline       = 1.0;
calibration_data->injection_cal_level          = 1.0;
strcpy_and_pad(calibration_data->cal_source_coord_ref,"MODEL",80);
for (i = 0; i < number_of_bins; i++){
   calibration_data->calibration_frequency[i]  = (float)(i-1)*frequency_resolution;
   calibration_data->calibration_magnitude[i]  = 1.0;
   calibration_data->calibration_phase[i]      = 0.0;
}
```

```
// DAS data

DAS_data->total_channels         = 97;
DAS_data->adbits                 = 24;
DAS_data->coupling               = 1;
DAS_data->full_scale_range       = 10000.0;
DAS_data->trigger_mode           = 0;
DAS_data->clock_mode             = 0;
DAS_data->sample_count           = 3000000;
DAS_data->sample_period          = 0.005;
DAS_data->lpf_cutoff_freq        = 102300.0;
DAS_data->lpf_pregain            = 1.0;
DAS_data->lpf_postgain           = 1.0;
DAS_data->hpf_cutoff_freq        = 400.0;
DAS_data->hpf_pregain            = 1.0;
DAS_data->hpf_postgain           = 1.0;
DAS_data->external_gain_offset   = 20.0;
DAS_data->on_board_LP_filter     = 0;

// Array data

array_data->total_array_sensors              = 97;
array_data->sideline_traverse_x              = 0.0;
array_data->sideline_traverse_y              = 0.0;
array_data->sideline_traverse_z              = 0.0;
array_data->array_traverse_x                 = 0.0;
array_data->array_traverse_y                 = 0.0;
array_data->array_traverse_z                 = 0.0;
array_data->array_elevation                  = 0.0;
array_data->array_azimuth                    = 0.0;
array_data->number_of_inclinometers          = 3;
array_data->inclinometer_coordinates[0][0]   = 10.0;
array_data->inclinometer_coordinates[0][1]   = 20.0;
array_data->inclinometer_coordinates[0][2]   = 0.0;
array_data->inclinometer_coordinates[1][0]   = 20.0;
array_data->inclinometer_coordinates[1][1]   = 10.0;
array_data->inclinometer_coordinates[1][2]   = 0.0;
array_data->inclinometer_coordinates[2][0]   = -10.0;
array_data->inclinometer_coordinates[2][1]   = -20.0;
array_data->inclinometer_coordinates[2][2]   = 0.0;
array_data->inclinometer_phi_array[0]        = 0.0;
array_data->inclinometer_phi_array[1]        = 0.0;
array_data->inclinometer_phi_array[2]        = 0.0;
array_data->inclinometer_theta_array[0]      = 0.0;
array_data->inclinometer_theta_array[1]      = 0.0;
array_data->inclinometer_theta_array[2]      = 0.0;
array_data->inclinometer_rotation_array[0]   = 0.0;
array_data->inclinometer_rotation_array[1]   = 0.0;
array_data->inclinometer_rotation_array[2]   = 0.0;
array_data->inclinometer_temperature_array[0] = 65.0;
array_data->inclinometer_temperature_array[1] = 65.0;
array_data->inclinometer_temperature_array[2] = 65.0;
array_data->photogrammetric_x_location       = 0.0;
array_data->photogrammetric_y_location       = 0.0;
array_data->photogrammetric_z_location       = 0.0;
array_data->photogrammetric_yaw_angle        = 0.0;
array_data->photogrammetric_pitch_angle      = 0.0;
array_data->photogrammetric_roll_angle       = 0.0;
array_data->photogrammetric_number_targets   = 20;
array_data->photogrammetric_RMS_error        = 0.0;
strcpy_and_pad(array_data->traverse_configuration,“Standard configuration”,80);
strcpy_and_pad(array_data->sideline_traverse_coord_ref,“TUNNEL”,80);
strcpy_and_pad(array_data->array_traverse_coord_ref,“TUNNEL”,80);
strcpy_and_pad(array_data->inclinometer_coord_ref,“MODEL”,80);
```

```
// Facility data

strcpy_and_pad(facility_data->facility_name,"14- by 22-foot Subsonic Tunnel",80);
strcpy_and_pad(facility_data->facility_description,"Open Jet Configuration",80);
facility_data->total_facility_parameters = 4;

strcpy_and_pad(facility_data->facility_parameter[0].long_name,"PA",80);
strcpy_and_pad(facility_data->facility_parameter[0].description,
                                                    "Ambient Pressure",80);
strcpy_and_pad(facility_data->facility_parameter[0].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[0].units,"psfA",80);
strcpy_and_pad(facility_data->facility_parameter[0].value,"2129.52",80);

strcpy_and_pad(facility_data->facility_parameter[1].long_name,"PTOT",80);
strcpy_and_pad(facility_data->facility_parameter[1].description,
                                        "Settling Chamber Total Pressure",80);
strcpy_and_pad(facility_data->facility_parameter[1].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[1].units,"psfA",80);
strcpy_and_pad(facility_data->facility_parameter[1].value,"2129.429888",80);

strcpy_and_pad(facility_data->facility_parameter[2].long_name,"TA",80);
strcpy_and_pad(facility_data->facility_parameter[2].description,
                                      "Entrance Cone Ambient Temperature",80);
strcpy_and_pad(facility_data->facility_parameter[2].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[2].units,"deg F",80);
strcpy_and_pad(facility_data->facility_parameter[2].value,"57.315520",80);

strcpy_and_pad(facility_data->facility_parameter[3].long_name,"TDEW",80);
strcpy_and_pad(facility_data->facility_parameter[3].description,
                                         "Dew Point Temperature",80);
strcpy_and_pad(facility_data->facility_parameter[3].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[3].units,"deg F",80);
strcpy_and_pad(facility_data->facility_parameter[3].value,"29.393310",80);

for(i = 4; i < 200; i++){
   facility_data->facility_parameter[i].long_name = "";
   facility_data->facility_parameter[i].description = "";
   facility_data->facility_parameter[i].data_type = "";
   facility_data->facility_parameter[i].units = "";
   facility_data->facility_parameter[i].value = "";
}

// Model data

model_data->model_id = 1;
strcpy_and_pad(model_data->model_description,"HL-CRM Model",80);
model_data->total_model_parameters = 2;

strcpy_and_pad(model_data->model_parameter[0].long_name,"PITCHM",80);
strcpy_and_pad(model_data->model_parameter[0].description,
                                                    "Mast Pitch Angle",80);
strcpy_and_pad(model_data->model_parameter[0].data_type,"real",80);
strcpy_and_pad(model_data->model_parameter[0].units,"deg",80);
strcpy_and_pad(model_data->model_parameter[0].value,"0.000000",80);

strcpy_and_pad(model_data->model_parameter[1].long_name,"ALPHA",80);
strcpy_and_pad(model_data->model_parameter[1].description,
                                                  "Model Angle of Attack",80);
strcpy_and_pad(model_data->model_parameter[1].data_type,"real",80);
strcpy_and_pad(model_data->model_parameter[1].units,"deg",80);
strcpy_and_pad(model_data->model_parameter[1].value,"6.999002",80);

for(i = 2; i < 200; i++){
   model_data->model_parameter[i].long_name = "";
   model_data->model_parameter[i].description = "";
   model_data->model_parameter[i].data_type = "";
   model_data->model_parameter[i].units = "";
   model_data->model_parameter[i].value = "";
}

return;
```

Appendix D.2 - Populating Header Values in Ensemble Raw Data Files
C Code Fragment (assumes arrow access to structure)

```
// Version data

ensemble_version_data->netcdf_version = 3.4;   //If writing NetCDF files
ensemble_version_data->hdf5_version   = 1.8;   //If writing HDF5 files
ensemble_version_data->hdr_version    = 9.4;   //Note: Do not change these values

// Test data

ensemble_test_data->testno            = 643;
ensemble_test_data->runno             = 259;
ensemble_test_data->pointno           = 3541;
ensemble_test_data->identifier_prefix = 'X';
ensemble_test_data->GPS_start_time    = 28800.0;
ensemble_test_data->GPS_stop_time     = 28830.0;
strcpy_and_pad(ensemble_test_data->date_time,"12/20/2018 08:00",80);
strcpy_and_pad(ensemble_test_data->test_engineer,"William Humphreys",80);

// Channel data

For (i = 0; i < number_of_channels-1; i++){
   ensemble_channel_data->channo_array[i]             = i;
   ensemble_channel_data->sensor_sn_array[i]          = mic_sn[i];
   ensemble_channel_data->preamplifier_sn_array[i]    = preamp_sn[i];
   ensemble_channel_data->sensor_type_array[i]        = 1;
   ensemble_channel_data->preamplifier_type_array[i]  = 1;
   ensemble_channel_data->excitation_voltage_array[i] = 200.0;
   ensemble_channel_data->excitation_current_array[i] = 0.0;
}

for (i = 0; i < number_of_channels-1; i++){
   ensemble_channel_data->sensor_coord_array[i][0]   = sensor_x[i];
   ensemble_channel_data->sensor_coord_array[i][1]   = sensor_y[i];
   ensemble_channel_data->sensor_coord_array[i][2]   = sensor_z[i];
   strcpy_and_pad(ensemble_channel_data->sensor_coord_ref_array[i],"array",12);
}

// Calibration data

for (i = 0; i < number_of_channels-1; i++){
   ensemble_calibration_data->sensitivity_array[i] = 1500.0;
}

for (i = 0; i < number_of_bins-1; i++){
   ensemble_calibration_data.filter_response[i][0] =
                                        (float)(i-1)*frequency_resolution;
   ensemble_calibration_data.filter_response[i][1] = single(1.0);
   ensemble_calibration_data.filter_response[i][2] = single(0.0);
}

ensemble_calibration_data->number_of_cal_sources           = 3;
ensemble_calibration_data->cal_source_coordinates[0][0]    = 114.34;
ensemble_calibration_data->cal_source_coordinates[0][1]    = 22.85;
ensemble_calibration_data->cal_source_coordinates[0][2]    = -4.79;
ensemble_calibration_data->cal_source_coordinates[1][0]    = 137.56;
ensemble_calibration_data->cal_source_coordinates[1][1]    = 58.59;
ensemble_calibration_data->cal_source_coordinates[1][2]    = 0.99;
ensemble_calibration_data->cal_source_coordinates[2][0]    = 164.55;
ensemble_calibration_data->cal_source_coordinates[2][1]    = 99.85;
ensemble_calibration_data->cal_source_coordinates[2][2]    = 5.07;
strcpy_and_pad(ensemble_calibration_data->cal_source_coord_ref_array[0],"model",12);
strcpy_and_pad(ensemble_calibration_data->cal_source_coord_ref_array[1],"model",12);
strcpy_and_pad(ensemble_calibration_data->cal_source_coord_ref_array[2],"model",12);
```

```
ensemble_calibration_data->number_of_cal_measurements = 1;
strcpy_and_pad(ensemble_calibration_data->date_time_of_cal[0][0],
                                                    "12/20/2018 08:00",17);
strcpy_and_pad(ensemble_calibration_data->date_time_of_cal[1][0],
                                                    "12/20/2018 08:00",17);
strcpy_and_pad(ensemble_calibration_data->date_time_of_cal[2][0],
                                                    "12/20/2018 08:00",17);

for (i = 0; i < number_of_channels-1; i++){
   ensemble_calibration_data->cal_source_baseline_array[i][0][0]   = 0.0;
   ensemble_calibration_data->cal_source_baseline_array[i][1][0]   = 0.0;
   ensemble_calibration_data->cal_source_baseline_array[i][2][0]   = 0.0;
   ensemble_calibration_data->cal_source_level_array[i][0][0]      = 0.0;
   ensemble_calibration_data->cal_source_level_array[i][1][0]      = 0.0;
   ensemble_calibration_data->cal_source_level_array[i][2][0]      = 0.0;
   ensemble_calibration_data->injection_cal_baseline_array[i]      = 1.0;
   ensemble_calibration_data->injection_cal_level_array[i]         = 1.0;
}

for (i = 0; i < number_of_channels-1; i++){
   for(j = 0; j < number_of_bins-1; j++){
      ensemble_calibration_data->freq_calibration_array[i][j][0] =
                                          (float)(j-1)*frequency_resolution;
      ensemble_calibration_data.freq_calibration_array[i][j][1] = 1.0;
      ensemble_calibration_data.freq_calibration_array[i][j][2] = 0.0;
   }
}

// DAS data

ensemble_DAS_data->total_channels  = 97;
ensemble_DAS_data->adbits          = 24;

for (i = 0; i < number_of_channels-1; i++){
   ensemble_DAS_data->coupling_array[i]              = 1;
   ensemble_DAS_data->full_scale_range_array[i]      = 10000.0;
}

ensemble_DAS_data->trigger_mode  = 0;
ensemble_DAS_data->clock_mode    = 0;
ensemble_DAS_data->sample_count  = 3000000;
ensemble_DAS_data->sample_period = 0.005;

for (i = 0; i < number_of_channels-1; i++){
   ensemble_DAS_data->lpf_cutoff_freq_array[i]      = 102300.0;
   ensemble_DAS_data->lpf_pregain_array[i]          = 1.0;
   ensemble_DAS_data->lpf_postgain_array[i]         = 1.0;
   ensemble_DAS_data->hpf_cutoff_freq_array[i]      = 400.0;
   ensemble_DAS_data->hpf_pregain_array[i]          = 1.0;
   ensemble_DAS_data->hpf_postgain_array[i]         = 1.0;
   ensemble_DAS_data->external_gain_offset_array[i] = 20.0;
   ensemble_DAS_data->on_board_LP_filter_array[i]   = 0;
}

// Array data

ensemble_array_data->total_array_sensors              = 97;
ensemble_array_data->sideline_traverse_x              = 0.0;
ensemble_array_data->sideline_traverse_y              = 0.0;
ensemble_array_data->sideline_traverse_z              = 0.0;
ensemble_array_data->array_traverse_x                 = 0.0;
ensemble_array_data->array_traverse_y                 = 0.0;
ensemble_array_data->array_traverse_z                 = 0.0;
ensemble_array_data->array_elevation                  = 0.0;
ensemble_array_data->array_azimuth                    = 0.0;
ensemble_array_data->number_of_inclinometers          = 3;
ensemble_array_data->inclinometer_coordinates[0][0]   = 10.0;
ensemble_array_data->inclinometer_coordinates[0][1]   = 20.0;
ensemble_array_data->inclinometer_coordinates[0][2]   = 0.0;
ensemble_array_data->inclinometer_coordinates[1][0]   = 20.0;
ensemble_array_data->inclinometer_coordinates[1][1]   = 10.0;
ensemble_array_data->inclinometer_coordinates[1][2]   = 0.0;
```

```
ensemble_array_data->inclinometer_coordinates[2][0]    = -10.0;
ensemble_array_data->inclinometer_coordinates[2][1]    = -20.0;
ensemble_array_data->inclinometer_coordinates[2][2]    = 0.0;
ensemble_array_data->inclinometer_phi_array[0]         = 0.0;
ensemble_array_data->inclinometer_phi_array[1]         = 0.0;
ensemble_array_data->inclinometer_phi_array[2]         = 0.0;
ensemble_array_data->inclinometer_theta_array[0]       = 0.0;
ensemble_array_data->inclinometer_theta_array[1]       = 0.0;
ensemble_array_data->inclinometer_theta_array[2]       = 0.0;
ensemble_array_data->inclinometer_rotation_array[0]    = 0.0;
ensemble_array_data->inclinometer_rotation_array[1]    = 0.0;
ensemble_array_data->inclinometer_rotation_array[2]    = 0.0;
ensemble_array_data->inclinometer_temperature_array[0] = 65.0;
ensemble_array_data->inclinometer_temperature_array[1] = 65.0;
ensemble_array_data->inclinometer_temperature_array[2] = 65.0;
ensemble_array_data->photogrammetric_x_location        = 0.0;
ensemble_array_data->photogrammetric_y_location        = 0.0;
ensemble_array_data->photogrammetric_z_location        = 0.0;
ensemble_array_data->photogrammetric_yaw_angle         = 0.0;
ensemble_array_data->photogrammetric_pitch_angle       = 0.0;
ensemble_array_data->photogrammetric_roll_angle        = 0.0;
ensemble_array_data->photogrammetric_number_targets    = 20;
ensemble_array_data->photogrammetric_RMS_error         = 0.0;
strcpy_and_pad(ensemble_array_data->traverse_configuration,
                                                  "Standard configuration",80);
strcpy_and_pad(ensemble_array_data->sideline_traverse_coord_ref,"tunnel",12);
strcpy_and_pad(ensemble_array_data->array_traverse_coord_ref,"tunnel",12);
strcpy_and_pad(ensemble_array_data->inclinometer_coord_ref,"model",12);

// Facility data

strcpy_and_pad(facility_data->facility_name,"14- by 22-foot Subsonic Tunnel",80);
strcpy_and_pad(facility_data->facility_description,"Open Jet Configuration",80);
facility_data->total_facility_parameters = 4;

strcpy_and_pad(facility_data->facility_parameter[0].long_name,"PA",80);
strcpy_and_pad(facility_data->facility_parameter[0].description,
                                                     "Ambient Pressure",80);
strcpy_and_pad(facility_data->facility_parameter[0].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[0].units,"psfA",80);
strcpy_and_pad(facility_data->facility_parameter[0].value,"2129.52",80);

strcpy_and_pad(facility_data->facility_parameter[1].long_name,"PTOT",80);
strcpy_and_pad(facility_data->facility_parameter[1].description,
                                          "Settling Chamber Total Pressure",80);
strcpy_and_pad(facility_data->facility_parameter[1].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[1].units,"psfA",80);
strcpy_and_pad(facility_data->facility_parameter[1].value,"2129.429888",80);

strcpy_and_pad(facility_data->facility_parameter[2].long_name,"TA",80);
strcpy_and_pad(facility_data->facility_parameter[2].description,
                                        "Entrance Cone Ambient Temperature",80);
strcpy_and_pad(facility_data->facility_parameter[2].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[2].units,"deg_F",80);
strcpy_and_pad(facility_data->facility_parameter[2].value,"57.315520",80);

strcpy_and_pad(facility_data->facility_parameter[3].long_name,"TDEW",80);
strcpy_and_pad(facility_data->facility_parameter[3].description,
                                                   "Dew Point Temperature",80);
strcpy_and_pad(facility_data->facility_parameter[3].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[3].units,"deg_F",80);
strcpy_and_pad(facility_data->facility_parameter[3].value,"29.393310",80);

for(i = 4; i < 200; i++){
   facility_data.facility_parameter[i].long_name = "";
   facility_data.facility_parameter[i].description = "";
   facility_data.facility_parameter[i].data_type = "";
   facility_data.facility_parameter[i].units = "";
   facility_data.facility_parameter[i].value = "";
}
```

```
// Model data

model_data->model_id = 1;
strcpy_and_pad(model_data->model_description,"HL-CRM Model",80);
model_data->total_model_parameters = 2;

strcpy_and_pad(model_data->model_parameter[0].long_name,"PITCHM",80);
strcpy_and_pad(model_data->model_parameter[0].description,"Mast Pitch Angle",80);
strcpy_and_pad(model_data->model_parameter[0].data_type,"real",80);
strcpy_and_pad(model_data->model_parameter[0].units,"deg",80);
strcpy_and_pad(model_data->model_parameter[0].value,"0.000000",80);

strcpy_and_pad(model_data->model_parameter[1].long_name,"ALPHA",80);
strcpy_and_pad(model_data->model_parameter[1].description,"Model Angle of Attack",80);
strcpy_and_pad(model_data->model_parameter[1].data_type,"real",80);
strcpy_and_pad(model_data->model_parameter[1].units,"deg",80);
strcpy_and_pad(model_data->model_parameter[1].value,"6.999002",80);

for(i = 2; i < 200; i++){
   model_data->model_parameter[i].long_name = "";
   model_data->model_parameter[i].description = "";
   model_data->model_parameter[i].data_type = "";
   model_data->model_parameter[i].units = "";
   model_data->model_parameter[i].value = "";
}

return;
```

Appendix D.3 - Populating Header Values in Pressure Data Files
C Code Fragment (assumes arrow access to structure)

```
// Version data

version_data->netcdf_version = 3.4;     //If writing NetCDF files
version_data->hdf5_version   = 1.8;     //If writing HDF5 files
version_data->hdr_version    = 9.4;     //Note: Do not change these values

// Test data

test_data->testno            = 643;
test_data->runno             = 259;
test_data->pointno           = 3541;
test_data->GPS_start_time    = 28800.0;
test_data->GPS_stop_time     = 28830.0;
strcpy_and_pad(test_data->date_time,"12/20/2018 08:00",80);
strcpy_and_pad(test_data->test_engineer,"William Humphreys",80);

// Facility data

strcpy_and_pad(facility_data->facility_name,"14- by 22-foot Subsonic Tunnel",80);
strcpy_and_pad(facility_data->facility_description,"Open Jet Configuration",80);
facility_data->total_facility_parameters = 4;

strcpy_and_pad(facility_data->facility_parameter[0].long_name,"PA",80);
strcpy_and_pad(facility_data->facility_parameter[0].description,
                                                          "Ambient Pressure",80);
strcpy_and_pad(facility_data->facility_parameter[0].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[0].units,"psfA",80);
strcpy_and_pad(facility_data->facility_parameter[0].value,"2129.52",80);

strcpy_and_pad(facility_data->facility_parameter[1].long_name,"PTOT",80);
strcpy_and_pad(facility_data->facility_parameter[1].description,
                                              "Settling Chamber Total Pressure",80);
strcpy_and_pad(facility_data->facility_parameter[1].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[1].units,"psfA",80);
strcpy_and_pad(facility_data->facility_parameter[1].value,"2129.429888",80);

strcpy_and_pad(facility_data->facility_parameter[2].long_name,"TA",80);
strcpy_and_pad(facility_data->facility_parameter[2].description,
                                            "Entrance Cone Ambient Temperature",80);
strcpy_and_pad(facility_data->facility_parameter[2].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[2].units,"deg F",80);
strcpy_and_pad(facility_data->facility_parameter[2].value,"57.315520",80);

strcpy_and_pad(facility_data->facility_parameter[3].long_name,"TDEW",80);
strcpy_and_pad(facility_data->facility_parameter[3].description,
                                                        "Dew Point Temperature",80);
strcpy_and_pad(facility_data->facility_parameter[3].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[3].units,"deg F",80);
strcpy_and_pad(facility_data->facility_parameter[3].value,"29.393310",80);

for(i = 4; i < 200; i++){
   facility_data.facility_parameter[i].long_name = "";
   facility_data.facility_parameter[i].description = "";
   facility_data.facility_parameter[i].data_type = "";
   facility_data.facility_parameter[i].units = "";
   facility_data.facility_parameter[i].value = "";
}

// Model data

model_data->model_id = 1;
strcpy_and_pad(model_data->model_description,"HL-CRM Model",80);
model_data->total_model_parameters = 2;

strcpy_and_pad(model_data->model_parameter[0].long_name,"PITCHM",80);
strcpy_and_pad(model_data->model_parameter[0].description,"Mast Pitch Angle",80);
strcpy_and_pad(model_data->model_parameter[0].data_type,"real",80);
```

```
strcpy_and_pad(model_data->model_parameter[0].units,”deg”,80);
strcpy_and_pad(model_data->model_parameter[0].value,”0.000000”,80);

strcpy_and_pad(model_data->model_parameter[1].long_name,”ALPHA”,80);
strcpy_and_pad(model_data->model_parameter[1].description,”Model Angle of Attack”,80);
strcpy_and_pad(model_data->model_parameter[1].data_type,”real”,80);
strcpy_and_pad(model_data->model_parameter[1].units,”deg”,80);
strcpy_and_pad(model_data->model_parameter[1].value,”6.999002”,80);

for(i = 2; i < 200; i++){
   model_data->model_parameter[i].long_name = “”;
   model_data->model_parameter[i].description = “”;
   model_data->model_parameter[i].data_type = “”;
   model_data->model_parameter[i].units = “”;
   model_data->model_parameter[i].value = “”;
}

// Static tap data

static_data->total_static_taps = 3;

strcpy_and_pad(static_data->static_tap_parameter[0].static_tap_designation,
               ”ESP0101”,80);
strcpy_and_pad(static_data->static_tap_parameter[0].description,
                                                ”PSI Pressure Tap”,80);
strcpy_and_pad(static_data->static_tap_parameter[0].units,”psi”,80);
strcpy_and_pad(static_data->static_tap_parameter[0].coefficient_name,”ESP0101”,80);
strcpy_and_pad(static_data->static_tap_parameter[0].data_value,”0.109163”,80);

strcpy_and_pad(static_data->static_tap_parameter[1].static_tap_designation,
                                                ”ESP0102”,80);
strcpy_and_pad(static_data->static_tap_parameter[1].description,
                                                ”PSI Pressure Tap”,80);
strcpy_and_pad(static_data->static_tap_parameter[1].units,”psi”,80);
strcpy_and_pad(static_data->static_tap_parameter[1].coefficient_name,”ESP0102”,80);
strcpy_and_pad(static_data->static_tap_parameter[1].data_value,”0.052045”,80);

strcpy_and_pad(static_data->static_tap_parameter[2].static_tap_designation,
                                                ”ESP0103”,80);
strcpy_and_pad(static_data->static_tap_parameter[2].description,
                                                ”PSI Pressure Tap”,80);
strcpy_and_pad(static_data->static_tap_parameter[2].units,”psi”,80);
strcpy_and_pad(static_data->static_tap_parameter[2].coefficient_name,”ESP0103”,80);
strcpy_and_pad(static_data->static_tap_parameter[2].data_value,”-0.058525”,80);

for(i = 3; i < 1000; i++){
   model_data->model_parameter[i].static_tap_designation = “”;
   model_data->model_parameter[i].description = “”;
   model_data->model_parameter[i].units = “”;
   model_data->model_parameter[i].coefficient_name = “”;
   model_data->model_parameter[i].data_value = “”;
}

return;
```

Appendix D.4 - Populating Header Values in Cross Spectral Matrix Data Files
C Code Fragment (assumes arrow access to structure)

```
// Version data

csm_version_data->netcdf_version = 3.4;        //If writing NetCDF files
csm_version_data->hdf5_version   = 1.8;        //If writing HDF5 files
csm_version_data->hdr_version    = 9.4;        //Note: Do not change these values


// Test data

csm_test_data->testno            = 643;
csm_test_data->runno             = 259;
csm_test_data->pointno           = 3541;
csm_test_data->GPS_start_time    = 28800.0;
csm_test_data->GPS_stop_time     = 28830.0;
strcpy_and_pad(csm_test_data->date_time,"12/20/2018 08:00",17);

// Channel data

for (i = 1; i < number_of_channels; i++){
   csm_channel_data->sensor_sn_array[i]          = mic_sn(i);
   csm_channel_data->preamplifier_sn_array[i]    = preamp_sn(i);
   csm_channel_data->sensor_type_array[i]        = 1;
   csm_channel_data->preamplifier_type_array[i]  = 1;
   csm_channel_data->excitation_voltage_array[i] = 200.0;
   csm_channel_data->excitation_current_array[i] = 0.0;
}

for (i = 0; i < number_of_channels-1; i++){
   csm_channel_data->sensor_coord_array[i][0]        = sensor_x[i];
   csm_channel_data->sensor_coord_array[i][1]        = sensor_y[i];
   csm_channel_data->sensor_coord_array[i][2]        = sensor_z[i];
}

strcpy_and_pad(csm_channel_data->array_sensor_coord_ref,"array",12);
csm_channel_data->sideline_sensor_coord_ref,"tunnel",12);
csm_channel_data->kulite_sensor_coord_ref,"model",12);

// Calibration data

for (i = 0; i < number_of_channels-1; i++){
   csm_calibration_data->sensitivity_array[i] = 1500.0;
}

for (i = 0; i < number_of_bins-1; i++){
   csm_calibration_data.filter_response[i][0] =
                                        (float)(i-1)*frequency_resolution;
   csm_calibration_data.filter_response[i][1] = single(1.0);
   csm_calibration_data.filter_response[i][2] = single(0.0);
}

csm_calibration_data->number_of_cal_sources           = 3;
csm_calibration_data->cal_source_coordinates[0][0]    = 114.34;
csm_calibration_data->cal_source_coordinates[0][1]    = 22.85;
csm_calibration_data->cal_source_coordinates[0][2]    = -4.79;
csm_calibration_data->cal_source_coordinates[1][0]    = 137.56;
csm_calibration_data->cal_source_coordinates[1][1]    = 58.59;
csm_calibration_data->cal_source_coordinates[1][2]    = 0.99;
csm_calibration_data->cal_source_coordinates[2][0]    = 164.55;
csm_calibration_data->cal_source_coordinates[2][1]    = 99.85;
csm_calibration_data->cal_source_coordinates[2][2]    = 5.07;
strcpy_and_pad(csm_calibration_data->cal_source_coord_ref_array[0],"model",12);
strcpy_and_pad(csm_calibration_data->cal_source_coord_ref_array[1],"model",12);
strcpy_and_pad(csm_calibration_data->cal_source_coord_ref_array[2],"model",12);
csm_calibration_data->number_of_cal_measurements = 1;
strcpy_and_pad(csm_calibration_data->date_time_of_cal[0][0],
                                                    "12/20/2018 08:00",17);
strcpy_and_pad(csm_calibration_data->date_time_of_cal[1][0],
                                                    "12/20/2018 08:00",17);
```

```
strcpy_and_pad(csm_calibration_data->date_time_of_cal[2][0],
                                                    ”12/20/2018 08:00”,17);

for (i = 0; i < number_of_channels-1; i++){
   csm_calibration_data->cal_source_baseline_array[i][0][0]       = 0.0;
   csm_calibration_data->cal_source_baseline_array[i][1][0]       = 0.0;
   csm_calibration_data->cal_source_baseline_array[i][2][0]       = 0.0;
   csm_calibration_data->cal_source_level_array[i][0][0]          = 0.0;
   csm_calibration_data->cal_source_level_array[i][1][0]          = 0.0;
   csm_calibration_data->cal_source_level_array[i][2][0]          = 0.0;
   csm_calibration_data->injection_cal_baseline_array[i]    = 1.0;
   csm_calibration_data->injection_cal_level_array[i]       = 1.0;
}

for (i = 0; i < number_of_channels-1; i++){
   for(j = 0; j < number_of_bins-1; j++){
      csm_calibration_data->freq_calibration_array[i][j][0] =
                                          (float)(j-1)*frequency_resolution;
      csm_calibration_data.freq_calibration_array[i][j][1] = 1.0;
      csm_calibration_data.freq_calibration_array[i][j][2] = 0.0;
   }
}

// DAS data

csm_DAS_data->total_channels  = 97;
csm_DAS_data->sample_count  = 3000000;
csm_DAS_data->sample_period = 0.005;

for (i = 0; i < number_of_channels-1; i++){
   csm_DAS_data->full_scale_range_array[i]     = 10000.0;
}

for (i = 0; i < number_of_channels-1; i++){
   csm_DAS_data->lpf_cutoff_freq_array[i]      = 102300.0;
   csm_DAS_data->lpf_pregain_array[i]          = 1.0;
   csm_DAS_data->lpf_postgain_array[i]         = 1.0;
   csm_DAS_data->hpf_cutoff_freq_array[i]      = 400.0;
   csm_DAS_data->hpf_pregain_array[i]          = 1.0;
   csm_DAS_data->hpf_postgain_array[i]         = 1.0;
   csm_DAS_data->external_gain_offset_array[i] = 20.0;
}

// Array data

csm_array_data->total_array_sensors               = 97;
csm_array_data->sideline_traverse_x               = 0.0;
csm_array_data->sideline_traverse_y               = 0.0;
csm_array_data->sideline_traverse_z               = 0.0;
csm_array_data->array_traverse_x                  = 0.0;
csm_array_data->array_traverse_y                  = 0.0;
csm_array_data->array_traverse_z                  = 0.0;
csm_array_data->array_elevation                   = 0.0;
csm_array_data->array_azimuth                     = 0.0;
csm_array_data->number_of_inclinometers           = 3;
csm_array_data->inclinometer_coordinates[0][0]    = 10.0;
csm_array_data->inclinometer_coordinates[0][1]    = 20.0;
csm_array_data->inclinometer_coordinates[0][2]    = 0.0;
csm_array_data->inclinometer_coordinates[1][0]    = 20.0;
csm_array_data->inclinometer_coordinates[1][1]    = 10.0;
csm_array_data->inclinometer_coordinates[1][2]    = 0.0;
csm_array_data->inclinometer_coordinates[2][0]    = -10.0;
csm_array_data->inclinometer_coordinates[2][1]    = -20.0;
csm_array_data->inclinometer_coordinates[2][2]    = 0.0;
csm_array_data->inclinometer_phi_array[0]         = 0.0;
csm_array_data->inclinometer_phi_array[1]         = 0.0;
csm_array_data->inclinometer_phi_array[2]         = 0.0;
csm_array_data->inclinometer_theta_array[0]       = 0.0;
csm_array_data->inclinometer_theta_array[1]       = 0.0;
csm_array_data->inclinometer_theta_array[2]       = 0.0;
csm_array_data->inclinometer_rotation_array[0]    = 0.0;
csm_array_data->inclinometer_rotation_array[1]    = 0.0;
```

```
csm_array_data->inclinometer_rotation_array[2]    = 0.0;
csm_array_data->inclinometer_temperature_array[0] = 65.0;
csm_array_data->inclinometer_temperature_array[1] = 65.0;
csm_array_data->inclinometer_temperature_array[2] = 65.0;
csm_array_data->photogrammetric_x_location        = 0.0;
csm_array_data->photogrammetric_y_location        = 0.0;
csm_array_data->photogrammetric_z_location        = 0.0;
csm_array_data->photogrammetric_yaw_angle         = 0.0;
csm_array_data->photogrammetric_pitch_angle       = 0.0;
csm_array_data->photogrammetric_roll_angle        = 0.0;
csm_array_data->photogrammetric_number_targets    = 20;
csm_array_data->photogrammetric_RMS_error         = 0.0;
strcpy_and_pad(csm_array_data->traverse_configuration,
                                                  "Standard configuration",80);
strcpy_and_pad(csm_array_data->sideline_traverse_coord_ref,"tunnel",12);
strcpy_and_pad(csm_array_data->array_traverse_coord_ref,"tunnel",12);
strcpy_and_pad(csm_array_data->inclinometer_coord_ref,"model",12);

// Facility data

strcpy_and_pad(facility_data->facility_name,"14- by 22-foot Subsonic Tunnel",80);
strcpy_and_pad(facility_data->facility_description,"Open Jet Configuration",80);
facility_data->total_facility_parameters = 4;

strcpy_and_pad(facility_data->facility_parameter[0].long_name,"PA",80);
strcpy_and_pad(facility_data->facility_parameter[0].description,
                                                     "Ambient Pressure",80);
strcpy_and_pad(facility_data->facility_parameter[0].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[0].units,"psfA",80);
strcpy_and_pad(facility_data->facility_parameter[0].value,"2129.52",80);

strcpy_and_pad(facility_data->facility_parameter[1].long_name,"PTOT",80);
strcpy_and_pad(facility_data->facility_parameter[1].description,
                                          "Settling Chamber Total Pressure",80);
strcpy_and_pad(facility_data->facility_parameter[1].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[1].units,"psfA",80);
strcpy_and_pad(facility_data->facility_parameter[1].value,"2129.429888",80);

strcpy_and_pad(facility_data->facility_parameter[2].long_name,"TA",80);
strcpy_and_pad(facility_data->facility_parameter[2].description,
                                        "Entrance Cone Ambient Temperature",80);
strcpy_and_pad(facility_data->facility_parameter[2].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[2].units,"deg_F",80);
strcpy_and_pad(facility_data->facility_parameter[2].value,"57.315520",80);

strcpy_and_pad(facility_data->facility_parameter[3].long_name,"TDEW",80);
strcpy_and_pad(facility_data->facility_parameter[3].description,
                                                   "Dew Point Temperature",80);
strcpy_and_pad(facility_data->facility_parameter[3].data_type,"real",80);
strcpy_and_pad(facility_data->facility_parameter[3].units,"deg_F",80);
strcpy_and_pad(facility_data->facility_parameter[3].value,"29.393310",80);

for(i = 4; i < 200; i++){
   facility_data.facility_parameter[i].long_name = "";
   facility_data.facility_parameter[i].description = "";
   facility_data.facility_parameter[i].data_type = "";
   facility_data.facility_parameter[i].units = "";
   facility_data.facility_parameter[i].value = "";
}

// Model data

model_data->model_id = 1;
strcpy_and_pad(model_data->model_description,"HL-CRM Model",80);
model_data->total_model_parameters = 2;

strcpy_and_pad(model_data->model_parameter[0].long_name,"PITCHM",80);
strcpy_and_pad(model_data->model_parameter[0].description,"Mast Pitch Angle",80);
strcpy_and_pad(model_data->model_parameter[0].data_type,"real",80);
strcpy_and_pad(model_data->model_parameter[0].units,"deg",80);
strcpy_and_pad(model_data->model_parameter[0].value,"0.000000",80);
```

```
strcpy_and_pad(model_data->model_parameter[1].long_name,"ALPHA",80);
strcpy_and_pad(model_data->model_parameter[1].description,"Model Angle of Attack",80);
strcpy_and_pad(model_data->model_parameter[1].data_type,"real",80);
strcpy_and_pad(model_data->model_parameter[1].units,"deg",80);
strcpy_and_pad(model_data->model_parameter[1].value,"6.999002",80);

for(i = 2; i < 200; i++){
   model_data->model_parameter[i].long_name = "";
   model_data->model_parameter[i].description = "";
   model_data->model_parameter[i].data_type = "";
   model_data->model_parameter[i].units = "";
   model_data->model_parameter[i].value = "";
}

// CSM data

csm_data->fft_length      = 8192;
csm_data->overlap         = 0;
csm_data->num_blocks      = 500;
csm_data->num_blocks_used = 500;

for(i = 0; i < number_of_blocks; i++){
   for(j = 0; j < number_of_channels; j++){
       block_table[i][j] = 1;
   }
}

csm_data->cslength  = 4096;
strcpy_and_pad(csm_data->winfunc,"hamming",11);
csm_data->beta      = 0.0;
csm_data->CSM_units = 1;

return;
```

Appendix E.1 - Populating Header Values in Individual Raw Data Files
Fortran Code Fragment

```
! Version data

version_data%netcdf_version = 3.4             !Note: Do not change the version values
version_data%hdf5_version   = 1.8
version_data%hdr_version    = 9.4

! Test data

test_data%testno         = 643
test_data%runno          = 259
test_data%pointno        = 3541
test_data%date_time      = '12/20/2018 08:00 '
test_data%GPS_start_time = 28800.
test_data%GPS_stop_time  = 28830.
test_data%test_engineer  = 'William Humphreys'

! Channel data

channel_data%channo                   = 1
channel_data%nam_chan                 = 'Channel 001'
channel_data%chan_description         = 'Array Channel'
channel_data%sensor_sn                = 2688098
channel_data%preamplifier_sn          = 2688519
channel_data%sensor_type              = 1
channel_data%preamplifier_type        = 1
channel_data%sensor_description       = 'B&K 4938 Pressure Microphone'
channel_data%preamplifier_description = 'B&K 2670 1/4-inch Preamplifier'
channel_data%excitation_voltage       = 200.
channel_data%excitation_current       = 0.
channel_data%sensor_x                 = -0.041
channel_data%sensor_y                 = 0.999
channel_data%sensor_z                 = 0.
channel_data%sensor_coord_ref         = 'ARRAY'

! Calibration data

calibration_data%sensitivity                  = 63.
calibration_data%number_of_cal_sources        = 3
calibration_data%cal_source_coordinates(1,1)  = 114.34
calibration_data%cal_source_coordinates(1,2)  = 22.85
calibration_data%cal_source_coordinates(1,3)  = -4.79
calibration_data%cal_source_coordinates(2,1)  = 137.56
calibration_data%cal_source_coordinates(2,2)  = 58.59
calibration_data%cal_source_coordinates(2,3)  = 0.99
calibration_data%cal_source_coordinates(3,1)  = 164.55
calibration_data%cal_source_coordinates(3,2)  = 99.85
calibration_data%cal_source_coordinates(3,3)  = 5.07
calibration_data%cal_source_coord_ref         = ‘MODEL'
calibration_data%cal_source_baseline_array(1) = 0.
calibration_data%cal_source_baseline_array(2) = 0.
calibration_data%cal_source_baseline_array(3) = 0.
calibration_data%cal_source_level_array(1)    = 0.
calibration_data%cal_source_level_array(2)    = 0.
calibration_data%cal_source_level_array(3)    = 0.
calibration_data%injection_cal_baseline       = 1.
calibration_data%injection_cal_level          = 1.
do i = 1, number_of_bins
    calibration_data%calibration_frequency(i) = float(i-1)*frequency_resolution
    calibration_data%calibration_magnitude(i) = 1.
    calibration_data%calibration_phase(i)     = 0.
end do
```

```
! DAS data

DAS_data%total_channels       = 97
DAS_data%adbits               = 24
DAS_data%coupling             = 1
DAS_data%full_scale_range     = 10000.
DAS_data%trigger_mode         = 0
DAS_data%clock_mode           = 0
DAS_data%sample_count         = 3000000
DAS_data%sample_period        = 0.005
DAS_data%lpf_cutoff_freq      = 102300.
DAS_data%lpf_pregain          = 1.
DAS_data%lpf_postgain         = 1.
DAS_data%hpf_cutoff_freq      = 400.
DAS_data%hpf_pregain          = 1.
DAS_data%hpf_postgain         = 1.
DAS_data%external_gain_offset = 20.
DAS_data%on_board_LP_filter   = 0

! Array data

array_data%total_array_sensors                = 97
array_data%traverse_configuration             = 'Standard configuration'
array_data%sideline_traverse_x                = 0.
array_data%sideline_traverse_y                = 0.
array_data%sideline_traverse_z                = 0.
array_data%sideline_traverse_coord_ref        = 'TUNNEL'
array_data%array_traverse_x                   = 0.
array_data%array_traverse_y                   = 0.
array_data%array_traverse_z                   = 0.
array_data%array_traverse_coord_ref           = 'TUNNEL';
array_data%array_elevation                    = 0.
array_data%array_azimuth                      = 0.
array_data%number_of_inclinometers            = 3
array_data%inclinometer_coordinates(1,1)      = 10.
array_data%inclinometer_coordinates(1,2)      = 20.
array_data%inclinometer_coordinates(1,3)      = 0.
array_data%inclinometer_coordinates(2,1)      = 20.
array_data%inclinometer_coordinates(2,2)      = 10.
array_data%inclinometer_coordinates(2,3)      = 0.
array_data%inclinometer_coordinates(3,1)      = -10.
array_data%inclinometer_coordinates(3,2)      = -20.
array_data%inclinometer_coordinates(3,3)      = 0.
array_data%inclinometer_coord_ref             = 'MODEL'
array_data%inclinometer_phi_array(1)          = 0.
array_data%inclinometer_phi_array(2)          = 0.
array_data%inclinometer_phi_array(3)          = 0.
array_data%inclinometer_theta_array(1)        = 0.
array_data%inclinometer_theta_array(2)        = 0.
array_data%inclinometer_theta_array(3)        = 0.
array_data%inclinometer_rotation_array(1)     = 0.
array_data%inclinometer_rotation_array(2)     = 0.
array_data%inclinometer_rotation_array(3)     = 0.
array_data%inclinometer_temperature_array(1) = 65.
array_data%inclinometer_temperature_array(2) = 65.
array_data%inclinometer_temperature_array(3) = 65.
array_data%photogrammetric_x_location         = 0.
array_data%photogrammetric_y_location         = 0.
array_data%photogrammetric_z_location         = 0.
array_data%photogrammetric_yaw_angle          = 0.
array_data%photogrammetric_pitch_angle        = 0.
array_data%photogrammetric_roll_angle         = 0.
array_data%photogrammetric_number_targets     = 20
array_data%photogrammetric_RMS_error          = 0.
```

```
! Facility data

facility_data%facility_name = '14- by 22-foot Subsonic Tunnel'
facility_data%facility_description = 'Open Jet Configuration'
facility_data%total_facility_parameters = 4

facility_data%facility_parameter(1)%long_name = ‘PA’
facility_data%facility_parameter(1)%description = ‘Ambient Pressure’
facility_data%facility_parameter(1)%data_type = ‘real’
facility_data%facility_parameter(1)%units = ‘psfA’
facility_data%facility_parameter(1)%value = ’2129.52’

facility_data%facility_parameter(2)%long_name = ‘PTOT’;
facility_data%facility_parameter(2)%description = ‘Settling Chamber Total Pressure’;
facility_data%facility_parameter(2)%data_type = ‘real’;
facility_data%facility_parameter(2)%units = ‘psfA’;
facility_data%facility_parameter(2)%value = ‘2129.429888’;

facility_data%facility_parameter(3)%long_name = ‘TA’;
facility_data%facility_parameter(3)%description = ‘Entrance Cone Ambient Temperature’;
facility_data%facility_parameter(3)%data_type = ‘real’;
facility_data%facility_parameter(3)%units = ‘deg F’;
facility_data%facility_parameter(3)%value = ’57.315520’;

facility_data%facility_parameter(4)%long_name = ‘TDEW’;
facility_data%facility_parameter(4)%description = ‘Dew Point Temperature’;
facility_data%facility_parameter(4)%data_type = ‘real’;
facility_data%facility_parameter(4)%units = ‘deg F’;
facility_data%facility_parameter(4)%value = ’29.393310’;

do i = 5, 200
   facility_data%facility_parameter(i)%long_name = ‘’;
   facility_data%facility_parameter(i)%description = ‘’;
   facility_data%facility_parameter(i)%data_type = ‘’;
   facility_data%facility_parameter(i)%units = ‘’;
   facility_data%facility_parameter(i)%value = ‘’;
end do

! Model data

model_data%model_id = 1
model_data%model_description = ‘HL-CRM Model'
model_data%total_model_parameters = 2

model_data%model_parameter(1)%long_name = ‘PITCHM’;
model_data%model_parameter(1)%description = ‘Mast Pitch Angle’
model_data%model_parameter(1)%data_type = ‘real’
model_data%model_parameter(1)%units = ‘deg’
model_data%model_parameter(1)%value = ‘0.000000’

model_data%model_parameter(2)%long_name = ‘ALPHA’
model_data%model_parameter(2)%description = ‘Model Angle of Attack’
model_data%model_parameter(2)%data_type = ‘real’
model_data%model_parameter(2)%units = ‘deg’
model_data%model_parameter(2)%value = ‘6.999002’

do i = 3, 200
   model_data%model_parameter(i)%long_name = ‘’
   model_data%model_parameter(i)%description = ‘’
   model_data%model_parameter(i)%data_type = ‘’
   model_data%model_parameter(i)%units = ‘’
   model_data%model_parameter(i)%value = ‘’
end do

return
```

Appendix E.2 - Populating Header Values in Ensemble Raw Data Files
Fortran Code Fragment

```
! Version data

ensemble_version_data%netcdf_version = 3.4     !Note: Do not change these values
ensemble_version_data%hdf5_version   = 1.8
ensemble_version_data%hdr_version    = 9.4

! Test data

ensemble_test_data%testno            = 643
ensemble_test_data%runno             = 259
ensemble_test_data%pointno           = 3541
ensemble_test_data%identifier_prefix = 'X'
ensemble_test_data%date_time         = '12/20/2018 08:00 '
ensemble_test_data%GPS_start_time    = 28800.
ensemble_test_data%GPS_stop_time     = 28830.
ensemble_test_data%test_engineer     = 'William Humphreys'

! Channel data

do i = 1, number_of_channels
   ensemble_channel_data%channo_array(i)             = i
   ensemble_channel_data%sensor_sn_array(i)          = mic_sn(i)
   ensemble_channel_data%preamplifier_sn_array(i)    = preamp_sn(i)
   ensemble_channel_data%sensor_type_array(i)        = 1
   ensemble_channel_data%preamplifier_type_array(i)  = 1
   ensemble_channel_data%excitation_voltage_array(i) = 200.
   ensemble_channel_data%excitation_current_array(i) = 0.
end do

do i = 1, number_of_channels
   ensemble_channel_data%sensor_coord_array(i,1)     = sensor_x(i)
   ensemble_channel_data%sensor_coord_array(i,2)     = sensor_y(i)
   ensemble_channel_data%sensor_coord_array(i,3)     = sensor_z(i)
   ensemble_channel_data%sensor_coord_ref_array(i)   = 'array'
end do

! Calibration data

do i = 1, number_of_channels
  ensemble_calibration_data%sensitivity_array(i) = 1500.
end do

do i = 1, number_of_bins
  ensemble_calibration_data%filter_response(i,1) = float(i-1)*frequency_resolution
  ensemble_calibration_data%filter_response(i,2) = 1.
  ensemble_calibration_data%filter_response(i,3) = 0.
end do

ensemble_calibration_data%number_of_cal_sources           = 3
ensemble_calibration_data%cal_source_coordinates(1,1)     = 114.34
ensemble_calibration_data%cal_source_coordinates(1,2)     = 22.85
ensemble_calibration_data%cal_source_coordinates(1,3)     = -4.79
ensemble_calibration_data%cal_source_coordinates(2,1)     = 137.56
ensemble_calibration_data%cal_source_coordinates(2,2)     = 58.59
ensemble_calibration_data%cal_source_coordinates(2,3)     = 0.99
ensemble_calibration_data%cal_source_coordinates(3,1)     = 164.55
ensemble_calibration_data%cal_source_coordinates(3,2)     = 99.85
ensemble_calibration_data%cal_source_coordinates(3,3)     = 5.07
ensemble_calibration_data%cal_source_coord_ref_array(1)   = 'model'
ensemble_calibration_data%cal_source_coord_ref_array(2)   = 'model'
ensemble_calibration_data%cal_source_coord_ref_array(3)   = 'model'

ensemble_calibration_data%number_of_cal_measurements = 1
ensemble_calibration_data%date_time_of_cal(1,1)      = '12/20/2018 08:00 '
ensemble_calibration_data%date_time_of_cal(2,1)      = '12/20/2018 08:00 '
ensemble_calibration_data%date_time_of_cal(3,1)      = '12/20/2018 08:00 '
```

```
do i = 1, number_of_channels
   ensemble_calibration_data%cal_source_baseline_array(i,1,1) = 0.
   ensemble_calibration_data%cal_source_baseline_array(i,2,1) = 0.
   ensemble_calibration_data%cal_source_baseline_array(i,3,1) = 0.
   ensemble_calibration_data%cal_source_level_array(i,1,1)    = 0.
   ensemble_calibration_data%cal_source_level_array(i,2,1)    = 0.
   ensemble_calibration_data%cal_source_level_array(i,3,1)    = 0.
   ensemble_calibration_data%injection_cal_baseline_array(i)  = 1.
   ensemble_calibration_data%injection_cal_level_array(i)     = 1.
end do

do i = 1, number_of_channels
   do j = 1, number_of_bins
      ensemble_calibration_data%freq_calibration_array(i,j,1) = &
          float(j)*frequency_resolution
      ensemble_calibration_data%freq_calibration_array(i,j,2) = 1.
      ensemble_calibration_data%freq_calibration_array(i,j,3) = 0.
   end do
end do

! DAS data

ensemble_DAS_data%total_channels  = 97
ensemble_DAS_data%adbits          = 24

do i = 1, number_of_channels
   ensemble_DAS_data%coupling_array(i)         = 1
   ensemble_DAS_data%full_scale_range_array(i) = 10000.
end do

ensemble_DAS_data%trigger_mode  = 0
ensemble_DAS_data%clock_mode    = 0
ensemble_DAS_data%sample_count  = 3000000
ensemble_DAS_data%sample_period = 0.005

do i = 1, number_of_channels
   ensemble_DAS_data%lpf_cutoff_freq_array(i)      = 102300.
   ensemble_DAS_data%lpf_pregain_array(i)          = 1.
   ensemble_DAS_data%lpf_postgain_array(i)         = 1.
   ensemble_DAS_data%hpf_cutoff_freq_array(i)      = 400.
   ensemble_DAS_data%hpf_pregain_array(i)          = 1.
   ensemble_DAS_data%hpf_postgain_array(i)         = 1.
   ensemble_DAS_data%external_gain_offset_array(i) = 20.
   ensemble_DAS_data%on_board_LP_filter_array(i)   = 0
end

! Array data

ensemble_array_data%total_array_sensors               = 97
ensemble_array_data%traverse_configuration            = 'Standard configuration'
ensemble_array_data%sideline_traverse_x               = 0.
ensemble_array_data%sideline_traverse_y               = 0.
ensemble_array_data%sideline_traverse_z               = 0.
ensemble_array_data%sideline_traverse_coord_ref       = 'tunnel'
ensemble_array_data%array_traverse_x                  = 0.
ensemble_array_data%array_traverse_y                  = 0.
ensemble_array_data%array_traverse_z                  = 0.
ensemble_array_data%array_traverse_coord_ref          = 'tunnel'
ensemble_array_data%array_elevation                   = 0.
ensemble_array_data%array_azimuth                     = 0.
ensemble_array_data%number_of_inclinometers           = 3
ensemble_array_data%inclinometer_coordinates(1,1)     = 10.
ensemble_array_data%inclinometer_coordinates(1,2)     = 20.
ensemble_array_data%inclinometer_coordinates(1,3)     = 0.
ensemble_array_data%inclinometer_coordinates(2,1)     = 20.
ensemble_array_data%inclinometer_coordinates(2,2)     = 10.
ensemble_array_data%inclinometer_coordinates(2,3)     = 0.
ensemble_array_data%inclinometer_coordinates(3,1)     = -10.
ensemble_array_data%inclinometer_coordinates(3,2)     = -20.
ensemble_array_data%inclinometer_coordinates(3,3)     = 0.
ensemble_array_data%inclinometer_coord_ref            = 'model'
ensemble_array_data%inclinometer_phi_array(1)         = 0.
```

```
ensemble_array_data%inclinometer_phi_array(2)         = 0.
ensemble_array_data%inclinometer_phi_array(3)         = 0.
ensemble_array_data%inclinometer_theta_array(1)       = 0.
ensemble_array_data%inclinometer_theta_array(2)       = 0.
ensemble_array_data%inclinometer_theta_array(3)       = 0.
ensemble_array_data%inclinometer_rotation_array(1)    = 0.
ensemble_array_data%inclinometer_rotation_array(2)    = 0.
ensemble_array_data%inclinometer_rotation_array(3)    = 0.
ensemble_array_data%inclinometer_temperature_array(1) = 65.
ensemble_array_data%inclinometer_temperature_array(2) = 65.
ensemble_array_data%inclinometer_temperature_array(3) = 65.
ensemble_array_data%photogrammetric_x_location        = 0.
ensemble_array_data%photogrammetric_y_location        = 0.
ensemble_array_data%photogrammetric_z_location        = 0.
ensemble_array_data%photogrammetric_yaw_angle         = 0.
ensemble_array_data%photogrammetric_pitch_angle       = 0.
ensemble_array_data%photogrammetric_roll_angle        = 0.
ensemble_array_data%photogrammetric_number_targets    = 20
ensemble_array_data%photogrammetric_RMS_error         = 0.

! Facility data

facility_data%facility_name = '14- by 22-foot Subsonic Tunnel'
facility_data%facility_description = 'Open Jet Configuration'
facility_data%total_facility_parameters = 4

facility_data%facility_parameter(1)%long_name = ‘PA’
facility_data%facility_parameter(1)%description = ‘Ambient Pressure’
facility_data%facility_parameter(1)%data_type = ‘real’
facility_data%facility_parameter(1)%units = ‘psfA’
facility_data%facility_parameter(1)%value = ’2129.52’

facility_data%facility_parameter(2)%long_name = ‘PTOT’
facility_data%facility_parameter(2)%description = ‘Settling Chamber Total Pressure’
facility_data%facility_parameter(2)%data_type = ‘real’
facility_data%facility_parameter(2)%units = ‘psfA’
facility_data%facility_parameter(2)%value = ‘2129.429888’

facility_data%facility_parameter(3)%long_name = ‘TA’
facility_data%facility_parameter(3)%description = ‘Entrance Cone Ambient Temperature’
facility_data%facility_parameter(3)%data_type = ‘real’
facility_data%facility_parameter(3)%units = ‘deg F’
facility_data%facility_parameter(3)%value = ’57.315520’

facility_data%facility_parameter(4)%long_name = ‘TDEW’
facility_data%facility_parameter(4)%description = ‘Dew Point Temperature’
facility_data%facility_parameter(4)%data_type = ‘real’
facility_data%facility_parameter(4)%units = ‘deg F’
facility_data%facility_parameter(4)%value = ’29.393310’

do i = 5, 200
   facility_data%facility_parameter(i)%long_name = ‘’
   facility_data%facility_parameter(i)%description = ‘’
   facility_data%facility_parameter(i)%data_type = ‘’
   facility_data%facility_parameter(i)%units = ‘’
   facility_data%facility_parameter(i)%value = ‘’
end do

! Model data

model_data%model_id = 1
model_data%model_description = 'HL-CRM Model'
model_data%total_model_parameters = 2

model_data%model_parameter(1)%long_name = ‘PITCHM’
model_data%model_parameter(1)%description = ‘Mast Pitch Angle’
model_data%model_parameter(1)%data_type = ‘real’
model_data%model_parameter(1)%units = ‘deg’
model_data%model_parameter(1)%value = ‘0.000000’
```

```
model_data%model_parameter(2)%long_name = ‘ALPHA’
model_data%model_parameter(2)%description = ‘Model Angle of Attack’
model_data%model_parameter(2)%data_type = ‘real’
model_data%model_parameter(2)%units = ‘deg’
model_data%model_parameter(2)%value = ‘6.999002’

do i = 3, 200
   model_data%model_parameter(i)%long_name = ‘’
   model_data%model_parameter(i)%description = ‘’
   model_data%model_parameter(i)%data_type = ‘’
   model_data%model_parameter(i)%units = ‘’
   model_data%model_parameter(i)%value = ‘’
end do

return
```

Appendix E.3 - Populating Header Values in Pressure Data Files
Fortran Code Fragment

```
! Version data

version_data%netcdf_version = 3.4 !Note: Do not change these values
version_data%hdf5_version   = 1.8
version_data%hdr_version    = 9.4

! Test data

test_data%testno            = 643
test_data%runno             = 259
test_data%pointno           = 3541
test_data%GPS_start_time    = 28800.
test_data%GPS_stop_time     = 28830.
test_data%date_time         = '12/20/2018 08:00 '
test_data%test_engineer     = 'William Humphreys'

! Facility data

facility_data%facility_name = '14- by 22-foot Subsonic Tunnel'
facility_data%facility_description = 'Open Jet Configuration'
facility_data%total_facility_parameters = 4

facility_data%facility_parameter(1)%long_name = 'PA'
facility_data%facility_parameter(1)%description = 'Ambient Pressure'
facility_data%facility_parameter(1)%data_type = 'real'
facility_data%facility_parameter(1)%units = 'psfA'
facility_data%facility_parameter(1)%value = '2129.52'

facility_data%facility_parameter(2)%long_name = 'PTOT'
facility_data%facility_parameter(2)%description = 'Settling Chamber Total Pressure'
facility_data%facility_parameter(2)%data_type = 'real'
facility_data%facility_parameter(2)%units = 'psfA'
facility_data%facility_parameter(2)%value = '2129.429888'

facility_data%facility_parameter(3)%long_name = 'TA'
facility_data%facility_parameter(3)%description = 'Entrance Cone Ambient Temperature'
facility_data%facility_parameter(3)%data_type = 'real'
facility_data%facility_parameter(3)%units = 'deg F'
facility_data%facility_parameter(3)%value = '57.315520'

facility_data%facility_parameter(4)%long_name = 'TDEW'
facility_data%facility_parameter(4)%description = 'Dew Point Temperature'
facility_data%facility_parameter(4)%data_type = 'real'
facility_data%facility_parameter(4)%units = 'deg F'
facility_data%facility_parameter(4)%value = '29.393310'

do i = 5, 200
   facility_data%facility_parameter(i)%long_name = ''
   facility_data%facility_parameter(i)%description = ''
   facility_data%facility_parameter(i)%data_type = ''
   facility_data%facility_parameter(i)%units = ''
   facility_data%facility_parameter(i)%value = ''
end do

! Model data

model_data%model_id = 1
model_data%model_description = 'HL-CRM Model'
model_data%total_model_parameters = 2

model_data%model_parameter(1)%long_name = 'PITCHM'
model_data%model_parameter(1)%description = 'Mast Pitch Angle'
model_data%model_parameter(1)%data_type = 'real'
model_data%model_parameter(1)%units = 'deg'
model_data%model_parameter(1)%value = '0.000000'
```

```
model_data%model_parameter(2)%long_name = ‘ALPHA’
model_data%model_parameter(2)%description = ‘Model Angle of Attack’
model_data%model_parameter(2)%data_type = ‘real’
model_data%model_parameter(2)%units = ‘deg’
model_data%model_parameter(2)%value = ‘6.999002’

do i = 3, 200
   model_data%model_parameter(i)%long_name = ‘’
   model_data%model_parameter(i)%description = ‘’
   model_data%model_parameter(i)%data_type = ‘’
   model_data%model_parameter(i)%units = ‘’
   model_data%model_parameter(i)%value = ‘’
end do

! Static tap data

static_data%total_static_taps = 3

static_data%static_tap_parameter(1)%static_tap_designation = ‘ESP0101’
static_data%static_tap_parameter(1)%description = ‘PSI Pressure Tap’
static_data%static_tap_parameter(1)%units = ‘psi’
static_data%static_tap_parameter(1)%coefficient_name = ‘ESP0101’
static_data%static_tap_parameter(1)%data_value = ‘0.109163’

static_data%static_tap_parameter(2)%static_tap_designation = ‘ESP0102’
static_data%static_tap_parameter(2)%description = ‘PSI Pressure Tap’
static_data%static_tap_parameter(2)%units = ‘psi’
static_data%static_tap_parameter(2)%coefficient_name = ‘ESP0102’
static_data%static_tap_parameter(2)%data_value = ‘0.052045’

static_data%static_tap_parameter(3)%static_tap_designation = ‘ESP0103’
static_data%static_tap_parameter(3)%description = ‘PSI Pressure Tap’
static_data%static_tap_parameter(3)%units = ‘psi’
static_data%static_tap_parameter(3)%coefficient_name = ‘ESP0103’
static_data%static_tap_parameter(3)%data_value = ‘-0.058525’

do i = 4, 999
  static_data%static_tap_parameter(i)%static_tap_designation = ‘’
  static_data%static_tap_parameter(i)%sdescription = ‘’
  static_data%static_tap_parameter(i)%units = ‘’
  static_data%static_tap_parameter(i)%coefficient_name = ‘’
  static_data%static_tap_parameter(i)%data_values = ‘’
end do

return
```

Appendix E.4 - Populating Header Values in Cross Spectral Matrix Data Files
Fortran Code Fragment

```
! Version data

csm_version_data%netcdf_version = 3.4                !Note: Do not change these values
csm_version_data%hdf5_version   = 1.8                !
csm_version_data%hdr_version    = 9.4                !

! Test data

csm_test_data%testno            = 643
csm_test_data%runno             = 259
csm_test_data%pointno           = 3541
csm_test_data%date_time         = '12/20/2018 08:00 '
csm_test_data%GPS_start_time    = 28800.
csm_test_data%GPS_stop_time     = 28830.

! Channel data

do i = 1, number_of_channels
   csm_channel_data%sensor_sn_array(i)          = mic_sn(i)
   csm_channel_data%preamplifier_sn_array(i)    = preamp_sn(i)
   csm_channel_data%sensor_type_array(i)        = 1
   csm_channel_data%preamplifier_type_array(i)  = 1
   csm_channel_data%excitation_voltage_array(i) = 200.
   csm_channel_data%excitation_current_array(i) = 0.
end do

do i = 1, number_of_channels
   csm_channel_data%sensor_coord_array(i,1)     = sensor_x(i)
   csm_channel_data%sensor_coord_array(i,2)     = sensor_y(i)
   csm_channel_data%sensor_coord_array(i,3)     = sensor_z(i)
end do

csm_channel_data%array_sensor_coord_ref    = ‘array’
csm_channel_data%sideline_sensor_coord_ref = ‘tunnel’
csm_channel_data%kulite_sensor_coord_ref   = ‘model’

! Calibration data

do i = 1, number_of_channels
  csm_calibration_data%sensitivity_array(i) = 1500.
end do

do i = 1, number_of_bins
  csm_calibration_data%freq_calibration_array(i,1) = &
      float(i-1)*frequency_resolution
  csm_calibration_data%freq_calibration_array(i,2) = 1.
  csm_calibration_data%freq_calibration_array(i,3) = 0.
  csm_calibration_data%filter_response(i,1) = float(i-1)*frequency_resolution
  csm_calibration_data%filter_response(i,2) = 1.
  csm_calibration_data%filter_response(i,3) = 0.
end do

csm_calibration_data%number_of_cal_sources           = 3
csm_calibration_data%cal_source_coordinates(1,1)     = 114.34
csm_calibration_data%cal_source_coordinates(1,2)     = 22.85
csm_calibration_data%cal_source_coordinates(1,3)     = -4.79
csm_calibration_data%cal_source_coordinates(2,1)     = 137.56
csm_calibration_data%cal_source_coordinates(2,2)     = 58.59
csm_calibration_data%cal_source_coordinates(2,3)     = 0.99
csm_calibration_data%cal_source_coordinates(3,1)     = 164.55
csm_calibration_data%cal_source_coordinates(3,2)     = 99.85
csm_calibration_data%cal_source_coordinates(3,3)     = 5.07
csm_calibration_data%cal_source_coord_ref_array(1,:) = ‘model'
csm_calibration_data%cal_source_coord_ref_array(2,:) = ‘model'
csm_calibration_data%cal_source_coord_ref_array(3,:) = ‘model'

csm_calibration_data%number_of_cal_measurements = 1
```

```
csm_calibration_data%date_time_of_cal(1,1) = '12/20/2018 08:00 '
csm_calibration_data%date_time_of_cal(2,1) = '12/20/2018 08:00 '
csm_calibration_data%date_time_of_cal(3,1) = '12/20/2018 08:00 '

do i = 1, number_of_channels
   csm_calibration_data%cal_source_baseline_array(i,1,1) = 0.
   csm_calibration_data%cal_source_baseline_array(i,2,1) = 0.
   csm_calibration_data%cal_source_baseline_array(i,3,1) = 0.
   csm_calibration_data%cal_source_level_array(i,1,1)    = 0.
   csm_calibration_data%cal_source_level_array(i,2,1)    = 0.
   csm_calibration_data%cal_source_level_array(i,3,1)    = 0.
   csm_calibration_data%injection_cal_baseline_array(i)  = 1.
   csm_calibration_data%injection_cal_level_array(i)     = 1.
end do

csm_calibration_data%sensitivity_correction_switch = 0
csm_calibration_data%temperature_correction_switch = 0
csm_calibration_data%db_correction_switch          = 1

! DAS data

csm_DAS_data%total_channels  = 97
csm_DAS_data%sample_count    = 3000000
csm_DAS_data%sample_period   = 0.005

do i = 1, number_of_channels
   csm_DAS_data%full_scale_range_array(i) = 10000.
end do

do i = 1, number_of_channels
   csm_DAS_data%lpf_cutoff_freq_array(i)      = 102300.
   csm_DAS_data%lpf_pregain_array(i)          = 1.
   csm_DAS_data%lpf_postgain_array(i)         = 1.
   csm_DAS_data%hpf_cutoff_freq_array(i)      = 400.
   csm_DAS_data%hpf_pregain_array(i)          = 1.
   csm_DAS_data%hpf_postgain_array(i)         = 1.
   csm_DAS_data%external_gain_offset_array(i) = 20.
end do

! Array data

csm_array_data%total_array_sensors               = 97
csm_array_data%traverse_configuration            = 'Standard configuration'
csm_array_data%sideline_traverse_x               = 0.
csm_array_data%sideline_traverse_y               = 0.
csm_array_data%sideline_traverse_z               = 0.
csm_array_data%sideline_traverse_coord_ref       = 'tunnel'
csm_array_data%array_traverse_x                  = 0.
csm_array_data%array_traverse_y                  = 0.
csm_array_data%array_traverse_z                  = 0.
csm_array_data%array_traverse_coord_ref          = 'tunnel'
csm_array_data%array_elevation                   = 0.
csm_array_data%array_azimuth                     = 0.
csm_array_data%number_of_inclinometers           = 3
csm_array_data%inclinometer_coordinates(1,1)     = 10.
csm_array_data%inclinometer_coordinates(1,2)     = 20.
csm_array_data%inclinometer_coordinates(1,3)     = 0.
csm_array_data%inclinometer_coordinates(2,1)     = 20.
csm_array_data%inclinometer_coordinates(2,2)     = 10.
csm_array_data%inclinometer_coordinates(2,3)     = 0.
csm_array_data%inclinometer_coordinates(3,1)     = -10.
csm_array_data%inclinometer_coordinates(3,2)     = -20.
csm_array_data%inclinometer_coordinates(3,3)     = 0.
csm_array_data%inclinometer_coord_ref            = 'model'
csm_array_data%inclinometer_phi_array(1)         = 0.
csm_array_data%inclinometer_phi_array(2)         = 0.
csm_array_data%inclinometer_phi_array(3)         = 0.
csm_array_data%inclinometer_theta_array(1)       = 0.
csm_array_data%inclinometer_theta_array(2)       = 0.
csm_array_data%inclinometer_theta_array(3)       = 0.
csm_array_data%inclinometer_rotation_array(1)    = 0.
csm_array_data%inclinometer_rotation_array(2)    = 0.
```

```
csm_array_data%inclinometer_rotation_array(3)    = 0.
csm_array_data%inclinometer_temperature_array(1) = 65.
csm_array_data%inclinometer_temperature_array(2) = 65.
csm_array_data%inclinometer_temperature_array(3) = 65.
csm_array_data%photogrammetric_x_location        = 0.
csm_array_data%photogrammetric_y_location        = 0.
csm_array_data%photogrammetric_z_location        = 0.
csm_array_data%photogrammetric_yaw_angle         = 0.
csm_array_data%photogrammetric_pitch_angle       = 0.
csm_array_data%photogrammetric_roll_angle        = 0.
csm_array_data%photogrammetric_number_targets    = 20
csm_array_data%photogrammetric_RMS_error         = 0.

! Facility data

facility_data%facility_name             = '14- by 22-foot Subsonic Tunnel'
facility_data%facility_description      = 'Open Jet Configuration'
facility_data%total_facility_parameters = 4

facility_data%facility_parameter(1).long_name = ‘PA’
facility_data%facility_parameter(1).description = ‘Ambient Pressure’
facility_data%facility_parameter(1).data_type = ‘real’
facility_data%facility_parameter(1).units = ‘psfA’
facility_data%facility_parameter(1).value = ’2129.52’

facility_data%facility_parameter(2).long_name = ‘PTOT’
facility_data%facility_parameter(2).description = ‘Settling Chamber Total Pressure’
facility_data%facility_parameter(2).data_type = ‘real’
facility_data%facility_parameter(2).units = ‘psfA’
facility_data%facility_parameter(2).value = ‘2129.429888’

facility_data%facility_parameter(3).long_name = ‘TA’
facility_data%facility_parameter(3).description = ‘Entrance Cone Ambient Temperature’
facility_data%facility_parameter(3).data_type = ‘real’
facility_data%facility_parameter(3).units = ‘deg F’
facility_data%facility_parameter(3).value = ’57.315520’

facility_data%facility_parameter(4).long_name = ‘TDEW’
facility_data%facility_parameter(4).description = ‘Dew Point Temperature’
facility_data%facility_parameter(4).data_type = ‘real’
facility_data%facility_parameter(4).units = ‘deg F’
facility_data%facility_parameter(4).value = ’29.393310’

do i = 5, 200
   facility_data%facility_parameter(i).long_name = ‘’
   facility_data%facility_parameter(i).description = ‘’
   facility_data%facility_parameter(i).data_type = ‘’
   facility_data%facility_parameter(i).units = ‘’
   facility_data%facility_parameter(i).value = ‘’
end do

! Model data

model_data%model_id               = 1
model_data%model_description      = ‘HL-CRM Model'
model_data%total_model_parameters = 2

model_data%model_parameter(1).long_name = ‘PITCHM’
model_data%model_parameter(1).description = ‘Mast Pitch Angle’
model_data%model_parameter(1).data_type = ‘real’
model_data%model_parameter(1).units = ‘deg’
model_data%model_parameter(1).value = ‘0.000000’

model_data%model_parameter(2).long_name = ‘ALPHA’
model_data%model_parameter(2).description = ‘Model Angle of Attack’
model_data%model_parameter(2).data_type = ‘real’
model_data%model_parameter(2).units = ‘deg’
model_data%model_parameter(2).value = ‘6.999002’
```

```
do i = 3, 200
   model_data%model_parameter(i).long_name = ‘’
   model_data%model_parameter(i).description = ‘’
   model_data%model_parameter(i).data_type = ‘’
   model_data%model_parameter(i).units = ‘’
   model_data%model_parameter(i).value = ‘’
end do

! CSM data

csm_data%fft_length      = 8192
csm_data%overlap         = 0
csm_data%num_blocks      = 500
csm_data%num_blocks_used = 500

do i = 1, number_of_blocks
   do j = 1, number_of_channels
       block_table(i,j) = 1
   end do
end do

csm_data%cslength  = 4096
csm_data%winfunc   = ‘hamming’
csm_data%beta      = 0.
csm_data%CSM_units = 1

return
```

Appendix F.1 - Populating Fixed Header Values in Individual Raw Data Files
Matlab Code Fragment

```
%% Version data

version_data.netcdf_version = single(3.4);    %Note: Do not change the version values
version_data.hdf5_version   = single(1.8);
version_data.hdr_version    = single(9.4);

%% Test data

test_data.testno         = int32(643);
test_data.runno          = int32(259);
test_data.pointno        = int32(3541);
test_data.date_time      = '12/20/2018 08:00 ';
test_data.GPS_start_time = single(28800.0);
test_data.GPS_stop_time  = single(28830.0);
test_data.test_engineer  = 'William Humphreys';

%% Channel data

channel_data.channo                   = int32(1);
channel_data.nam_chan                 = 'Channel 001';
channel_data.chan_description         = 'Array Channel';
channel_data.sensor_sn                = int32(2688098);
channel_data.preamplifier_sn          = int32(2688519);
channel_data.sensor_type              = int32(1);
channel_data.preamplifier_type        = int32(1);
channel_data.sensor_description       = 'B&K 4938 Pressure Microphone';
channel_data.preamplifier_description = 'B&K 2670 1/4-inch Preamplifier';
channel_data.excitation_voltage       = single(200.0);
channel_data.excitation_current       = single(0.0);
channel_data.sensor_x                 = single(-0.041);
channel_data.sensor_y                 = single(0.999);
channel_data.sensor_z                 = single(0.0);
channel_data.sensor_coord_ref         = 'ARRAY';

%% Calibration data

calibration_data.sensitivity                  = single(63.0);
calibration_data.number_of_cal_sources        = int32(3);
calibration_data.cal_source_coordinates(1,1)  = single(114.34);
calibration_data.cal_source_coordinates(1,2)  = single(22.85);
calibration_data.cal_source_coordinates(1,3)  = single(-4.79);
calibration_data.cal_source_coordinates(2,1)  = single(137.56);
calibration_data.cal_source_coordinates(2,2)  = single(58.59);
calibration_data.cal_source_coordinates(2,3)  = single(0.99);
calibration_data.cal_source_coordinates(3,1)  = single(164.55);
calibration_data.cal_source_coordinates(3,2)  = single(99.85);
calibration_data.cal_source_coordinates(3,3)  = single(5.07);
calibration_data.cal_source_coord_ref         = ‘MODEL';
calibration_data.cal_source_baseline_array(1) = single(0.0);
calibration_data.cal_source_baseline_array(2) = single(0.0);
calibration_data.cal_source_baseline_array(3) = single(0.0);
calibration_data.cal_source_level_array(1)    = single(0.0);
calibration_data.cal_source_level_array(2)    = single(0.0);
calibration_data.cal_source_level_array(3)    = single(0.0);
calibration_data.injection_cal_baseline       = single(1.0);
calibration_data.injection_cal_level          = single(1.0);
for i = 1:number_of_bins
    calibration_data.calibration_frequency(i) = single((i-1)*frequency_resolution);
    calibration_data.calibration_magnitude(i) = single(1.0);
    calibration_data.calibration_phase(i)     = single(0.0);
end
```

```
%% DAS data

DAS_data.total_channels       = int32(97);
DAS_data.adbits               = int32(24);
DAS_data.coupling             = int32(1);
DAS_data.full_scale_range     = single(10000.0);
DAS_data.trigger_mode         = int32(0);
DAS_data.clock_mode           = int32(0);
DAS_data.sample_count         = int32(3000000);
DAS_data.sample_period        = single(0.005);
DAS_data.lpf_cutoff_freq      = single(102300.0);
DAS_data.lpf_pregain          = single(1.0);
DAS_data.lpf_postgain         = single(1.0);
DAS_data.hpf_cutoff_freq      = single(400.0);
DAS_data.hpf_pregain          = single(1.0);
DAS_data.hpf_postgain         = single(1.0);
DAS_data.external_gain_offset = single(20.0);
DAS_data.on_board_LP_filter   = int32(0);

%% Array data

array_data.total_array_sensors                = int32(97);
array_data.traverse_configuration             = 'Standard configuration';
array_data.sideline_traverse_x                = single(0.0);
array_data.sideline_traverse_y                = single(0.0);
array_data.sideline_traverse_z                = single(0.0);
array_data.sideline_traverse_coord_ref        = 'TUNNEL';
array_data.array_traverse_x                   = single(0.0);
array_data.array_traverse_y                   = single(0.0);
array_data.array_traverse_z                   = single(0.0);
array_data.array_traverse_coord_ref           = 'TUNNEL';
array_data.array_elevation                    = single(0.0);
array_data.array_azimuth                      = single(0.0);
array_data.number_of_inclinometers            = int32(3);
array_data.inclinometer_coordinates(1,1)      = single(10.0);
array_data.inclinometer_coordinates(1,2)      = single(20.0);
array_data.inclinometer_coordinates(1,3)      = single(0.0);
array_data.inclinometer_coordinates(2,1)      = single(20.0);
array_data.inclinometer_coordinates(2,2)      = single(10.0);
array_data.inclinometer_coordinates(2,3)      = single(0.0);
array_data.inclinometer_coordinates(3,1)      = single(-10.0);
array_data.inclinometer_coordinates(3,2)      = single(-20.0);
array_data.inclinometer_coordinates(3,3)      = single(0.0);
array_data.inclinometer_coord_ref             = 'MODEL';
array_data.inclinometer_phi_array(1)          = single(0.0);
array_data.inclinometer_phi_array(2)          = single(0.0);
array_data.inclinometer_phi_array(3)          = single(0.0);
array_data.inclinometer_theta_array(1)        = single(0.0);
array_data.inclinometer_theta_array(2)        = single(0.0);
array_data.inclinometer_theta_array(3)        = single(0.0);
array_data.inclinometer_rotation_array(1)     = single(0.0);
array_data.inclinometer_rotation_array(2)     = single(0.0);
array_data.inclinometer_rotation_array(3)     = single(0.0);
array_data.inclinometer_temperature_array(1) = single(65.0);
array_data.inclinometer_temperature_array(2) = single(65.0);
array_data.inclinometer_temperature_array(3) = single(65.0);
array_data.photogrammetric_x_location         = single(0.0);
array_data.photogrammetric_y_location         = single(0.0);
array_data.photogrammetric_z_location         = single(0.0);
array_data.photogrammetric_yaw_angle          = single(0.0);
array_data.photogrammetric_pitch_angle        = single(0.0);
array_data.photogrammetric_roll_angle         = single(0.0);
array_data.photogrammetric_number_targets     = int32(20);
array_data.photogrammetric_RMS_error          = single(0.0);
```

```
%% Facility data

facility_data.facility_name = strcpy_and_pad('14- by 22-foot Subsonic Tunnel',80);
facility_data.facility_description = strcpy_and_pad('Open Jet Configuration',80);
facility_data.total_facility_parameters = int32(4);

facility_data.facility_parameter(1).long_name = 'PA';
facility_data.facility_parameter(1).description = 'Ambient Pressure';
facility_data.facility_parameter(1).data_type = 'real';
facility_data.facility_parameter(1).units = 'psfA';
facility_data.facility_parameter(1).value = '2129.52';

facility_data.facility_parameter(2).long_name = 'PTOT';
facility_data.facility_parameter(2).description = 'Settling Chamber Total Pressure';
facility_data.facility_parameter(2).data_type = 'real';
facility_data.facility_parameter(2).units = 'psfA';
facility_data.facility_parameter(2).value = '2129.429888';

facility_data.facility_parameter(3).long_name = 'TA';
facility_data.facility_parameter(3).description = 'Entrance Cone Ambient Temperature';
facility_data.facility_parameter(3).data_type = 'real';
facility_data.facility_parameter(3).units = 'deg F';
facility_data.facility_parameter(3).value = '57.315520';

facility_data.facility_parameter(4).long_name = 'TDEW';
facility_data.facility_parameter(4).description = 'Dew Point Temperature';
facility_data.facility_parameter(4).data_type = 'real';
facility_data.facility_parameter(4).units = 'deg F';
facility_data.facility_parameter(4).value = '29.393310';

for i = 5:200
   facility_data.facility_parameter(i).long_name = '';
   facility_data.facility_parameter(i).description = '';
   facility_data.facility_parameter(i).data_type = '';
   facility_data.facility_parameter(i).units = '';
   facility_data.facility_parameter(i).value = '';
end

%% Model data

model_data.model_id = int32(1);
model_data.model_description = strcpy_and_pad('HL-CRM Model',80);
model_data.total_model_parameters = int32(2);

model_data.model_parameter(1).long_name = 'PITCHM';
model_data.model_parameter(1).description = 'Mast Pitch Angle';
model_data.model_parameter(1).data_type = 'real';
model_data.model_parameter(1).units = 'deg';
model_data.model_parameter(1).value = '0.000000';

model_data.model_parameter(2).long_name = 'ALPHA';
model_data.model_parameter(2).description = 'Model Angle of Attack';
model_data.model_parameter(2).data_type = 'real';
model_data.model_parameter(2).units = 'deg';
model_data.model_parameter(2).value = '6.999002';

for i = 3:200
   model_data.model_parameter(i).long_name = '';
   model_data.model_parameter(i).description = '';
   model_data.model_parameter(i).data_type = '';
   model_data.model_parameter(i).units = '';
   model_data.model_parameter(i).value = '';
end

return;
```

Appendix F.2 - Populating Header Values in Ensemble Raw Data Files
Matlab Code Fragment

```
%% Version data

ensemble_version_data.netcdf_version = single(3.4);  %Note: Do not change these values
ensemble_version_data.hdf5_version   = single(1.8);
ensemble_version_data.hdr_version    = single(9.4);

%% Test data

ensemble_test_data.testno            = int32(643);
ensemble_test_data.runno             = int32(259);
ensemble_test_data.pointno           = int32(3541);
ensemble_test_data.identifier_prefix = 'X';
ensemble_test_data.date_time         = '12/20/2018 08:00 ';
ensemble_test_data.GPS_start_time    = single(28800.0);
ensemble_test_data.GPS_stop_time     = single(28830.0);
ensemble_test_data.test_engineer     = 'William Humphreys';

%% Channel data

for i = 1:number_of_channels
   ensemble_channel_data.channo_array(i)             = int32(i);
   ensemble_channel_data.sensor_sn_array(i)          = int32(mic_sn(i));
   ensemble_channel_data.preamplifier_sn_array(i)    = int32(preamp_sn(i));
   ensemble_channel_data.sensor_type_array(i)        = int32(1);
   ensemble_channel_data.preamplifier_type_array(i)  = int32(1);
   ensemble_channel_data.excitation_voltage_array(i) = single(200.0);
   ensemble_channel_data.excitation_current_array(i) = single(0.0);
end

for i = 1:number_of_channels
   ensemble_channel_data.sensor_coord_array(i,1)     = single(sensor_x(i));
   ensemble_channel_data.sensor_coord_array(i,2)     = single(sensor_y(i));
   ensemble_channel_data.sensor_coord_array(i,3)     = single(sensor_z(i));
   ensemble_channel_data.sensor_coord_ref_array(i,:) = 'array';
end

%% Calibration data

for i = 1:number_of_channels
  ensemble_calibration_data.sensitivity_array(i) = single(1500.0);
end

for i = 1:number_of_bins
  ensemble_calibration_data.filter_response(i,1) = single((i-1)*frequency_resolution);
  ensemble_calibration_data.filter_response(i,2) = single(1.0);
  ensemble_calibration_data.filter_response(i,3) = single(0.0);
end

ensemble_calibration_data.number_of_cal_sources          = int32(3);
ensemble_calibration_data.cal_source_coordinates(1,1)    = single(114.34);
ensemble_calibration_data.cal_source_coordinates(1,2)    = single(22.85);
ensemble_calibration_data.cal_source_coordinates(1,3)    = single(-4.79);
ensemble_calibration_data.cal_source_coordinates(2,1)    = single(137.56);
ensemble_calibration_data.cal_source_coordinates(2,2)    = single(58.59);
ensemble_calibration_data.cal_source_coordinates(2,3)    = single(0.99);
ensemble_calibration_data.cal_source_coordinates(3,1)    = single(164.55);
ensemble_calibration_data.cal_source_coordinates(3,2)    = single(99.85);
ensemble_calibration_data.cal_source_coordinates(3,3)    = single(5.07);
ensemble_calibration_data.cal_source_coord_ref_array(1,:) = 'model';
ensemble_calibration_data.cal_source_coord_ref_array(2,:) = 'model';
ensemble_calibration_data.cal_source_coord_ref_array(3,:) = 'model';

ensemble_calibration_data.number_of_cal_measurements = int32(1);
ensemble_calibration_data.date_time_of_cal(1,1,:)    = '12/20/2018 08:00 ';
ensemble_calibration_data.date_time_of_cal(2,1,:)    = '12/20/2018 08:00 ';
ensemble_calibration_data.date_time_of_cal(3,1,:)    = '12/20/2018 08:00 ';
```

```
for i = 1:number_of_channels
   ensemble_calibration_data.cal_source_baseline_array(i,1,1) = single(0.0);
   ensemble_calibration_data.cal_source_baseline_array(i,2,1) = single(0.0);
   ensemble_calibration_data.cal_source_baseline_array(i,3,1) = single(0.0);
   ensemble_calibration_data.cal_source_level_array(i,1,1)    = single(0.0);
   ensemble_calibration_data.cal_source_level_array(i,2,1)    = single(0.0);
   ensemble_calibration_data.cal_source_level_array(i,3,1)    = single(0.0);
   ensemble_calibration_data.injection_cal_baseline_array(i)  = single(1.0);
   ensemble_calibration_data.injection_cal_level_array(i)     = single(1.0);
end

for i = 1:number_of_channels
   for j = 1:number_of_bins
      ensemble_calibration_data.freq_calibration_array(i,j,1) = ...
          single((j-1)*frequency_resolution);
      ensemble_calibration_data.freq_calibration_array(i,j,2) = single(1.0);
      ensemble_calibration_data.freq_calibration_array(i,j,3) = single(0.0);
   end
end

%% DAS data

ensemble_DAS_data.total_channels  = int32(97);
ensemble_DAS_data.adbits          = int32(24);

for i = 1:number_of_channels
   ensemble_DAS_data.coupling_array(i)         = int32(1);
   ensemble_DAS_data.full_scale_range_array(i) = single(10000.0);
end

ensemble_DAS_data.trigger_mode  = int32(0);
ensemble_DAS_data.clock_mode    = int32(0);
ensemble_DAS_data.sample_count  = int32(3000000);
ensemble_DAS_data.sample_period = single(0.005);

for i = 1:number_of_channels
   ensemble_DAS_data.lpf_cutoff_freq_array(i)      = single(102300.0);
   ensemble_DAS_data.lpf_pregain_array(i)          = single(1.0);
   ensemble_DAS_data.lpf_postgain_array(i)         = single(1.0);
   ensemble_DAS_data.hpf_cutoff_freq_array(i)      = single(400.0);
   ensemble_DAS_data.hpf_pregain_array(i)          = single(1.0);
   ensemble_DAS_data.hpf_postgain_array(i)         = single(1.0);
   ensemble_DAS_data.external_gain_offset_array(i) = single(20.0);
   ensemble_DAS_data.on_board_LP_filter_array(i)   = int32(0);
end

%% Array data

ensemble_array_data.total_array_sensors               = int32(97);
ensemble_array_data.traverse_configuration            = 'Standard configuration';
ensemble_array_data.sideline_traverse_x               = single(0.0);
ensemble_array_data.sideline_traverse_y               = single(0.0);
ensemble_array_data.sideline_traverse_z               = single(0.0);
ensemble_array_data.sideline_traverse_coord_ref       = 'tunnel';
ensemble_array_data.array_traverse_x                  = single(0.0);
ensemble_array_data.array_traverse_y                  = single(0.0);
ensemble_array_data.array_traverse_z                  = single(0.0);
ensemble_array_data.array_traverse_coord_ref          = 'tunnel';
ensemble_array_data.array_elevation                   = single(0.0);
ensemble_array_data.array_azimuth                     = single(0.0);
ensemble_array_data.number_of_inclinometers           = int32(3);
ensemble_array_data.inclinometer_coordinates(1,1)     = single(10.0);
ensemble_array_data.inclinometer_coordinates(1,2)     = single(20.0);
ensemble_array_data.inclinometer_coordinates(1,3)     = single(0.0);
ensemble_array_data.inclinometer_coordinates(2,1)     = single(20.0);
ensemble_array_data.inclinometer_coordinates(2,2)     = single(10.0);
ensemble_array_data.inclinometer_coordinates(2,3)     = single(0.0);
ensemble_array_data.inclinometer_coordinates(3,1)     = single(-10.0);
ensemble_array_data.inclinometer_coordinates(3,2)     = single(-20.0);
ensemble_array_data.inclinometer_coordinates(3,3)     = single(0.0);
ensemble_array_data.inclinometer_coord_ref            = 'model';
ensemble_array_data.inclinometer_phi_array(1)         = single(0.0);
```

```
ensemble_array_data.inclinometer_phi_array(2)         = single(0.0);
ensemble_array_data.inclinometer_phi_array(3)         = single(0.0);
ensemble_array_data.inclinometer_theta_array(1)       = single(0.0);
ensemble_array_data.inclinometer_theta_array(2)       = single(0.0);
ensemble_array_data.inclinometer_theta_array(3)       = single(0.0);
ensemble_array_data.inclinometer_rotation_array(1)    = single(0.0);
ensemble_array_data.inclinometer_rotation_array(2)    = single(0.0);
ensemble_array_data.inclinometer_rotation_array(3)    = single(0.0);
ensemble_array_data.inclinometer_temperature_array(1) = single(65.0);
ensemble_array_data.inclinometer_temperature_array(2) = single(65.0);
ensemble_array_data.inclinometer_temperature_array(3) = single(65.0);
ensemble_array_data.photogrammetric_x_location        = single(0.0);
ensemble_array_data.photogrammetric_y_location        = single(0.0);
ensemble_array_data.photogrammetric_z_location        = single(0.0);
ensemble_array_data.photogrammetric_yaw_angle         = single(0.0);
ensemble_array_data.photogrammetric_pitch_angle       = single(0.0);
ensemble_array_data.photogrammetric_roll_angle        = single(0.0);
ensemble_array_data.photogrammetric_number_targets    = int32(20);
ensemble_array_data.photogrammetric_RMS_error         = single(0.0);

%% Facility data

facility_data.facility_name = strcpy_and_pad('14- by 22-foot Subsonic Tunnel',80);
facility_data.facility_description = strcpy_and_pad('Open Jet Configuration',80);
facility_data.total_facility_parameters = int32(4);

facility_data.facility_parameter(1).long_name = 'PA';
facility_data.facility_parameter(1).description = 'Ambient Pressure';
facility_data.facility_parameter(1).data_type = 'real';
facility_data.facility_parameter(1).units = 'psfA';
facility_data.facility_parameter(1).value = '2129.52';

facility_data.facility_parameter(2).long_name = 'PTOT';
facility_data.facility_parameter(2).description = 'Settling Chamber Total Pressure';
facility_data.facility_parameter(2).data_type = 'real';
facility_data.facility_parameter(2).units = 'psfA';
facility_data.facility_parameter(2).value = '2129.429888';

facility_data.facility_parameter(3).long_name = 'TA';
facility_data.facility_parameter(3).description = 'Entrance Cone Ambient Temperature';
facility_data.facility_parameter(3).data_type = 'real';
facility_data.facility_parameter(3).units = 'deg F';
facility_data.facility_parameter(3).value = '57.315520';

facility_data.facility_parameter(4).long_name = 'TDEW';
facility_data.facility_parameter(4).description = 'Dew Point Temperature';
facility_data.facility_parameter(4).data_type = 'real';
facility_data.facility_parameter(4).units = 'deg F';
facility_data.facility_parameter(4).value = '29.393310';

for i = 5:200
   facility_data.facility_parameter(i).long_name = '';
   facility_data.facility_parameter(i).description = '';
   facility_data.facility_parameter(i).data_type = '';
   facility_data.facility_parameter(i).units = '';
   facility_data.facility_parameter(i).value = '';
end

%% Model data

model_data.model_id = int32(1);
model_data.model_description = strcpy_and_pad('HL-CRM Model',80);
model_data.total_model_parameters = int32(2);

model_data.model_parameter(1).long_name = 'PITCHM';
model_data.model_parameter(1).description = 'Mast Pitch Angle';
model_data.model_parameter(1).data_type = 'real';
model_data.model_parameter(1).units = 'deg';
model_data.model_parameter(1).value = '0.000000';

model_data.model_parameter(2).long_name = 'ALPHA';
model_data.model_parameter(2).description = 'Model Angle of Attack';
```

```
model_data.model_parameter(2).data_type = ‘real’;
model_data.model_parameter(2).units = ‘deg’;
model_data.model_parameter(2).value = ‘6.999002’;

for i = 3:200
   model_data.model_parameter(i).long_name = ‘’;
   model_data.model_parameter(i).description = ‘’;
   model_data.model_parameter(i).data_type = ‘’;
   model_data.model_parameter(i).units = ‘’;
   model_data.model_parameter(i).value = ‘’;
end

return;
```

Appendix F.3 - Populating Header Values in Pressure Data Files
Matlab Code Fragment

```
%% Version data

version_data.netcdf_version = single(3.4);     %Note: Do not change these values
version_data.hdf5_version   = single(1.8);
version_data.hdr_version    = single(9.4);

%% Test data

test_data.testno            = int32(643);
test_data.runno             = int32(259);
test_data.pointno           = int32(3541);
test_data.GPS_start_time    = single(28800.0);
test_data.GPS_stop_time     = single(28830.0);
test_data.date_time         = '12/20/2018 08:00 ';
test_data.test_engineer     = 'William Humphreys';

%% Facility data

facility_data.facility_name = strcpy_and_pad('14- by 22-foot Subsonic Tunnel',80);
facility_data.facility_description = strcpy_and_pad('Open Jet Configuration',80);
facility_data.total_facility_parameters = int32(4);

facility_data.facility_parameter(1).long_name = 'PA';
facility_data.facility_parameter(1).description = 'Ambient Pressure';
facility_data.facility_parameter(1).data_type = 'real';
facility_data.facility_parameter(1).units = 'psfA';
facility_data.facility_parameter(1).value = '2129.52';

facility_data.facility_parameter(2).long_name = 'PTOT';
facility_data.facility_parameter(2).description = 'Settling Chamber Total Pressure';
facility_data.facility_parameter(2).data_type = 'real';
facility_data.facility_parameter(2).units = 'psfA';
facility_data.facility_parameter(2).value = '2129.429888';

facility_data.facility_parameter(3).long_name = 'TA';
facility_data.facility_parameter(3).description = 'Entrance Cone Ambient Temperature';
facility_data.facility_parameter(3).data_type = 'real';
facility_data.facility_parameter(3).units = 'deg F';
facility_data.facility_parameter(3).value = '57.315520';

facility_data.facility_parameter(4).long_name = 'TDEW';
facility_data.facility_parameter(4).description = 'Dew Point Temperature';
facility_data.facility_parameter(4).data_type = 'real';
facility_data.facility_parameter(4).units = 'deg F';
facility_data.facility_parameter(4).value = '29.393310';

for i = 5:200
   facility_data.facility_parameter(i).long_name = '';
   facility_data.facility_parameter(i).description = '';
   facility_data.facility_parameter(i).data_type = '';
   facility_data.facility_parameter(i).units = '';
   facility_data.facility_parameter(i).value = '';
end

%% Model data

model_data.model_id = int32(1);
model_data.model_description = strcpy_and_pad('HL-CRM Model',80);
model_data.total_model_parameters = int32(2);

model_data.model_parameter(1).long_name = 'PITCHM';
model_data.model_parameter(1).description = 'Mast Pitch Angle';
model_data.model_parameter(1).data_type = 'real';
model_data.model_parameter(1).units = 'deg';
model_data.model_parameter(1).value = '0.000000';

model_data.model_parameter(2).long_name = 'ALPHA';
```

```
model_data.model_parameter(2).description = ‘Model Angle of Attack’;
model_data.model_parameter(2).data_type = ‘real’;
model_data.model_parameter(2).units = ‘deg’;
model_data.model_parameter(2).value = ‘6.999002’;

for i = 3:200
   model_data.model_parameter(i).long_name = ‘’;
   model_data.model_parameter(i).description = ‘’;
   model_data.model_parameter(i).data_type = ‘’;
   model_data.model_parameter(i).units = ‘’;
   model_data.model_parameter(i).value = ‘’;
end

%% Static tap data

static_data.total_static_taps = 3;

static_data.static_tap_parameter(1).static_tap_designation = ‘ESP0101’;
static_data.static_tap_parameter(1).description = ‘PSI Pressure Tap’;
static_data.static_tap_parameter(1).units = ‘psi’;
static_data.static_tap_parameter(1).coefficient_name = ‘ESP0101’;
static_data.static_tap_parameter(1).data_value = ‘0.109163’;

static_data.static_tap_parameter(2).static_tap_designation = ‘ESP0102’;
static_data.static_tap_parameter(2).description = ‘PSI Pressure Tap’;
static_data.static_tap_parameter(2).units = ‘psi’;
static_data.static_tap_parameter(2).coefficient_name = ‘ESP0102’;
static_data.static_tap_parameter(2).data_value = ‘0.052045’;

static_data.static_tap_parameter(3).static_tap_designation = ‘ESP0103’;
static_data.static_tap_parameter(3).description = ‘PSI Pressure Tap’;
static_data.static_tap_parameter(3).units = ‘psi’;
static_data.static_tap_parameter(3).coefficient_name = ‘ESP0103’;
static_data.static_tap_parameter(3).data_value = ‘-0.058525’;

for i = 4:999
  static_data.static_tap_parameter(i).static_tap_designation = ‘’;
  static_data.static_tap_parameter(i).sdescription = ‘’;
  static_data.static_tap_parameter(i).units = ‘’;
  static_data.static_tap_parameter(i).coefficient_name = ‘’;
  static_data.static_tap_parameter(i).data_values = ‘’;
end do

return;
```

Appendix F.4 - Populating Header Values in Cross Spectral Matrix Data Files
Matlab Code Fragment

```
%% Version data

csm_version_data.netcdf_version = single(3.4);       %Note: Do not change these values
csm_version_data.hdf5_version   = single(1.8);       %
csm_version_data.hdr_version    = single(9.4);       %

%% Test data

csm_test_data.testno            = int32(643);
csm_test_data.runno             = int32(259);
csm_test_data.pointno           = int32(3541);
csm_test_data.date_time         = '12/20/2018 08:00 ';
csm_test_data.GPS_start_time    = single(28800.0);
csm_test_data.GPS_stop_time     = single(28830.0);

%% Channel data

for i = 1:number_of_channels
   csm_channel_data.sensor_sn_array(i)          = int32(mic_sn(i));
   csm_channel_data.preamplifier_sn_array(i)    = int32(preamp_sn(i));
   csm_channel_data.sensor_type_array(i)        = int32(1);
   csm_channel_data.preamplifier_type_array(i)  = int32(1);
   csm_channel_data.excitation_voltage_array(i) = single(200.0);
   csm_channel_data.excitation_current_array(i) = single(0.0);
end

for i = 1:number_of_channels
   csm_channel_data.sensor_coord_array(i,1)     = single(sensor_x(i));
   csm_channel_data.sensor_coord_array(i,2)     = single(sensor_y(i));
   csm_channel_data.sensor_coord_array(i,3)     = single(sensor_z(i));
end

csm_channel_data.array_sensor_coord_ref    = 'array';
csm_channel_data.sideline_sensor_coord_ref = 'tunnel';
csm_channel_data.kulite_sensor_coord_ref   = 'model';

%% Calibration data

for i = 1:number_of_channels
  csm_calibration_data.sensitivity_array(i) = single(1500.0);
end

for i = 1:number_of_bins
  csm_calibration_data.freq_calibration_array(i,1) = ...
      single((i-1)*frequency_resolution);
  csm_calibration_data.freq_calibration_array(i,2) = single(1.0);
  csm_calibration_data.freq_calibration_array(i,3) = single(0.0);
  csm_calibration_data.filter_response(i,1) = single((i-1)*frequency_resolution);
  csm_calibration_data.filter_response(i,2) = single(1.0);
  csm_calibration_data.filter_response(i,3) = single(0.0);
end

csm_calibration_data.number_of_cal_sources           = int32(3);
csm_calibration_data.cal_source_coordinates(1,1)     = single(114.34);
csm_calibration_data.cal_source_coordinates(1,2)     = single(22.85);
csm_calibration_data.cal_source_coordinates(1,3)     = single(-4.79);
csm_calibration_data.cal_source_coordinates(2,1)     = single(137.56);
csm_calibration_data.cal_source_coordinates(2,2)     = single(58.59);
csm_calibration_data.cal_source_coordinates(2,3)     = single(0.99);
csm_calibration_data.cal_source_coordinates(3,1)     = single(164.55);
csm_calibration_data.cal_source_coordinates(3,2)     = single(99.85);
csm_calibration_data.cal_source_coordinates(3,3)     = single(5.07);
csm_calibration_data.cal_source_coord_ref_array(1,:) = 'model';
csm_calibration_data.cal_source_coord_ref_array(2,:) = 'model';
csm_calibration_data.cal_source_coord_ref_array(3,:) = 'model';

csm_calibration_data.number_of_cal_measurements = int32(1);
```

```
csm_calibration_data.date_time_of_cal(1,1,:) = '12/20/2018 08:00 ';
csm_calibration_data.date_time_of_cal(2,1,:) = '12/20/2018 08:00 ';
csm_calibration_data.date_time_of_cal(3,1,:) = '12/20/2018 08:00 ';

for i = 1:number_of_channels
   csm_calibration_data.cal_source_baseline_array(i,1,1) = single(0.0);
   csm_calibration_data.cal_source_baseline_array(i,2,1) = single(0.0);
   csm_calibration_data.cal_source_baseline_array(i,3,1) = single(0.0);
   csm_calibration_data.cal_source_level_array(i,1,1)    = single(0.0);
   csm_calibration_data.cal_source_level_array(i,2,1)    = single(0.0);
   csm_calibration_data.cal_source_level_array(i,3,1)    = single(0.0);
   csm_calibration_data.injection_cal_baseline_array(i)  = single(1.0);
   csm_calibration_data.injection_cal_level_array(i)     = single(1.0);
end

csm_calibration_data.sensitivity_correction_switch = int32(0);
csm_calibration_data.temperature_correction_switch = int32(0);
csm_calibration_data.db_correction_switch          = int32(1);

%% DAS data

csm_DAS_data.total_channels  = int32(97);
csm_DAS_data.sample_count  = int32(3000000);
csm_DAS_data.sample_period = single(0.005);

for i = 1:number_of_channels
   csm_DAS_data.full_scale_range_array(i) = single(10000.0);
end

for i = 1:number_of_channels
   csm_DAS_data.lpf_cutoff_freq_array(i)      = single(102300.0);
   csm_DAS_data.lpf_pregain_array(i)          = single(1.0);
   csm_DAS_data.lpf_postgain_array(i)         = single(1.0);
   csm_DAS_data.hpf_cutoff_freq_array(i)      = single(400.0);
   csm_DAS_data.hpf_pregain_array(i)          = single(1.0);
   csm_DAS_data.hpf_postgain_array(i)         = single(1.0);
   csm_DAS_data.external_gain_offset_array(i) = single(20.0);
end

%% Array data

csm_array_data.total_array_sensors               = int32(97);
csm_array_data.traverse_configuration            = 'Standard configuration';
csm_array_data.sideline_traverse_x               = single(0.0);
csm_array_data.sideline_traverse_y               = single(0.0);
csm_array_data.sideline_traverse_z               = single(0.0);
csm_array_data.sideline_traverse_coord_ref       = 'tunnel';
csm_array_data.array_traverse_x                  = single(0.0);
csm_array_data.array_traverse_y                  = single(0.0);
csm_array_data.array_traverse_z                  = single(0.0);
csm_array_data.array_traverse_coord_ref          = 'tunnel';
csm_array_data.array_elevation                   = single(0.0);
csm_array_data.array_azimuth                     = single(0.0);
csm_array_data.number_of_inclinometers           = int32(3);
csm_array_data.inclinometer_coordinates(1,1)     = single(10.0);
csm_array_data.inclinometer_coordinates(1,2)     = single(20.0);
csm_array_data.inclinometer_coordinates(1,3)     = single(0.0);
csm_array_data.inclinometer_coordinates(2,1)     = single(20.0);
csm_array_data.inclinometer_coordinates(2,2)     = single(10.0);
csm_array_data.inclinometer_coordinates(2,3)     = single(0.0);
csm_array_data.inclinometer_coordinates(3,1)     = single(-10.0);
csm_array_data.inclinometer_coordinates(3,2)     = single(-20.0);
csm_array_data.inclinometer_coordinates(3,3)     = single(0.0);
csm_array_data.inclinometer_coord_ref            = 'model';
csm_array_data.inclinometer_phi_array(1)         = single(0.0);
csm_array_data.inclinometer_phi_array(2)         = single(0.0);
csm_array_data.inclinometer_phi_array(3)         = single(0.0);
csm_array_data.inclinometer_theta_array(1)       = single(0.0);
csm_array_data.inclinometer_theta_array(2)       = single(0.0);
csm_array_data.inclinometer_theta_array(3)       = single(0.0);
csm_array_data.inclinometer_rotation_array(1)    = single(0.0);
csm_array_data.inclinometer_rotation_array(2)    = single(0.0);
```

```
csm_array_data.inclinometer_rotation_array(3)    = single(0.0);
csm_array_data.inclinometer_temperature_array(1) = single(65.0);
csm_array_data.inclinometer_temperature_array(2) = single(65.0);
csm_array_data.inclinometer_temperature_array(3) = single(65.0);
csm_array_data.photogrammetric_x_location        = single(0.0);
csm_array_data.photogrammetric_y_location        = single(0.0);
csm_array_data.photogrammetric_z_location        = single(0.0);
csm_array_data.photogrammetric_yaw_angle         = single(0.0);
csm_array_data.photogrammetric_pitch_angle       = single(0.0);
csm_array_data.photogrammetric_roll_angle        = single(0.0);
csm_array_data.photogrammetric_number_targets    = int32(20);
csm_array_data.photogrammetric_RMS_error         = single(0.0);

%% Facility data

facility_data.facility_name = strcpy_and_pad('14- by 22-foot Subsonic Tunnel',80);
facility_data.facility_description = strcpy_and_pad('Open Jet Configuration',80);
facility_data.total_facility_parameters = int32(4);

facility_data.facility_parameter(1).long_name = 'PA';
facility_data.facility_parameter(1).description = 'Ambient Pressure';
facility_data.facility_parameter(1).data_type = 'real';
facility_data.facility_parameter(1).units = 'psfA';
facility_data.facility_parameter(1).value = '2129.52';

facility_data.facility_parameter(2).long_name = 'PTOT';
facility_data.facility_parameter(2).description = 'Settling Chamber Total Pressure';
facility_data.facility_parameter(2).data_type = 'real';
facility_data.facility_parameter(2).units = 'psfA';
facility_data.facility_parameter(2).value = '2129.429888';

facility_data.facility_parameter(3).long_name = 'TA';
facility_data.facility_parameter(3).description = 'Entrance Cone Ambient Temperature';
facility_data.facility_parameter(3).data_type = 'real';
facility_data.facility_parameter(3).units = 'deg F';
facility_data.facility_parameter(3).value = '57.315520';

facility_data.facility_parameter(4).long_name = 'TDEW';
facility_data.facility_parameter(4).description = 'Dew Point Temperature';
facility_data.facility_parameter(4).data_type = 'real';
facility_data.facility_parameter(4).units = 'deg F';
facility_data.facility_parameter(4).value = '29.393310';

for i = 5:200
   facility_data.facility_parameter(i).long_name = '';
   facility_data.facility_parameter(i).description = '';
   facility_data.facility_parameter(i).data_type = '';
   facility_data.facility_parameter(i).units = '';
   facility_data.facility_parameter(i).value = '';
end

%% Model data

model_data.model_id = int32(1);
model_data.model_description = strcpy_and_pad('HL-CRM Model',80);
model_data.total_model_parameters = int32(2);

model_data.model_parameter(1).long_name = 'PITCHM';
model_data.model_parameter(1).description = 'Mast Pitch Angle';
model_data.model_parameter(1).data_type = 'real';
model_data.model_parameter(1).units = 'deg';
model_data.model_parameter(1).value = '0.000000';

model_data.model_parameter(2).long_name = 'ALPHA';
model_data.model_parameter(2).description = 'Model Angle of Attack';
model_data.model_parameter(2).data_type = 'real';
model_data.model_parameter(2).units = 'deg';
model_data.model_parameter(2).value = '6.999002';

for i = 3:200
   model_data.model_parameter(i).long_name = '';
   model_data.model_parameter(i).description = '';
```

```
   model_data.model_parameter(i).data_type = '';
   model_data.model_parameter(i).units = '';
   model_data.model_parameter(i).value = '';
end

%% CSM data

csm_data.fft_length      = int32(8192);
csm_data.overlap         = int32(0);
csm_data.num_blocks      = int32(500);
csm_data.num_blocks_used = int32(500);

for i = 1:number_of_blocks
   for j = 1:number_of_channels
       block_table(i,j) = int32(1);
   end
end

csm_data.cslength  = int32(4096);
csm_data.winfunc   = 'hamming';
csm_data.beta      = single(0.0);
csm_data.CSM_units = int32(1);

return;
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
1-09-2019

2. REPORT TYPE
Technical Memorandum

3. DATES COVERED *(From - To)*

4. TITLE AND SUBTITLE
Microphone Phased Array NetCDF/HDF5 Archival Files: Application Program Interface Reference

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Humphreys, William M., Jr.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
081876.02.07.03.01.02

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
L-21055

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)
NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NASA-TM-2019-220402

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-
Subject Category 71
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT

An application program interface (API) has been developed for the creation and access of structured data files generated by microphone phased arrays utilized in aeroacoustics research. Two structured binary file formats are supported, namely NetCDF (Network Common Data Form) and HDF5 (Hierarchical Data Format) files. The API consists of a library of routines callable from C, Fortran or Matlab, with native versions of the API provided for each language. The libraries are divided into categories for file handling, file definition and initialization, data writing, data recovery, and error handling. The API is intended to provide a mechanism for generating self-describing binary files for long-term archiving of raw and processed data generated by phased array systems.

15. SUBJECT TERMS
API; HDF5; NetCDF; acoustics; phased arrays

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			STI Help Desk (email: help@sti.nasa.gov)
U	U	U	UU	192	19b. TELEPHONE NUMBER *(Include area code)* (757) 864-9658