



UNEXMIN DELIVERABLE D6.1

DATABASE SPECIFICATIONS MANUAL

Summary:

This document serves the purpose of providing a general presentation of the UNEXMIN Website, describing all its content, elements, design and functionalities. The final Project Website has detailed information about project's objectives, concept and approach, partners, work packages, dissemination material and all other important project related content.

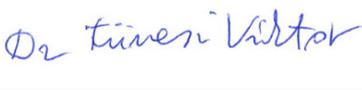
Authors:

RCI - Resources Computing International Ltd



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 690008.

Lead beneficiary:		Resources Computing International Ltd (RCI)	
Other beneficiaries:			
Due date:		M11	
Nature:		Report	
Diffusion		Public	
Revision history	Author	Delivery date	Summary of changes
Version 1.0	Stephen Henley	18.12.2016	
Version 2.0			
Version 3.0			
Version 4.0			
Version 4.1			

Approval status			
Function	Name	Date	Signature
Deliverable responsible	Stephen Henley	22.12.2016	
WP leader	Stephen Henley	22.12.2016	
Reviewer 1	Csaba Vörös	20.12.2016	
Reviewer 2	Viktor Füvesi	19.12.2016	
Project leader	Norbert Zajzon	22.12.2016	

Disclaimer: This report reflects only the author's view. The European Commission is not responsible for any use that may be made of the information it contains.

Table of Contents

1	EXECUTIVE SUMMARY	5
2	INTRODUCTION	6
3	NOTATION	7
4	PRE-PROCESSING AND STORAGE	8
4.1	DATA STORAGE, BACKUP PROCEDURES.....	8
4.2	CONVERSION AND VALIDATION.....	8
5	METADATA	10
5.1	MISSION METADATA.....	10
5.2	SENSOR METADATA.....	10
5.3	CAMERA METADATA.....	10
6	DATA	11
6.1	DATA TYPES.....	11
6.2	DATA FILE STRUCTURES.....	11
6.3	NAVIGATION AND ATTITUDE DATA.....	11
6.4	SENSORS AND CAMERAS.....	13
6.5	UX SYSTEM DATA.....	18
7	FILE FORMATS AND NAMES	19
7.1	FILE AND TABLE NAMING CONVENTIONS.....	19
7.2	FILE FORMATS.....	19
8	DATABASE MANAGEMENT	21
8.1	CHOICE OF DATA MANAGEMENT STRATEGY.....	21
8.2	DATABASE MANAGEMENT SYSTEM.....	25
8.3	TABLE STRUCTURE.....	26
8.4	CODD FLAGS.....	26
8.5	TIMESTAMP INTERPOLATION.....	27
8.6	TABLE LINKAGE.....	28
8.7	SEARCH, SELECTION AND RETRIEVAL OF DATA.....	28
9	REFERENCES	30
10	ANNEX A: SUMMARY OF DATA TABLES	31

1 Executive Summary

This document identifies the steps required to transfer data and metadata gathered in missions of submersible UX-1 to a database where it is available for use in post-processing applications. All data will be stored: navigation-related, data from sensors and cameras, and submersible status and mission control/response data. The wide range of different items of recording equipment will store data on-board the submersible in different formats and widely differing volumes.

On completion of a mission, these data and metadata will all be transferred to a backup storage medium, but cannot easily be used in the recorded formats. They require pre-processing and conversion to a common standard file structure and format which can be held in a standard database management system. The details of the conversion software required will necessarily be defined separately for each data set. However, the destination file structure can be specified, and will be a relational database table. This document defines the structures of proposed database tables for all data types.

There are a large number of database management systems from which to choose as hosts for the database. A short list of candidate systems has been prepared, and one system PostgreSQL has been identified as a potential system to be used. This is an open-source software system, and also has an add-on product (also open-source) PostGIS, for spatial data handling. These are currently under evaluation.

Other questions, independent of the software platform to be used, are also addressed. These include the naming conventions for files and tables, the processing which will be required to generate consistent sets of records from raw data in which timestamps do not necessarily match, and a proposed method of handling missing data values.

2 Introduction

UX-1 will operate autonomously, collecting data of a variety of different kinds during submerged missions from a few minutes to several hours in length. Within each mission, sensors, light sources, and cameras will be switched on and off at different times under control of a set of mission commands. The timing for all switching will be set according to a central system clock, and data from the sensors and cameras can be linked through time stamps recorded with their data.

Together with the recorded data, there will also necessarily be metadata which records relative locations of sensors, angles of view of cameras, and operating parameters. Information linking metadata to the data sets for each sensor or camera will be encoded in standardised file names.

On completion of each mission, a full backup copy of the recorded data will be made. The metadata and data (recorded within ROS or Linux operating systems) will then be converted as necessary into files which can be read by software running on the Microsoft Windows platform.

First pass validation which will exclude or correct rogue data items should be carried out during the conversion process.

Because timestamps on data from different sensors and cameras will not necessarily coincide, it will in general be necessary to expand the data tables to provide interpolated estimates for all timestamps. This interpolation process may be carried out most conveniently by a separate program before import into a database management system.

After import into the database management system, second-pass validation will be carried out to check the consistency of the data.

The database management system to be used will be selected based on its capabilities to handle very large data tables efficiently, to allow the use of flags or markers to distinguish between real data elements and computed (interpolated) estimates, and to deal with image and video data in addition to the simpler univariate or multivariate data sets from sensors.

3 Notation

This section defines the notation that will be used not only in this document but in the design and development of the post-processing software for the UNEXMIN project.

Lower case letters in normal type denote scalar numeric data items which may be of either 'integer' or 'double' (64-bit double-precision floating-point) type	x
Timestamp values are represented by the lower case letter t with a subscript indicating the sequential clock tick	t_i
Lower case letters in bold type denote vectors of numeric items $\mathbf{x} = (x_1, x_2, \dots, x_n)$	\mathbf{x}
Upper case letters in normal type denote character data items	A
Values in curly brackets denote mission and sensor identifiers	$\{M\}$
Values in square brackets after a data item represent "Codd flags" which are markers to indicate presence/absence/significance of the data item they are attached to.	$[c]$

Data from each sensor typically would be represented for the j 'th timestamp by $t_j \mathbf{x}_j$

In other words, simply the timestamp and a data vector for each time j

Location and attitude data are recorded in the same way, and linked to each observation from each sensor through the timestamp values.

4 Pre-processing and storage

4.1 Data storage, backup procedures

Data from UX-1 will be of three main kinds:

- navigation-related,
- sensor and camera data, and
- UX-1 status.

A standard procedure that should be adopted to minimise risk of data loss is that at the end of a mission ALL data must be stored in its raw form (ROS/Linux file system) on a backup hard drive which is kept in a secure environment. As soon as practicable, a further backup of all data from this drive will also be taken and maintained in an independent off-site location.

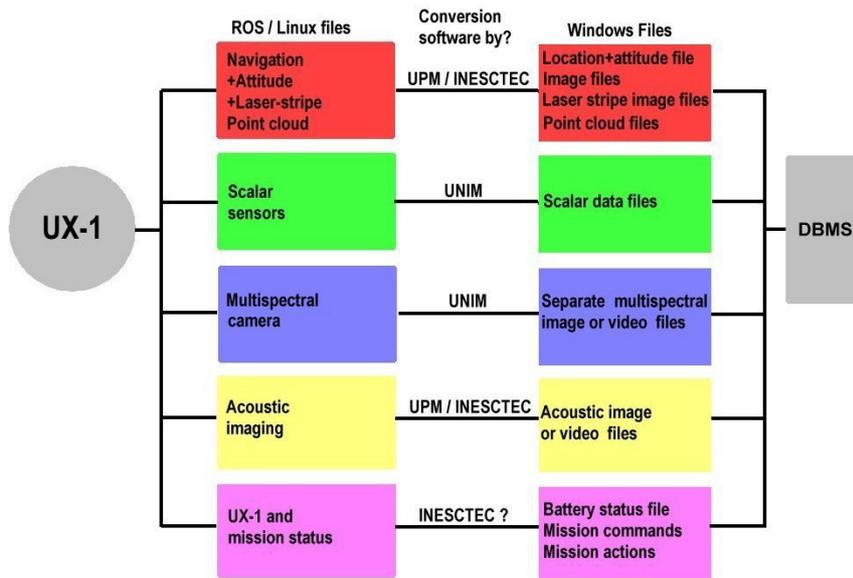
Data will then be processed as described in this document, and a full backup copy of the processed data (Windows file system) will also be transferred on a backup hard drive to a secure off-site location.

At each step in the computational process, backup copies need to be made, and retained in safe locations. We would not advise "cloud" storage because of the time taken to upload and download the very large volumes of data. It is better to have separate external hard disk drives which can physically be stored in different locations.

4.2 Conversion and validation

Figure 4.2.1 shows the anticipated requirements for data conversion. Details of file structures to be exported by the various conversion programs are given in section 5 below.

Figure 4.2.1: UX-1 data conversion tasks from UX-1 robot to DBMS (database management system)



UX-1 data conversion requirements

First-stage validation will be carried out during the conversion process, to ensure that gross corruption is detected and offending data are excluded or corrected as appropriate.

This will include plausibility range checks on each data item, and sequence checks on time stamps. Suggested responsibilities for development of conversion software are indicated in Figure 4.2.1, but detailed allocation of tasks is yet to be agreed.

5 Metadata

All data held in the database will have associated metadata to provide the context: a mission identifier, mission location, date, and more detailed information as may be relevant to the instrument including reference to any initialisations or calibration checks carried out.

5.1 Mission metadata

For each mission, a general mission metadata record will be created. This will include a location code, robot identification, date of mission, and other information, including a free text summary of the mission purpose. This record, after conversion, will be appended to a mission metadata file in the database.

There is also a need for more descriptive information which should be both generated by, and available to, the team operating the robot. This would include a record of events, problems encountered, weather conditions which might be relevant (such as excessive rainfall that would affect water chemistry or flow rates), names of visitors and observers, etc. This could be held within the database and/or recorded on paper, such as an engineer's logbook.

5.2 Sensor metadata

For each sensor, one or more metadata records will be generated, to record such information as the mission identifier, the sensor identifier, start and stop timestamps (there could be several of these within any one mission), recording interval expressed in UX-1 system clock ticks, and other data which may be required such as sensor self-test data or its 3D offset position relative to the UX-1 centre point.

5.3 Camera metadata

Cameras will require all of the same metadata as scalar sensors, plus also direction of view (relative to the UX-1 reference frame), field of view, horizontal and vertical resolution in pixels, start and end UX-1 system clock ticks, and any relevant addition information such as light source unit(s) used, and recording format (RAW, TIF, PNG, JPG, etc). As with sensor metadata there could be one or more metadata records for each camera within any mission, with one record for each recording interval. Cameras may have their own internal clocks, but the start/end system clock ticks recorded in the metadata may be used to convert recorded camera time to UX-1 system time.

Before every mission a calibration process of the cameras is necessary. The distortion of lens and dome have to be calculated for use during the post-processing calculations. This information must also be included in the camera metadata. The distortion transformation is a function of lens, positioning of the sensor in the housing, irregularity of material of the glass dome, etc.

6 Data

6.1 Data types

Most data will be numeric.

The primary key will generally be the **timestamp**, which must be recorded as an integer value (notional clock ticks as recorded by the UX-1 central system clock - possibly milliseconds) as an offset from start of mission. If held as a 32-bit integer variable, the range of integers which can be stored is from -2147483647 to +2147483647 which will be more than adequate for any realistic mission length.

A data set for which the timestamp is NOT the primary key will be the point cloud resulting from the laser-stripe camera system, which records the walls, roof, and floor of the flooded mine. This will have a composite primary key: the X,Y,Z point location, probably encoded in an octree index.

Most recorded values will themselves be either integer or floating point numbers. A standard 32-bit floating point value (single precision) provides only 6 digits precision, which may be insufficient for some data. A decision on whether to use 32-bit or 64-bit floating point values will be dependent upon the precision provided by the instruments themselves. Current information suggests that 32-bit precision may be sufficient.

The sensors and navigation units will not generate character data. Characters will in general be needed only for construction of file names and for codes and descriptive text in metadata. 8-bit characters using the standard ASCII set should be sufficient, though additional flexibility could be obtained if required by using 16-bit Unicode characters.

6.2 Data file structures

All data will be held in flat tables in which all rows carry the same types of data in the same set of fields. This follows the relational model. For pure relational databases, the order of rows is immaterial. In UNEXMIN, the order of data from sensors and other data recording units on UX-1 will be fixed (ascending timestamp values). For derived or computed data, the order may differ from this: for example, point cloud data may be sorted by spatial coordinate values or octree index values¹.

6.3 Navigation and attitude data

6.3.1 Data to be generated

The navigation and attitude system on UX-1 generates different types of data:

¹ Octree encoding description: <https://en.wikipedia.org/wiki/Octree>

- a) UX-1 location (x,y,z coordinates)
- b) UX-1 attitude (pitch, yaw, and roll angles)
- c) Camera images from LED illumination
- d) Camera images from laser stripe illumination
- e) Point cloud extracted from laser stripe, computed on-board with absolute x,y,z coordinates (and other data - intensity and colour?)
- f) Acoustic data (sonar) images

These different data sets need to be held in separate files. The camera images (c and d) may be held either as separate uncompressed data files or collectively in a non-lossy video file format.

6.3.2 Survey coordinate system

All data will be converted, before admission to the database, to the global coordinate system defined for the location (the mine where the current set of missions is carried out). This may be a locally defined mine grid, or a national grid, or UTM coordinates. Whichever grid system is used, the launch location must be accurately keyed to the grid using existing or new survey stations, and the location and orientation of UX-1 must be initialised accurately to this grid at the start of each mission. This is a crucial requirement, as data collected on successive missions (including new point clouds) must be registered accurately to the same point cloud for the mine walls, roof, and floor.

6.3.3 UX-1 location

The navigation subsystem will use the mission initialisation data (starting coordinates and UX-1 orientation) to record absolute locations in three dimensions at a set of regular time intervals, and will produce a table of data in which each row consists of timestamp, X, Y, Z. The time intervals must be sufficiently short that any linear interpolation of locations will yield estimated locations at an acceptable level of accuracy (+/- 1 mm). The laser stripe system should produce stripes at not greater than 25mm spacing.

6.3.4 UX-1 attitude: pitch, yaw and roll

The pitch, yaw, and roll data obtained from the pendulum and spinning disk subsystems will be recorded on the same regular time intervals as the locations, yielding a table with each row containing timestamp, pitch angle, yaw angle, roll angle.

From the combination of location and attitude data it will be possible to determine the absolute location (and direction of view for cameras) of every data recording device on UX-1. Interpolation of location and attitude data may be required if time stamps on the recording devices (sensors) do not match those of the navigation and attitude systems.

6.3.5 Camera LED images

The laser stripe units will each include a colour camera which will record a full colour high-resolution image (with white-light LED illumination) in between a sequence of laser stripe images. This image may be stored either as a frame within a (non-compressed) video format file, or as a separate still image file. These images will most likely either be stored initially in uncompressed RAW or JPG format, or will be extracted automatically (such as every n'th image from a video) into a series of uncompressed RAW or JPG files. It is possible that other formats may be used instead: the only restrictions from the point of view of post-processing is that images must be stored using well-defined industry standard formats; if compressed must use a non-loss data compression method.

6.3.6 Camera laser stripe images

The laser stripes will be processed on-board the robot. These must in general be considered as 'noisy' data, as the stripes can be disturbed by suspended particles in the water and other problems. There will be some on-board automatic cleaning of the data to generate the point cloud. It is not intended that the raw images will be retained.

6.3.7 Point cloud from laser stripes

On-board processing will sample the laser stripes at regular intervals to generate a set of points along each stripe. These points must be recorded together with associated information: timestamp, stripe packet no. (the image no.), stripe no. within packet, X,Y,Z absolute location or location relative to UX-1 (which can be converted subsequently to absolute coordinates). Also to be recorded for each point, whether or not the X,Y,Z are absolute locations, should be the UX-1 location (reference point, whether the centre or other defined reference point for within-robot relative coordinates), attitude angles, and traversed distance within mission. For each laser stripe the scanning angle (as the laser nods up and down) should also be recorded. Stripe packets are the set of one or more stripes between LED images. Stripe numbers are the sequential numbers of stripes within a packet. These numbers are important because they will allow more efficient post-processing such as triangulation of the point cloud.

6.3.8 Acoustic images

Data from the forward-looking acoustic camera will be recorded as virtual images, with format to be identified, dependent on selection of the sensor device to be used.

6.4 Sensors and cameras

The exact combination of sensors to be installed is not finally decided and in any case can change from one mission to another. The sets of data below are therefore not the only data which may be available, as additional sensors may also be included at a later date. For this reason also, precise formats are not defined: the CSV format allows sufficient freedom to encode any necessary data. Annex A includes as full a list as it is

possible to construct at the present stage, but may need to be updated from time to time as the list of sensors changes.

6.4.1 Temperature

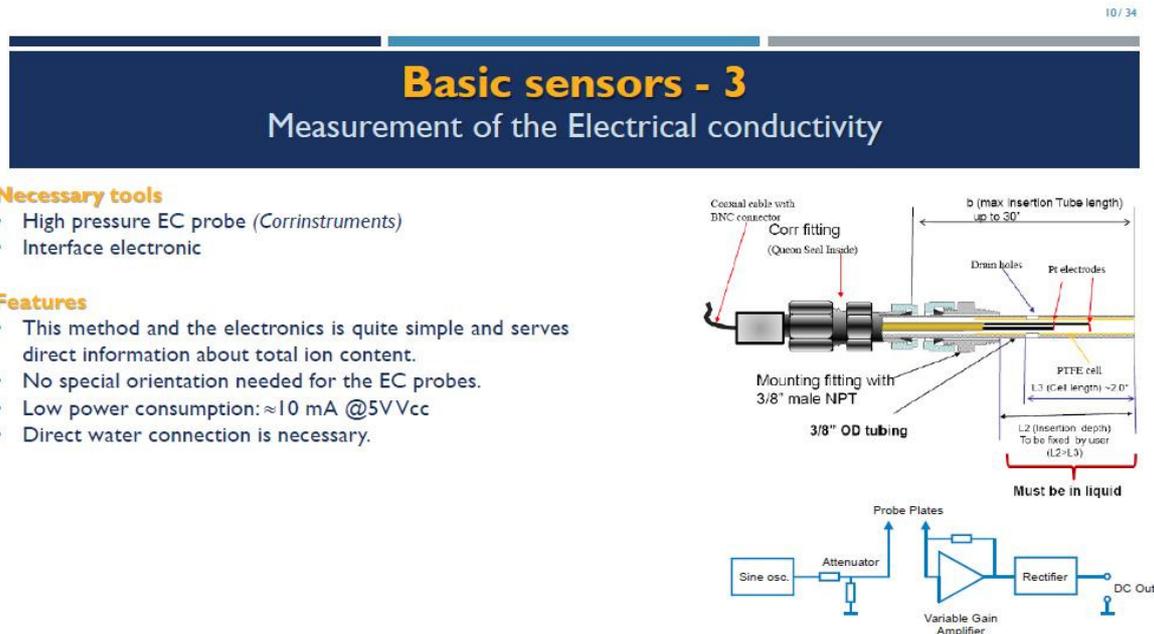
Temperature will be measured by various sensors on UX-1 but during conversion of data from instruments should be extracted to a separate database file as it may be required for various different purposes. This file will contain just two columns: timestamp and temperature.

6.4.2 Pressure

As with temperature, water pressure will be measured by various sensors on UX-1 but during conversion of data from instruments should be extracted to a separate database file as it may be required for various different purposes. This file will contain just two columns: timestamp and water pressure.

6.4.3 Electrical conductivity

Figure 6.4.3.1 Measurement of electrical conductivity



This file will contain just two columns: timestamp and conductivity.

6.4.4 pH (acidity/alkalinity) values

Figure 6.4.4.1 Measurement of pH

11 / 34

Basic sensors - 4

Measurement of pH - equipments

Necessary tools

- High pressure glass based pH probe (*Corrinstruments*)
- High pressure reference probe

Features

- Up to 80°C and 140 bar
- An Ag/AgCl separate reference electrode is required
- Warranty is 1 year / probe's life should be good for 2 years
- Direct water connection is necessary.
- The Lo (Overall Length) value for the shortest probe would be about 10" after we reduce the b (Maximum Insertion Length) value to about 5"

The pH value is sensitive to pressure, and more strongly to temperature, so it will be necessary to pre-process the data before admitting it to the database, using data from the reference electrode. pH values in the range of 0 to 14 will be stored, together with the timestamp.

6.4.5 Magnetic field

Figure 6.4.5.1 measurement of magnetic field

15 / 34

Magnetic Field

Sensors

Sensors

The arrangement is same as the Spontaneous potential: 6 pcs
(at the end of the 3 main axes in arbitrary Descartes coordinate systems)

Manufacturer and type: Speake & Co Llanfapley, FGM-series Magnetic Field Sensors

Maximum Supply Voltage: 7 volts

Recommended supply Voltage: 5 volts ±0.5 volts

Typical Supply Current: 12 mA

Operating Temperature Range: 0 - 50 °C

Approximate Outline Dimensions

Magnetic field data are directional. The sensors will record total field strength and field orientation, which must be converted to the global coordinate system before admission to the database. The sensors will also give six temperature readings which can be averaged to give an output temperature value, and are also used for UX-1 self-test diagnostic purposes.

After conversion, the data to be returned will include the timestamp, field strength, and direction cosines for field direction, as well as a temperature value to be stored in a separate table.

6.4.6 Gamma Ray Count

Figure 6.4.6.1 Measurement of gamma ray flux

19 / 34

Natural gamma ray measurements




Properties
IH-99DNC dose rate meter for high dose rate measurement

Detector: energy-compensated GM-tube.
Power supply: 12V, (+/-20%) max. 25mA
Temperature range: -30 ... + 50°C
Communication: RS-485

Effective gamma dose-rate measurement range:
Range: 1µGy/h ... 1 Gy/h
Indication range: 20nGy/h ... 100 Gy/h
Energy range: 60keV...1,5Mev
Measurement time: 2s ... 4min, automatic
β-radiosensitivity: 0.3%

The instrument is a nuclear radiation measuring device with two operational modes. It operates as a gamma radiation dose and dose rate meter.



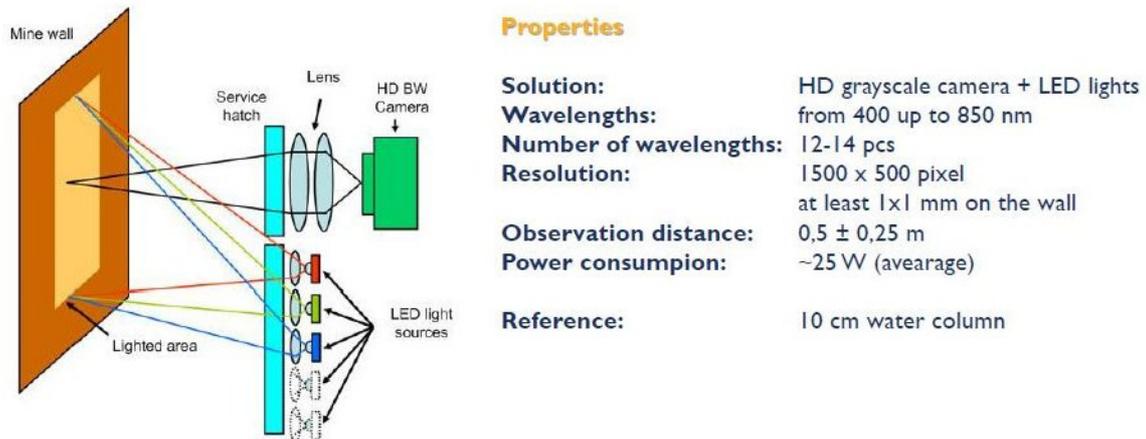
MISKOLCI
EGYETEM
UNIVERSITY OF MISKOLC

Gamma ray data will be recorded as a total count within a standard time interval. This will be recorded with a single timestamp (which indicates the start of this interval). The gamma ray count is significantly attenuated by water, so it will be necessary to post-process the data by adjusting the recorded counts for the distance of the sensor from the rock wall, using appropriate calibration curves. The attenuation curves for uranium and thorium are different, so it will be necessary in any mission to decide which element (or a mixture) should be the most suitable.

After conversion, the data file will consist of a table containing timestamp and the raw and adjusted gamma counts.

6.4.7 Multispectral camera data

Figure 6.4.7.1 Multi-spectral camera setup



This camera will record data in greyscale, using a set of 12-14 monochromatic LEDs used sequentially with fixed (probably 50millisecond) exposures. The metadata for this camera must record its position on UX-1, calibration and distortion data, the set of wavelengths used, the direction of view, and the angle of view, so that ray tracing may be used to map each pixel to the wall point cloud obtained from the navigation laser stripes. For each recording session (of which there may be several within each mission) the main UX-1 clock will be used, to put the start and stop times into the metadata file.

Data will be recorded on the camera either as a series of still images in separate files, or as a virtual video file from which the still images can be extracted during the conversion process. Timestamps within any recording session may be generated automatically by interpolation from the start and stop times recorded in the metadata.

6.4.8 Fluorescence images

An ultraviolet light source will be used to allow fluorescent images to be obtained, using the colour recording navigation camera. The exposure times for this will be recorded from the main UX-1 clock. These images will be recorded within the cycle of multi-spectral images, but by a colour camera rather than the monochrome camera used for the multi-spectral data.

6.4.9 Sub-bottom sonar

Data from a sub-bottom sonar profiling unit, if fitted, will allow a continuous profile of the floor along horizontal tunnels, to give information on the floor properties - indicating whether it is solid rock or includes some depth of mud, for example. It is likely that timestamps will not be recorded within a profiling sequence, but will need to be computed from a metadata recorded giving profiler start and stop times. Location and direction of view will be recorded. Data will be recorded in, or converted to, the standard SEG-Y format as used for seismic data.

6.4.10 Summary of Data Formats

The formats for data from all instruments, for import into the database, are summarised in Annex A.

6.5 UX system data

6.5.1 Battery status file

This file records the battery status (for each system battery, if this is possible) at regular time intervals: charge level and any warnings (such as high discharge rate which might indicate a short-circuit, or fuse burnouts).

6.5.2 Mission command file

The mission setup file is the set of commands supplied to the UX-1 central processor for each mission. This file includes absolute instructions (to be executed unconditionally) as well as instructions to make decisions dependent upon data obtained during the mission.

6.5.3 Mission actions file

This file, which may also be called a "mission log", records the actions that are **actually** taken by UX-1 during a mission, including the decisions taken, and the starting and stopping of data recording on each sensor, and of thrusters, pendulum, buoyancy unit, and other navigation controls. There will be at least one mission log, and potentially multiple mission logs for actions by different subsystems of UX-1.

7 File Formats and Names

7.1 File and Table naming conventions

There will be a very large number of separate data and metadata files generated by UNEXMIN missions. A file naming convention is therefore absolutely essential, in order to assist organisation and retrieval of data. The following standard naming convention is proposed for use both as file names and in naming relational database tables (see notation conventions in section 3 above):

{M}rrrr.UXM for metadata

{M}rrrr.UXD for scalar data

{M}rrrr.VVV for video data, where VVV is the standard extension for the video format used

{M}rrrr-fffff.PPP for image data, where PPP is the standard extension for the image format used

{M}	mission/sensor identifier in the form XXXyyyymmddnnSSS
XXX	three-character location code (such as IDR = Idrija),
yyyymmdd	the date (e.g. 20160801 for 1st August 2016)
nn	sequential number of the mission on that date (in range 01 to 99)
SSS	a code identifying the sensor unit data stream
rrrr	sequential number of the recording session within the mission (in range 0001-9999). A recording session is the interval between turning a sensor on and turning it off again
fffff	a six-digit sequential identifier (in the range 000001-999999) of the frame or image within a recording interval

Industry standard video and image formats are allowed. It is preferred that uncompressed data, or formats using lossless compression, be used, to avoid any loss of data collected by the cameras. The preferred image format is uncompressed RAW, though if necessary to save storage space, uncompressed JPG may be used. Each image and each video sequence shall be held as a separate file.

7.2 File formats

The image and video formats may be selected from those identified above. Data from other sensors, and metadata, should all be converted to a standard CSV format, using semicolon or tab delimiters (rather than commas). The delimiter character used should be excluded from the list of legal characters within data sets to avoid any confusion. The semicolon might be preferable because it is printable, but care should then be taken to exclude it from any free text fields within metadata.

In pre-processing any free text within metadata, the chosen delimiter character should be replaced by spaces. In floating-point numbers, there must be no thousands separator, but either dot or comma may be used for the decimal point character, provided that usage is consistent within any file. **For consistency the consortium**

must adopt a single standard for this: it is proposed here that the decimal point be represented by the dot character: "."

Where dates and times are used, a standard format should always be adopted: the format proposed is:

YYYY-MM-DD hh:mm:ss+HH (or -HH) offset to GMT

No special coding is needed for adjustment to daylight saving time.

The primary key for most files will be the **timestamp**, which must be recorded as an integer value (notional clock ticks - possibly milliseconds) as an offset from start of mission. If held as a 32-bit integer variable, the representable range is -2147483647 to +2147483647 which will be more than adequate for any realistic tick length. In practice, tick counts will start at zero. The negative part of the range will not be used.

CSV files should contain field names in the top line, and data values in all succeeding lines.

8 Database Management

8.1 Choice of Data Management Strategy

There are 3 principal options:-

- (1) simple text-format files (such as CSV) for everything
- (2) a database management system such as PostgreSQL for data storage / sharing
- (3) a standard data interchange format such as NetCDF or HDF5 for data storage / sharing

Each has advantages and disadvantages.

8.1.1 Text-format files

Either fixed-format or CSV files are a standard and well-tried medium for data transfer and (less so) for data storage. They have the advantages of simplicity, legibility, and portability across different computer architectures, operating systems, and languages. There are three significant disadvantages: storage space requirements (unless zip or some other compression algorithm is used), processing time (conversion to and from the text format), and inflexibility: lack of any functionality for selective retrieval or for multi-file operations such as merging two data sets, for which explicit coding must be provided.

There are a few text-format database management systems which have been developed. The purpose of these is to provide simple human-readable data tables which can be processed in the same way as normal relational database management systems. Although they would appear to offer the best of both worlds, they are generally incomplete, and because of the need to convert numeric data to and from text formats they are very slow for large volume data sets, and also potentially compromise data integrity through rounding errors in the format conversion. These systems are not widely used and are of questionable reliability.

8.1.2 Database management systems

A standard relational database system such as MS Access, Oracle, MySQL, or PostgreSQL provides completely general and transparent functionality for data files and collections of files with simple to highly complex interrelationships. The principal disadvantages are concerned with speed of operation. However, this is outweighed by the flexibility of data handling (file merging, selective retrieval, etc) using SQL. Most DBMSs on Windows operating system can be accessed directly from applications software through an ODBC (Open DataBase Connectivity) library. Where this may be unavailable, it is always possible to export data into CSV or other text format files for import to applications programs.

8.1.3 Data interchange formats

The two most widely used interchange formats are NetCDF² and HDF5³, with the intended purpose of fast and storage-efficient handling of simple or complex data structures. The definitions of the two are fairly similar. Both are supported by open-source libraries written in C, with bindings for several other languages. Both are claimed to give substantial speed advantages over DBMS systems, and even provide some SQL-like functionality for selective retrieval of required subsets of data.

However, the definitions for both formats are long and complex, and users are discouraged from using these directly. Instead, the use of 'standard' libraries is promoted. However, this leads to a number of problems⁴. NetCDF shares most of the same issues. The following sections are abbreviated quotations from this blog:

8.1.3.1 Single implementation

*The **HDF5 specification is very complex and low level**. It spans about **150 pages**. In theory, since the specification is open, anyone can write their own implementation. In practice, this is so complex that **there is a single implementation, spanning over 300,000 lines of C code**. The library may be hard to compile on some systems. There are wrappers in many languages, including Python. They all rely on the same C library, so they all share the bugs and performance issues of this implementation. Of course, the wrappers can add their own bugs and issues.*

*The code repository of the reference implementation is hard to find. It looks like there is an **unofficial GitHub clone of an SVN repository**. There are no issues, pull requests, little documentation, etc. just a bunch of commits. To understand the code properly, you have to become very familiar with the 150 pages of specification.*

Overall, using HDF5 means that, to access your data, you're going to depend on a very complex specification and library, slowly developed over almost 20 years by a handful of persons, and which are probably understood by just a few people in the world. This is a bit scary.

8.1.3.2 Corruption risks

*Corruption may happen if your software crashes while it's accessing an HDF5 file. Once a file is corrupted, all of your data is basically lost forever. **This is a major drawback of storing a lot of data in a single file, which is what HDF5 is designed for**. Users of our software have lost many hours of work because of this. Of course, you need to write your software properly to minimize the risk of crashes, but it is hard to avoid them completely. Some crashes are due to the HDF5 library itself.*

² NetCDF format specification: <http://www.unidata.ucar.edu/software/netcdf/netcdf/File-Format-Specification.html>

³ HDF5 format specification: <https://support.hdfgroup.org/HDF5/doc/H5.format.html>

⁴ Problems with HDF5 are discussed here: <http://cyrille.rossant.net/moving-away-hdf5/>

8.1.3.3 Various limitations and bugs

Once, we had to tell our Windows Python language users to downgrade their version of `h5py` because a segmentation fault occurred with variable-length strings in the latest version. This is one of the disadvantages of using a compiled library instead of a pure Python library. There is no other choice since the only known implementation of HDF5 is written in C.

UTF-8 support in HDF5 seems limited, so in practice you need to rely on ASCII to avoid any potential problems.

There are few supported data types for metadata attributes. In Python, if your attributes are in an unsupported type (for example, tuples), they might be silently serialized via pickle to an opaque binary blob, making them unreadable in another language like MATLAB.

A surprising limitation: **as of today, you still can't delete an array in an HDF5 file**. More precisely, you can delete the link, but the data remains in the file so that the file size isn't reduced. The only way around this is to make a copy of the file without the deleted array (for example with the `h5repack` tool). This is problematic when you have 1TB+ files. The upcoming HDF5 1.10 promises to fix this partially, but it is still in alpha stage at the time of this writing.

8.1.3.4 Performance issues

Since HDF5 is a sort of file system within a file, it cannot benefit from the smart caching/predictive strategies provided by modern operating systems. This can lead to poor performance.

If you use chunking, you need to be very careful with the chunk size and your CPU cache size, otherwise you might end up with terrible performance. Optimizing performance with HDF5 is a rather complicated topic.

In conclusion, we found out the hard way that **HDF5 may be quite slower than simpler container formats, and as such, it is not always a good choice in performance-critical applications**. This was quite surprising as we (wrongly) expected HDF5 to be particularly fast in most situations. Note that performance might be good enough in other use-cases. If you consider using HDF5 or another format, be sure to run detailed benchmarks in challenging situations before you commit to it.

8.1.3.5 Poor support on distributed architectures

Parallel access in HDF5 exists but it is a bit limited and not easy to use. MPI is required for multiprocessing.

HDF5 was designed at a time where MPI was the state-of-the-art for high performance computing. Now, we have large-scale distributed architectures like Hadoop, Spark, etc. HDF5 isn't well supported on these

systems. For example, on Spark, you have to split your data into multiple HDF5 files, which is precisely the opposite of what HDF5 encourages you to do [see also this document by the HDF Group].

By contrast, flat binary files are natively supported on Spark.

8.1.3.6 Opacity

You depend on the HDF5 library to do anything with an HDF5 file. What is in a file? How many arrays there are? What are their paths, shapes, data types? What is the metadata? Without the HDF5 library, you can't answer any of these questions. Even when HDF5 is installed, you need dedicated tools or, worse, you need to write your own script. This adds considerable cognitive overhead when working with scientific data in HDF5.

You can't use standard Unix/Windows tools like *awk*, *wc*, *grep*, Windows Explorer, text editors, and so on, because the structure of HDF5 files is hidden in a binary blob that only the standard *libhdf5* understands. There is a Windows-Explorer-like *HDFView* tool written in Java that allows you to look inside HDF5 files, but it is very limited compared to the tools you find in modern operating systems.

8.1.3.7 Philosophy

HDF5 encourages you to put within a single file many data arrays corresponding to a given experiment or trial. These arrays are organized in a POSIX-like structure.

One can wonder why not just use a hierarchy of files within a directory. Modern file systems are particularly complex. They have been designed, refined, battle-tested, and optimized over decades. As such, despite their complexity, they're now very robust. They're also highly efficient, and they implement advanced caching strategies. HDF5 is just more limited and slower. Perhaps things were different when HDF5 was originally developed.

If you replace your HDF5 file by a hierarchy of flat binary files and text files, as described in the previous section, you obtain a file format that is more robust, more powerful, more efficient, more maintainable, more transparent, and more amenable to distributed systems than HDF5.

The only disadvantage of this more rudimentary container format I can think of is portability. You can always zip up the archive, but this is generally slow, especially with huge datasets. That being said, today's datasets are so big that they don't tend to move a lot.

When datasets are really too big to fit on a single computer, distributed architectures like Spark are preferred, and we saw that these architectures don't support HDF5 well.

8.1.4 Conclusions

The promise of standard data interchange formats like NetCDF and HDF5, of speed advantages when processing very large data sets, appears to be greatly outweighed by the problems identified above. These formats have been used effectively, but the underlying software libraries are opaque, and the user base is much smaller than for commonly used database management systems, so there is a smaller pool of expertise available to assist when problems arise.

Even though the data structures needed for UNEXMIN are not excessively complex, there are requirements for both merging and selective retrieval of data sets, which can be coded very easily using SQL. Standard RDBMSs are now used extensively and have been optimised for efficiency in processing either small or very large databases.

The recommendation therefore is that a text-based interchange format (like CSV) be used where necessary, such as in conversion from sensor data to a common form, or in export of data from databases for processing by applications software where an ODBC library is not available, and that a standard relational DBMS should be used for long-term storage and management of UNEXMIN data.

8.2 Database Management System

There is a wide choice of database management systems available, at varying prices (ranging from free, open-source, upwards) and with varying capabilities. The majority are developed for a commercial market, and have limited numerical processing capabilities. Currently there are three main types of DBMS: relational⁵, object-oriented⁶, and network or tree-structured ("CODASYL"⁷).

- Relational DBMSs are based on concepts originally developed by E.F.Codd in the 1970s, and use flat tables consisting of rows (tuples) and fields (attributes), linked through special attributes identified as key fields.
- Object oriented DBMSs are based on the concept of an 'object' and are intimately linked to object-oriented programming languages.
- CODASYL DBMSs are almost invariably used for commercial applications with queries programmed in the COBOL language

Of the three types of system, by far the majority of scientific applications use relational DBMSs. This is mainly because of their flexibility (it is not necessary, in general, to define the full database schema in advance) and the ease of interfacing with a wide range of data processing applications including numerical scientific programs.

There is a wide choice of these available⁸. The most accessible is Microsoft Access⁹, which is a partial implementation of the relational model. Microsoft SQLServer¹⁰ and the open source MySQL¹¹ are probably the most widely used DBMSs on the Internet.

⁵ Description of the relational database concept: https://en.wikipedia.org/wiki/Relational_database_management_system

⁶ Description of the object-oriented database concept: https://en.wikipedia.org/wiki/Object_database

⁷ Description of the Codasyl database concept: <https://en.wikipedia.org/wiki/CODASYL>

⁸ https://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems

Two leading open-source DBMSs which should be considered are PostgreSQL¹² and Firebird¹³ (originally Borland Interbase). Of the many proprietary DBMSs to be considered, probably the leading contender is Oracle¹⁴.

A preliminary comparison of these different systems suggests that PostgreSQL may be the preferred option, but further investigation will be needed. PostgreSQL has an extension "PostGIS¹⁵" for spatial data, and also a number of developers have added point cloud processing capabilities^{16 17 18 19 20}. The Firebird DBMS, although it has a good reputation for robustness and speed, does not appear to have similar spatial data handling capabilities.

8.3 Table structure

All data will be held in flat tables in which all rows carry the same types of data in the same set of fields. This follows the relational model. For pure relational databases, the order of rows is immaterial. In UNEXMIN, the order of data from sensors and other data recording units on UX-1 will be fixed (ascending timestamp values) but to make the order explicit, a record (row) number should also be added when the database table is created, as an additional primary key field.

For point cloud data - in which points are not uniquely related to timestamps, a unique key (point number) will be generated on construction of the database table. For derived or computed data, the order may differ from this: for example, point cloud data may be sorted by spatial coordinate values or octree index values.

8.4 Codd flags

In most commercial database management systems, missing data elements are represented by a "NULL" code. However, there are many possible reasons for such missing data. It may simply be an unknown value, or it may be an artifact of a database process such as a table 'join' or 'union' procedure (Codd, 1). When data are present, also, these may be real data items as supplied from an external source, or they may be computed within the database.

Edgar Codd (1), the originator of the relational database model, proposed to use a special flag to identify the reason for absence of a data item. Such a flag could also be used to indicate relative reliability of a data value, or whether a data value is real or computed. In the RCI VMINE system, and one or two commercial systems, there is

⁹ <https://products.office.com/en-gb/access>

¹⁰ <https://www.microsoft.com/en-gb/cloud-platform/sql-server>

¹¹ <https://www.mysql.com/>

¹² <https://www.postgresql.org/>

¹³ <http://www.firebirdsql.org/>

¹⁴ <https://www.oracle.com/database/index.html>

¹⁵ <http://postgis.net/features/>

¹⁶ <https://github.com/pgpointcloud/pointcloud>

¹⁷ <http://www.oslandia.com/pages/postgis-pointcloud-en.html>

¹⁸ <http://gis.stackexchange.com/questions/57031/huge-point-cloud-laser-data-in-postgis-storing-and-processing-it>

¹⁹ <http://lanyrd.com/2013/foss4gna-2013/schqtd/>

²⁰ <http://s3.cleverelephant.ca/foss4gna2013-pointcloud.pdf>

provision for a Codd flag to be included automatically. In other systems, it is in general possible to set up an additional field (where needed) for each data field, to hold the Codd flag. In UNEXMIN, it is particularly important to identify whether a data item is a real observation or is computed, for example by interpolation of data recorded at different time intervals as explained in 8.5 below.

The standard flags defined by Codd are:

- "A" for missing but applicable (data value would be valid if supplied)
- "I" for missing and inapplicable (data value cannot exist: this is an artifact of the database functionality)

In either of these cases, the content of the data value field itself should be ignored as it is considered to be missing or 'null'.

It is proposed to add a new flag:

- "E" for an estimated or computed value, estimating the data value which 'should' be present

This will make it easy to distinguish between real recorded data and estimated values generated purely for post-processing.

Table 8.4.1 Codd flags

Data element	Codd flag	Meaning
Actual value		Value as recorded by instrument
"Null"	A	Absent: missing data element
"Null"	I	Inapplicable: an artifact of database processing
Estimated value	E	Computed estimate of value for this tuple

In any DBMS, the SQL queries can be framed in such a way as to test the Codd flag before accessing the data value. A DBMS that contains Codd flags may be considered as an "open-world" database management system in contrast with the normal "closed-world" database management systems used in business applications (Henley, 3,4).

8.5 Timestamp interpolation

The navigation/attitude system and all sensors will take their timestamps from the UX-1 central clock. For video recording, these timestamps will define recording start and stop times, while a local clock within each camera will define the times for each frame or image. In the conversion process, these local times will be translated to UX-1 central times using the start and stop timestamps.

In general, it cannot be assumed that all sensors will record data at exactly the same clock ticks: there will be offsets as illustrated in Figure 8.5.1.

In order to link the multiple data streams, it is necessary to carry out a database union of timestamps from all sensors. An outer join of the data tables will yield a table with many 'missing data' holes (the left-hand table in Figure 8.5.1). These holes may be filled with computed estimates as in the right-hand table. In order to distinguish these estimates from original data, it is proposed to use a Codd mark 'E'. Computation of the estimates should be by simple linear interpolation, thus:

$$a(E)_{ti} = a_{t1} + (a_{t2}-a_{t1}) (t_i-t_1) / (t_2-t_1) \quad \dots\dots(1)$$

where a_{t_2} and a_{t_1} are the sensor data values at time t_2 and t_1 respectively, and $a(E)_{t_i}$ is the estimated value at time t_i where $t_2 > t_i > t_1$

Figure 8.5.1. Illustration of mismatched recording times

DATA AS RECORDED				DATA PLUS INTERPOLATED ESTIMATES						
Time	Navigation	Sensor A	Sensor B	Time	Navigation	N-Codd	Sensor A	A-Codd	Sensor B	B-Codd
t1	XXX	AAA		t1	XXX		AAA		b	
t2		AAA	BBB	t2	x	E	AAA		BBB	E
t3	XXX			t3	XXX		a	E	b	E
t4			BBB	t4	x	E	a	E	BBB	
t5	XXX	AAA		t5	XXX		AAA		b	E
t6				t6	x	E	a	E	b	E
t7	XXX			t7	XXX		a	E	b	E
t8		AAA	BBB	t8	x	E	AAA		BBB	
t9	XXX	AAA		t9	XXX		a	E	b	E
t10		AAA		t10	x	E	AAA		b	E
...

The 'XXX', 'AAA', and 'BBB' represent real recorded values, and would have blank associated Codd flags.

The 'x', 'a', and 'b' represent the interpolated values and would have associated Codd flags of 'E' (for 'estimated').

8.6 Table linkage

The principal method of linking data from different tables will be through a common key field: in most cases this will be the time stamp. This linkage will use standard database methods.

Where the timestamp is insufficient to provide unique linkages between data elements - such as for pixels in different images created at the same time - it may be necessary to use computational methods to combine data sets, for example by first calculating the exact x,y,z location of each pixel in each image. These computational methods are unlikely to be supplied as standard in any DBMS, and would need to be programmed by UNEXMIN participants.

8.7 Search, Selection and Retrieval of Data

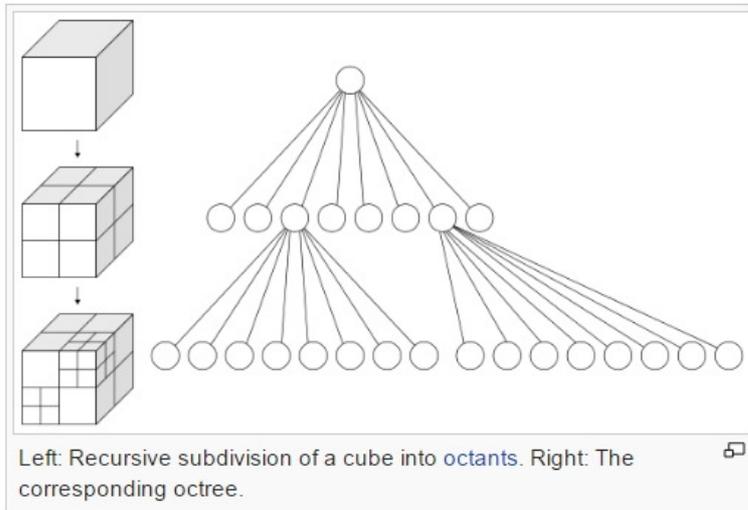
8.7.1 Search and selection methods

Usually, although the primary key will be the timestamp, this is not the field which will be used for searching the database unless it is known where UX-1 was located at a particular time. The most common search parameters will be locational.

Efficient location search in a 2 or 3 dimensional space is a well-known computational problem. The least efficient solution is 'brute force' searching of every record of a table. A number of more efficient methods have been developed. Of these, the most general

and probably the simplest to implement is octree indexing²¹. An octree index to a data table is generated by identifying the cube in which each data point lies, within a hierarchic set of cubes of progressively smaller dimensions (Figure 8.7.1)

Figure 8.7.1 Octree encoding



8.7.2 Data Retrieval

Data sets retrieved from one or more database tables are themselves held in tables. Depending on the database implementation and user requirements, these may be virtual tables that are created only in memory and lost when no longer required for immediate processing tasks, or they may be permanent ('persistent') tables which can be retained within the database or can be exported for processing by applications software. Most DBMSs allow virtual tables to be saved as persistent tables for future use.

²¹ Octree encoding method explained: <https://en.wikipedia.org/wiki/Octree>

9 References

- (1) Codd, E.F., 1990. The Relational Model for Database Management, Version 2. *Addison-Wesley, Reading, Massachusetts, 538pp.*
- (2) Date, C.J., Hugh Darwen, and Nikos A. Lorentzos, 2003. Temporal Data and the Relational Model. *Morgan Kaufmann, Amsterdam, 422pp.*
- (3) Henley, S., 2005. The Man Who Wasn't There: the problem of partially missing data: *Computers & Geosciences v. 31, no.6, pp. 780-785*
- (4) Henley, S., 2006. The problem of missing data in geoscience databases: *Computers & Geosciences, v.32, no.9, pp1368-1377.*

10 ANNEX A: Summary of Data Tables

Data type	Primary key	Other fields
UX-1 location	Timestamp	X,Y,Z
UX-1 attitude	Timestamp	Pitch_angle, Yaw_angle, Roll_angle
UX-1 speed, acceleration	Timestamp	dx,dy,dz speed in 3 directions d2x,d2y,d2z, accelerations Both can be estimated if necessary from successive locations at different timestamps
Camera images from LED	Timestamp	Camera ID, X,Y,Z, Image_filename
Colour Camera images	Timestamp	Camera ID, X,Y,Z, Image_filename
Point cloud extracted from laser stripes	Timestamp	X,Y,Z (,Intensity,R,G,B), Stripe_No, Point_No, Mission_ID, UX-1 location x,y,z, UX-1 attitude angles, laser stripe angles, cumulative distance and time, additional fields from multispectral and colour cameras
Acoustic camera images	Timestamp	X,Y,Z, Image filename?
Thermometer	Timestamp	Temperature
Water Pressure	Timestamp	Water Pressure
Conductivity	Timestamp	Conductivity
pH	Timestamp	pH
Magnetic field	Timestamp	Field strength, Direction (dX,dY,dZ)
Gamma Ray	Timestamp	Count, Wall_Distance
Multispectral camera	Timestamp	Camera ID, LED_ID, Frame number
Colour camera fluorescence images	Timestamp	Camera ID, X,Y,Z, Image_filename
Sub-bottom sonar	Timestamp	Reflection-1,2,3,4... times using the SEG-Y format
Battery status	Mission_ID Timestamp	Charge level
Mission command file	Mission_ID Timestamp	Command
Mission actions file (log)	Mission_ID Timestamp	Action taken
Mission_Metadata	Mission_ID	Start_Time, End_Time, Description
Camera Metadata	Mission_ID	Camera properties including location in UX-1, view direction relative to UX-1 coordinate system, distortion correction parameters.....
Sensor Metadata	Mission_ID	Sensor properties