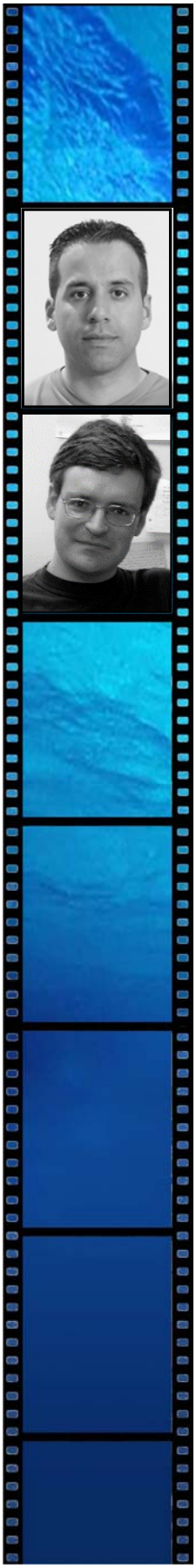
Article

HUDDL:
THE HYDROGRAPHIC UNIVERSAL DATA DESCRIPTION LANGUAGE

G. MASETTI and B. CALDER
Center for Coastal and Ocean Mapping & Joint Hydrographic Center
University of New Hampshire, Durham, New Hampshire, USA



Abstract

Since many of the attempts to introduce a universal hydrographic data format have failed or have been only partially successful, a different approach is proposed. Our solution is the Hydrographic Universal Data Description Language (HUDDL), a descriptive XML-based language that permits the creation of a standardized description of (past, present, and future) data formats, and allows for applications like HUDDLER, a compiler that automatically creates drivers for data access and manipulation. HUDDL also represents a powerful solution for archiving data along with their structural description, as well as for cataloguing existing format specifications and their version control. HUDDL is intended to be an open, community-led initiative to simplify the issues involved in hydrographic data access.

Résumé

Etant donné que de nombreuses tentatives d'introduction d'un format universel de données hydrographiques ont échoué ou n'ont que partiellement été couronnées de succès, une approche différente est proposée. Notre solution est le langage hydrographique universel de description des données (HUDDL), un langage descriptif basé sur la norme XML qui permet une description normalisée des formats de données (passés, actuels et futurs) et à partir duquel peuvent être développées des applications comme HUDDLER, un compilateur qui crée automatiquement des pilotes pour l'accès et la manipulation des données. HUDDL constitue également une solution puissante pour l'archivage des données avec leur description structurelle, ainsi que pour le catalogage des spécifications de format actuelles et le contrôle de version. HUDDL se veut une initiative communautaire ouverte pour résoudre les difficultés d'accès aux données hydrographiques.

Resumen

Dado que muchos de los intentos de introducir un formato universal de datos hidrográficos han fracasado o han sido sólo un éxito parcial, se propone un enfoque diferente. Nuestra solución es el Lenguaje Universal de la Descripción de los Datos Hidrográficos (HUDDL), un lenguaje descriptivo basado en el XML, que permite la creación de una descripción normalizada de formatos de datos (pasados, presentes y futuros), y que permite aplicaciones como HUDDLER, un compilador que crea automáticamente controladores para el acceso a y la manipulación de datos. HUDDL también representa una solución muy potente para el archivo de datos, junto con su descripción estructural, así como para la catalogación de las especificaciones de formato existentes y el control de sus versiones. HUDDL pretende ser una iniciativa abierta, dirigida por la comunidad para simplificar las cuestiones relacionadas con el acceso a los datos hidrográficos.

1. Introduction

Data acquired during a hydrographic survey may be stored in a number of different formats. Essentially, every manufacturer has developed their own format specification. Ongoing development during the lifetime of existing or new acquisition systems usually requires a sequence of file format releases. Trying to keep abreast of all of the different formats, their change-points, and idiosyncrasies, can be a time-consuming and problematic endeavor for anyone who has to read multiple different data formats, or deals with archival data.

One potential solution to this problem is to convert each data type into a 'universal' format for archive or processing. Many of the attempts to introduce such a format for hydrographic data have however failed or have been only partially successful. This is because such formats either have to simplify the data to such an extent that they only support the lowest common subset of all the formats covered, or they attempt to be a superset of all formats and quickly become cumbersome. Neither choice works well in practice.

This issue is exacerbated by a lack of a common repository for hydrographic data formats. Each manufacturer documents their own format in a different way, and often in different locations, with different release schedules, and, often, only partially consistent release announcements. To find details of a particular data format requires a user to navigate many different websites - often driven by having attempted and failed, to read survey lines in a new variant of the format. This also means that data conversion parsers or tools for different data formats are only available for a limited number of format pairs (some for free, the largest part with a cost).

One of the biggest (and negative) consequences of the current situation is that each data handling application has its own data parsers (coded mostly from scratch) for every supported data format. These parsers must be kept up to date. This is a significant resource soak that could be reduced, and entails the danger of allowing variant data content interpretations in different software packages. Survey data access becomes more complicated if the source files are stored in legacy data formats, where negotiation of multiple versions of even one format may be required if historical trend analysis is the primary goal. A useful solution to survey data access should offer access to mixed-format historical data, providing a mechanism to describe data collected and archived in sometimes 'exotic' data formats (e.g., developed by defunct manufacturers). Solving this issue implies the definition of a reliable way to access the data collected today by our descendants, with obvious advantages in the adoption of these methods by hydrographic data archiving centers.

Our long-term solution to this issue is a descriptive language flexible enough to describe past, existing, and likely future hydrographic data formats: the Hydrographic Universal Data Description Language (HUDDL). This can also be readily extended to convert new data format concepts that might appear in the future. The key point of the HUDDL approach, is to describe the existing formats as they are, rather than define another chimeric format able to encapsulate the information present in all the existing data formats (with all the related semantic issues in case a conversion is attempted) (Masetti and Calder, 2014).

A HUDDL File Description (HFD) is a machine-readable description of the content of a data format that can be used in multiple ways. For example, it is possible to use an HFD for automatic generation of data drivers, validation of the content of survey lines claiming to be consistent with a particular format release, recovery of partial information from corrupted data, storage and reference of the description of how data are organized in a given data format, or for incremental update of data format specifications. Since HFDs are implemented as XML files, they can also be uniformly and consistently converted to produce documentation in different formats (e.g. HTML). A simple metadata link to an online repository of HFDs represents a robust way to uniquely identify the data organization.

A uniform collection of data format descriptions represents a powerful resource for data

format conversion, a step that has been described as "the soft underbelly of processing scientific data," consuming immense amounts of time for those who work in a heterogeneous software environment (Georgieva et al., 2009). At the same time, the collection of information in one place, and the simplicity of the descriptive mechanism, provides a tool for inspiration, definition, and testing of new data formats and updated releases before being made public to the hydrographic community.

Given the many different fields of interest, we believe that the final overall result of HUDDL will be to drastically reduce the resource burden focused on accessing the information stored in hydrographic data formats.

2. The Description Language

Language requirements and features development

A number of different solutions have been developed to describe data files over the last 30-40 years. None of them provides the full set of requirements of a hydrographic format description language (Masetti and Calder, 2014). In essence, the language must be:

- Readily adoptable (e.g., using XML-based syntax, familiar to a large number of potential users);
- Well-maintained (some languages, e.g. ESML (Ramachandran et al., 2004), do not have any time schedule for standard development);
- Widely accepted (there is a common lack of this requirement in any of the existing solutions, which may represent a weakness of available methods);
- Flexible, with a low-cost implementation (e.g., JSON requires data conversion in different structures (Nurseitov et al., 2009), while manufacturers likely want to maintain their own data formats);
- Based on a simple syntax, while still retaining enough expressivity to describe hydrographic binary data formats (the intent, for example, of DFDL (Powell et al., 2011; Westhead and Bull, 2003) to be universal increases the overall complexity); and
- Available with an open-source and open-community implementation (the hydrographic community is narrower than the communities targeted by each existing solution, which may speed up the adoption and the contribution to develop a working approach). For instance, Protocol Buffers, while open source, is not open in the development process (Kaur and Fuad, 2010; Varda, 2008).

None of the existing proposed data description methods explicitly focus on hydrographic use cases, which are dominated by data streams of sensor data, and arrays and lists of floating point numbers. The more structured nature of these data streams allows for some simplification in the implementation of a data description. Useful features from each of the existing solutions have, however, been adopted into HUDDL.

The HUDDL development was focused on a language that can:

- Provide a common set of many basic validation and computation functionalities;
- Explain the structure of binary files to users (readability);
- Automatically generate a parser directly from a schema;
- Provide a convenient basis for building arbitrary transformations between binary data formats (data conversion/ transformation) and data file indexing; and
- Extend applications with content-aware functionalities (e.g., tools that can inspect any binary file given a schema, file comparison, etc.).

HUDDL describes the physical representation, the overall structure and the semantics of various existing data formats used in the hydrographic field. The language relies on a set of core schemas that make available various description tools such as array data structures and primitive data types. New elements may be added each time a new unknown structure is encountered. This solution

was preferred to an attempt to *a priori* define all the possible required elements (e.g., rare middle-endianesses or some uncommon IEEE formats for representing floating point numbers) and their exponentially growing combination. Doing otherwise might make HUDDL-aware technology too complex and difficult to adopt.

HUDDL is focused on describing most types of hydrographic data formats in a simple syntax, rather than attempting to be a generic and arbitrary spatial acquisition format. The main reason for this is the desire to maintain implementation libraries that are lightweight and as simple as possible, providing order rather than adding complexity to the existing scenario of hydrographic data formats.

This solution also provides an inexpensive but robust way to deal with many legacy data formats. When required to access old datasets in an arbitrary binary format, an *ad hoc* HFD may be created that can deal with the particular vagueness of some legacy formats or some rare variant implementations.

Conceptual and Physical Data Modelling

HUDDL was developed as a community-specific, format-oriented data description language. These characteristics provide a certain level of simplification since existing data formats are different answers to the same problem: fast storage of data acquired in real-time. All of the data format specifications targeted by HUDDL have three components which formed the requirements for the abstract conceptual model and physical implementation reported here:

- The semantic: what a given value collected in the data format actually means (e.g., the unit of measure);
- The physical description: how the bits and the bytes are stored on disk (e.g., endianess, memory alignment); and
- The logical structure: what data structures are used to organize the data (e.g., an array)

The analysis of existing data formats suggested a natively tree-structured model: a top-level container, called a 'Schema' that may hold several different descriptions of data formats, each of which has both a 'Prolog' and a 'Content' element (***Figure 1***). The 'Prolog' represents a collection of metadata related to the described data format, such as the organization that created it, the personnel responsible for its maintenance, or the history of releases (***Figure 2***). This information is required to create a homogeneous and consistent documentation for different data formats, although the extent to which it is implemented can vary between formats – the better the information, the more complete, and useful, the documentation.

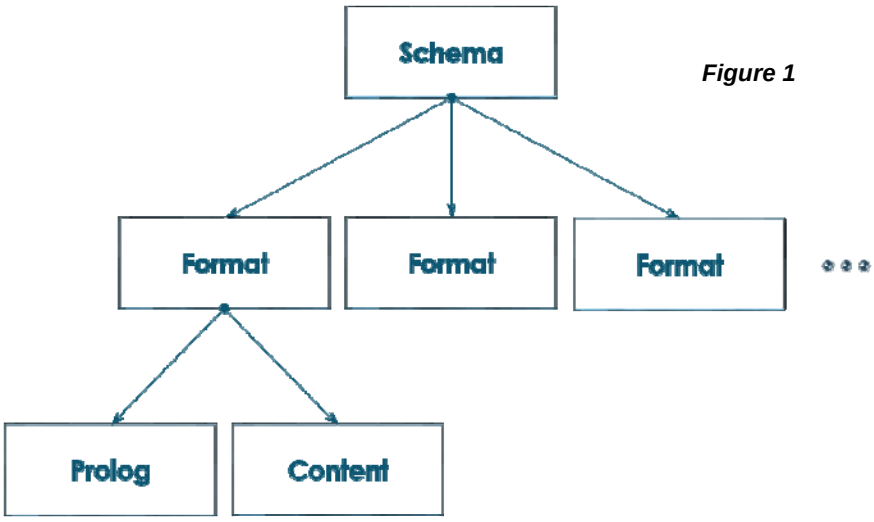


Figure 1

Figure 1 *: Top-level elements of the HUDDL format model. A Schema provides the ability to host more than one format, each of which contains a prolog to provide metadata on the format, and then a content description providing the details of data's format.*

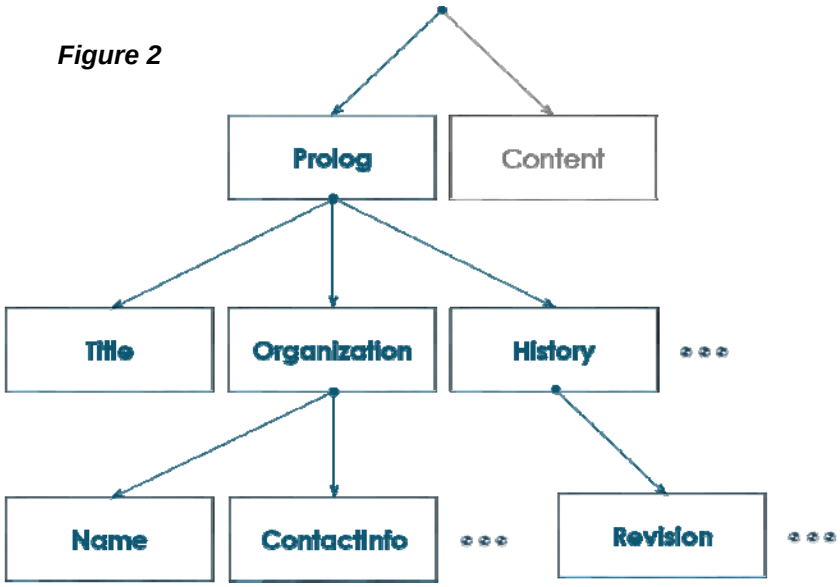


Figure 2

Figure 2 *:* Example of elements present in the 'Prolog' branch of the HUDDL format model. Any amount of metadata on the data format can be provided. This information is not strictly required for some uses of the HFD (e.g., to generate source code to access the data), but has significant benefits when documentation is being generated.

The 'Content' branch is used to describe both the structure and the format of a binary data file in a platform-independent way. This branch has three main containers (***Figure 3***):

- Blocks: which may contain any number of fields and available data structures (e.g., 2D array) (***Figure 4***). A 'Block' represents a logically related group of information elements committed to file at the same

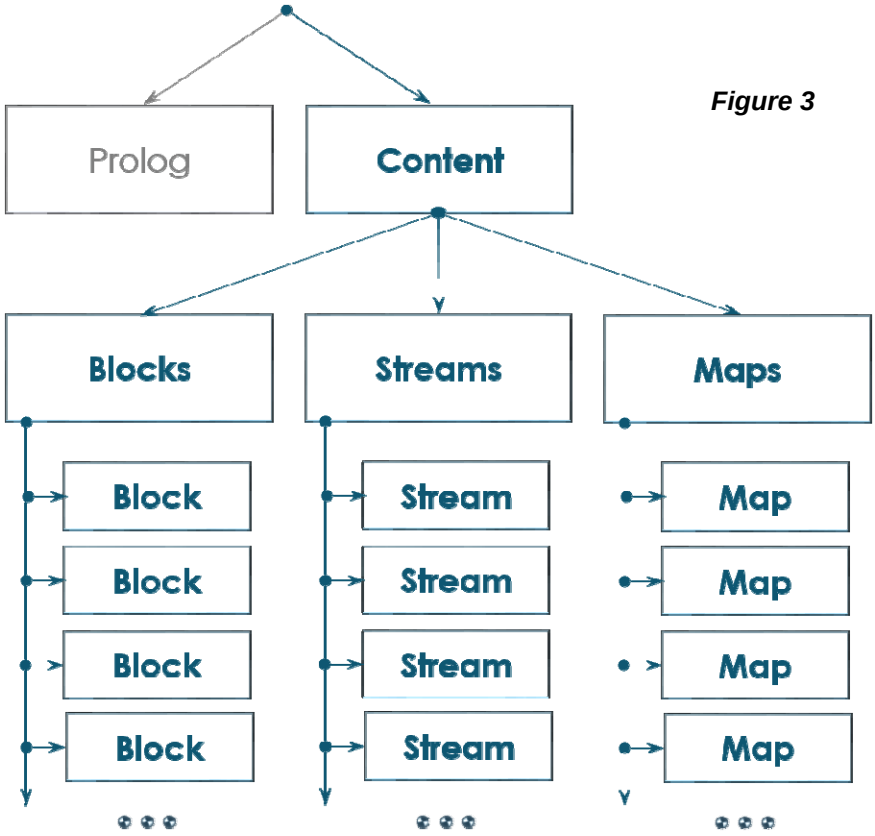


Figure 3

Figure 3 *: Four sections of the 'Content' branch of the HUDDL format model. Blocks represent a group of data elements written to file as a group (e.g., a single ping's worth of bathymetric data), while Streams represent a collection of Blocks that can be read together to provide in composite the description of a single version of a data format. Maps provide the means to link semantic representation to the data, for example, by providing physical units or the means to translate encoded values into physical units.*

time instant. A 'Field' is used as a basic value container (e.g., bytes, two- or four-byte integers, floating point numbers, etc.). In order to allow for reading of complex data structures, blocks can also contain other blocks, optionally as one- and two-dimensional arrays with either fixed or variable sizes defined in the preceding data block.

- Streams: which lists all the releases of a data format. Each release is represented by a 'Stream' containing the overall composition of a data file (***Figure 5***): a 'Header' which describes initial shared data fields present in all top-level blocks, a 'TopBlocks' list with all the blocks that can be encountered at the top level of the format, and an optional 'Tail' description, representing a common data element found at the end of all top-level blocks (e.g., a checksum). Thus, it is possible to have, in the same document, multiple streams that reflect different releases of the same data format, and hence updates are only required to be incremental (this characteristic makes them smaller, simpler, and faster than would be required for a single-release format description approach).
- Maps: for explicitly describing relations among fields and other available data structures for a given 'Stream' such as, adding sensor-specific semantic context.

Among the wide range of possible solutions for the physical implementation, the Extensible Markup Language (XML), with the support of strictly linked XML Schemas, was selected. XML provides a representation standard that is convenient because it is both human- and machine-readable, easily and quickly extensible, has wide adoption, and is a mature technology.

In HUDDL, XML is used to give a structural description of the contents of a file format (rather than the content of a particular file). Coupling one or more of the proposed descriptive XML schemas with a given hydrographic dataset, as metadata, provides a detailed definition on how data have been actually stored. For the structural representations HUDDL follows the main data structures (e.g., data streams) present in the most used hydrographic data formats (eXtended Triton Format, Generic Sensor Format, Kongsberg EM series, etc.) as well as the work done for XDR and BinX (variable and fixed length arrays, simple structures, strings, unions, etc.) (Eisler, 2006; Kongsberg Maritime AS, 2013; SAIC, 2012; Triton, 2013; Westhead and Bull, 2003).

XML already has a key role in the representation of metadata associated with a hydrographic dataset since it represents the accepted means to describe information related to the data collector, acquisition parameters, meteorological conditions, etc. At the same time, hydrographic applications that once were tightly-coupled and monolithic are now becoming more modular, with collaborating components spread across diverse computational elements (Calder, 2013). In such a distributed environment, open metadata systems are increasingly important and useful to communicate substantial amounts of structured data (Widener et al., 2001). The increasing popularity of XML in the field of marine science and engineering is also driven by its role in the ISO 19000 series metadata standard (Georgieva et al., 2009; Hua and Weiss, 2011; ISO, 2008; Yongguo et al., 2009).

While relatively simple, the implementation of the HUDDL Format Model is quite expressive. For example, since blocks can contain other blocks and arrays of blocks, a data unit which contains a header segment (e.g., the parameters for a given ping's depth detections), along with a record of the depths detected per beam, can be easily represented by, block for each detection, and a block that contains the header information as elemental fields, with an embedded 1D array of the detection blocks. The HUDDL Core Schemas also allow for variable length arrays (e.g., if the number of beams reported is variable per ping), for two dimensional arrays of fields or blocks, and other common features of typical hydrographic data formats. It is therefore typically a fairly simple matter to translate a given data format into a HUDDL description given the appropriate documentation.

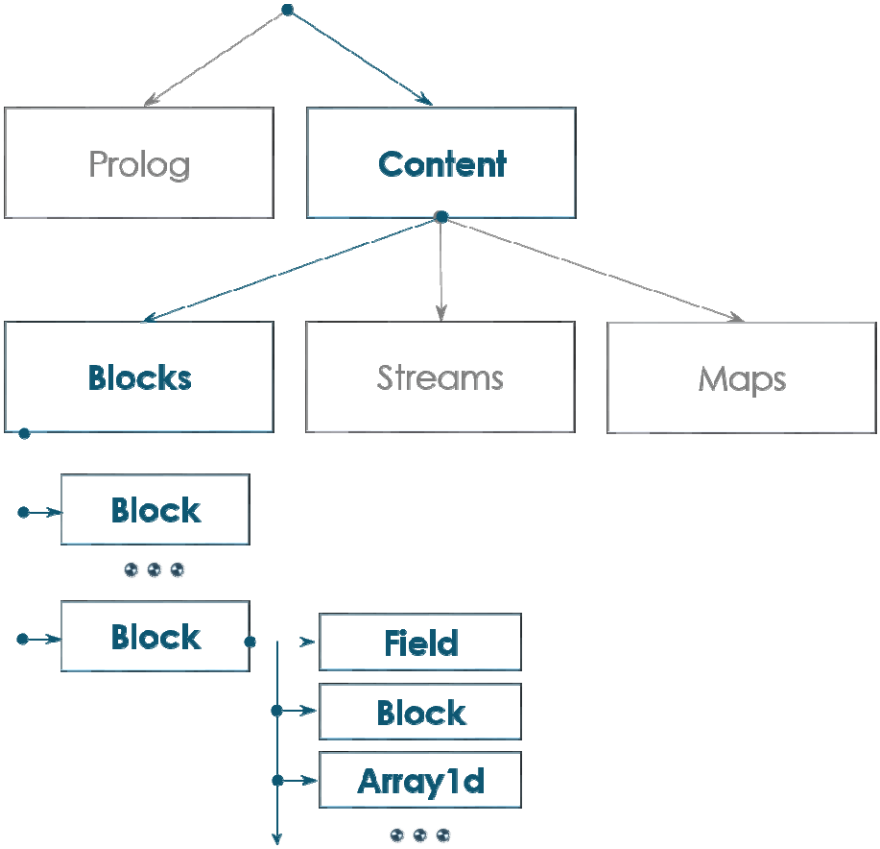


Figure 4 *: 'Blocks' internal structure. Each block may contain any number of basic data objects (e.g., integers of different signedness and sizes, floating point values, etc.) as Fields, other Blocks to provide for composite and complex data types, and 1D and 2D fixed and variable arrays of Fields or Blocks.*

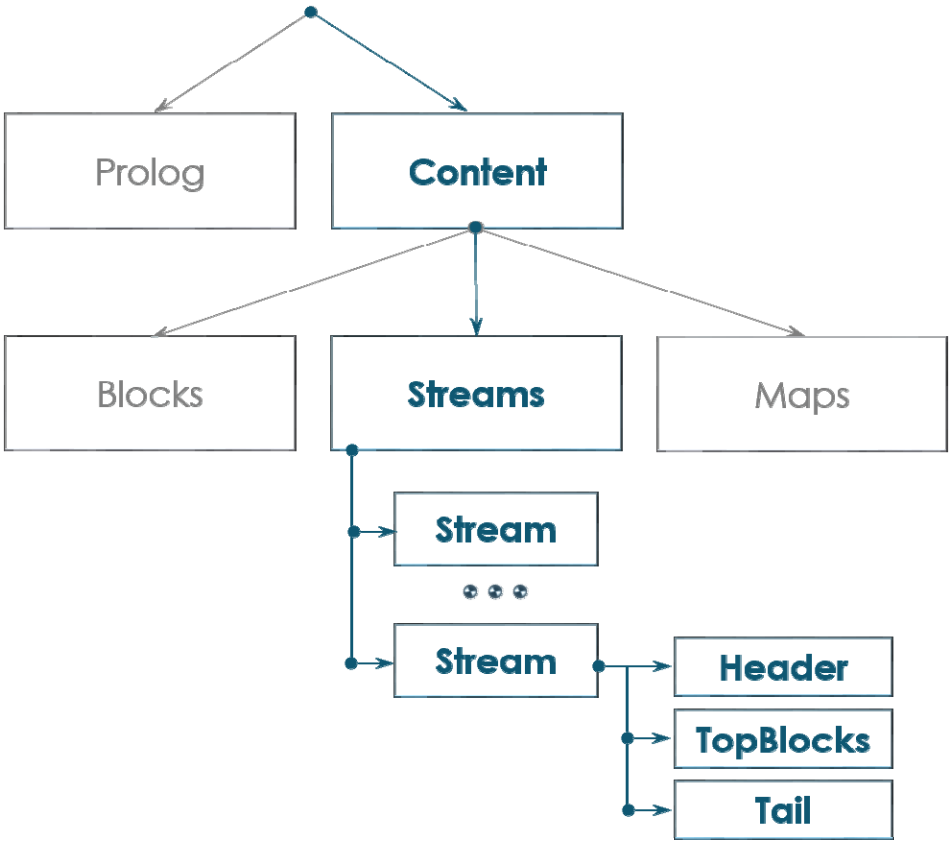


Figure 5 *: 'Streams' internal structure. Each Stream consists of a special Block that appears at the start of each data object in the file (typically, this contains a length and identification integer to indicate what data is being stored), along with a list of all of the Blocks that can occur at the outer-most level of the data file (TopBlocks), and an optional Tail block for a common data structure that appears at the end of each Block (e.g., a* checksum).

The HUDDL Format Descriptions are based on the HUDDL Core Schemas (which are also XML schemas) so that they describe the physical and logical implementation of a data format. They can also be used to create documentation to explain the semantic meaning to a human through the use of a suitable XSLT (Extensible Stylesheet Language Transformations) translator, which are widely available. The latest generation of web browsers is able to use XSLT stylesheets directly, so that XML documents can be viewed easily by a human (e.g., HTML pages, PDF files), as well as being understandable by machines. Extant tools for XML creation can also be used to structure HFDs. These tools are intelligent, disallowing invalid data entry, and suggesting that which is valid. Programs can also read HFDs through any of a variety of parsers. Some XML parsers are already built into programming languages (e.g., Java, Python), and there are a variety of external parsers (e.g., Xerces, libxml). HUDDL schemas can also be easily translated to other formats using XSLT.

There are also advantages in using a data-description language such as HUDDL versus using diagrams (e.g., UML). HUDDL is more formal than diagrams (leading to less ambiguous descriptions of data formats) and easier to understand (allowing software developers to focus on other issues instead of the low-level details of bit encoding). Also, there is a close analogy between the types used by HUDDL and a high-level language such as C/C++ or Python. Finally, the language specifications themselves are XML files that can be passed from machine to machine to perform on-the-fly data interpretation.

A web repository for HFDs was created at the Center for Coastal and Ocean Mapping (CCOM) to provide an initial safe and easy-to-check common point for data format specifications. Widely used systems (e.g., RSS, or an open-subscription mailing list) could assist in staying current with the last release of data formats for all of the interested players. The repository is part of a community-oriented website to access, catalogue, and disseminate hydrographic data formats resources and HUDDL-specific information that has been developed and is now publicly available

3. The Format Driver Compiler

HUDDLER is an implementation of one of the many advantages of having available machine-readable HUDDL Format Descriptions: a compiler that automatically creates drivers for data access and manipulation (Calder and Masetti, 2015).

HUDDLER implements the HUDDL-philosophy of constraining the description of the data format to the schema, so that the user has to touch the minimal amount of code to reflect any change in the data format specification (Masetti and Calder, 2014). Instead of having to change the user's application code directly to reflect the format changes, changes to the schema are translated automatically by HUDDLER into the library that represents the data format, and this can be readily automated in most software build systems. In practice, updating the software to support a new data format version is as simple as changing the schema and then recompiling the library or application, as appropriate, leaving the programmer to work on the application logic to use the new facilities added by the new version of the format.

The compiler is based on an XML parsing library that loads into memory the format description (frontend), and a code generator (backend) that creates code able to access the data in three different types of computer languages: procedural ANSI C, object-oriented C++, and multi-paradigm Python. The system is designed to admit other languages readily (e.g., Matlab).

The creation of a new format driver is structured in four steps (***Figure 6***):

- HFD validation: which automatically checks that the description follows the HUDDL Core Schemas;
- HFD parsing: which loads the format description into memory;
- Format processing: which performs additional checks on the format description and solves internal block and field cross-references; and
- Code generation in one of more of the available code generators.

Generating code directly in the target language allows the code generator to take advantage of particular language features that would simplify the generated code, or better express the idiomatic nature of the target language usage. However, particularly for some interpreted languages, performance issues dictate that it is preferable to automatically generate a language-specific wrapper around a C/C++ library. This was the approach followed for the Python backend. It would be possible to build a pure Python backend driver if required, for pedagogical purposes, but the performance would generally be sufficiently constrained as to make its practical application limited. As a common factor, the output code from the language-specific generators attempts to provide data types that are as transparent as possible in order to reduce the complexity of manipulating routines in the master application.

To better illustrate the simplicity and the potential of this approach, an example that accesses and plots attitude data from a real file is shown in ***Figure 7***. The left pane shows the part of a HUDDL Format Description used to describe the specific blocks containing attitude data, and the stream that provides access to them as top-blocks. The right pane displays the code snippets specifically created by HUDDLER to read the format version, which internally calls a helper function to retrieve the top-block containing the attitude measurements. Once the generated code is compiled, a simple script (***Figure 8, left pane***) can be used to import the HUDDLER-generated library and use the generated methods to open the data file, access the attitude data, and manipulate the data (e.g., to plot roll, pitch, heave and heading as shown in the ***right pane of Figure***). The Python script is a simple demonstration of the many advantages of HUDDL, since it provides a means to easily access hydrographic data taking advantage both of the flexibility and ease-of-use of Python and the speed of C code for data reading. The full working code for this example, and the con-

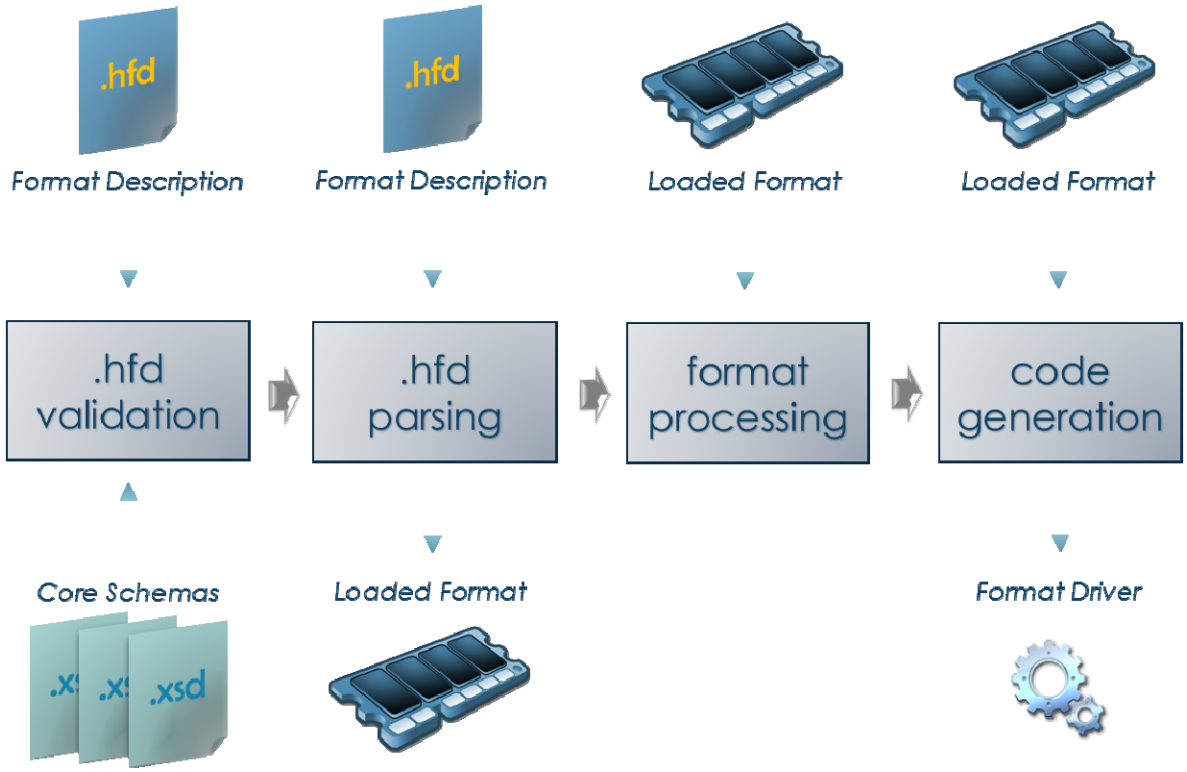


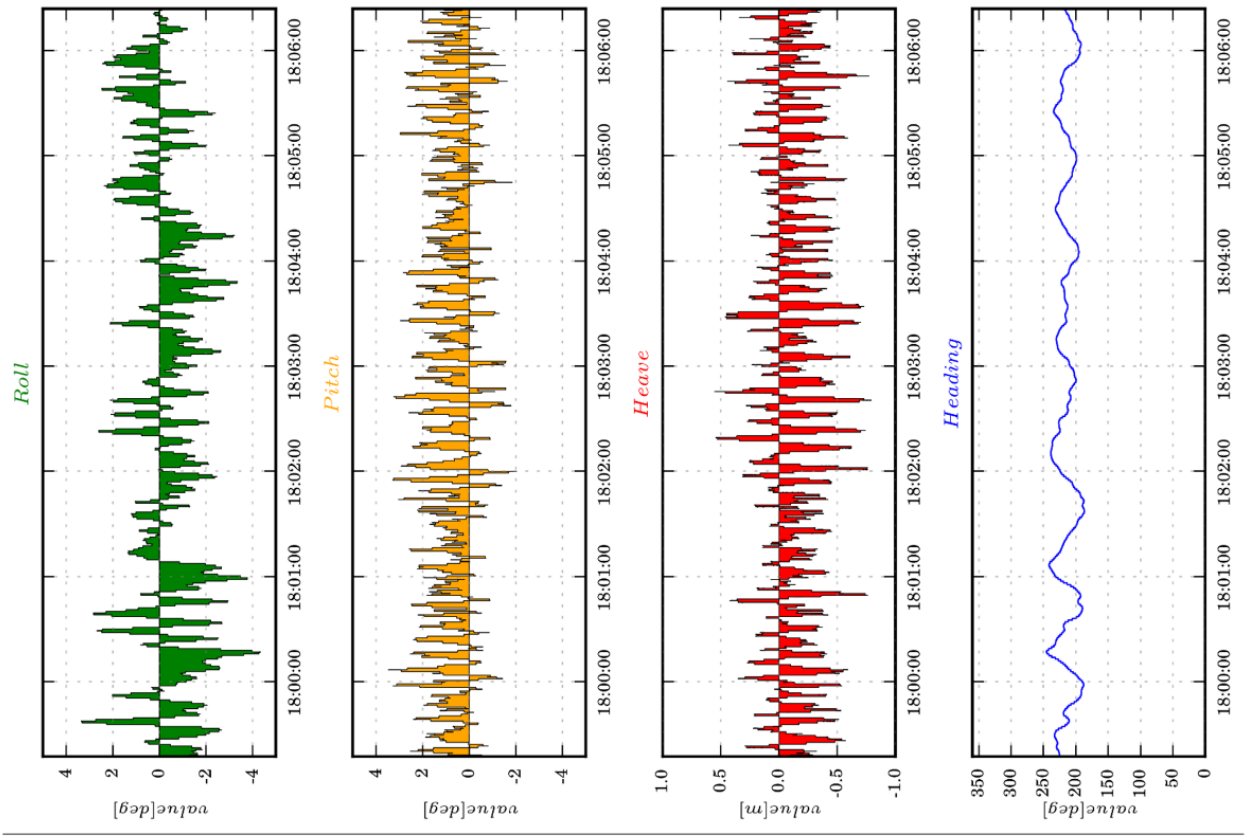
Figure 6 : *HUDDLER steps to create a new format driver: HFD validation, checking that the description follows the HUDDL Core Schemas; HFD parsing, loading the format description into memory; Format processing, performing additional checks on the format description and solving internal Block and Field references; Code generation in one of more of the available code generators.*

Figure 7 *: On the left pane, the part of a HUDDL Format Description that describes the specific blocks containing attitude data and the stream that provides access to them as top-blocks. On the right pane, the code snippets created by HUDDLER to read the specific format release and the top-block containing the attitude measurements.*

```
<?xml version="1.0" encoding="UTF-8"?>
<huddl:schema version="1.0.0">
  <huddl:format name="Kongsberg EM Series" scope="kng">

    <huddl:prolog>
      <huddl:title>Kongsberg EM Series formats</huddl:title>
      <huddl:organization>
        <huddl:name>Kongsberg Maritime AS</huddl:name>
      </huddl:organization>
      [… additional metadata as history and contact info …]
    </huddl:prolog>

    <huddl:content>
      <huddl:blocks>
        <huddl:block name="em96att">
          <huddl:field name="latency" type="huddl:u16"/>
          <huddl:field name="status" type="huddl:u16"/>
          <huddl:field name="roll" type="huddl:s16">
          [… additional fields as in format specs …]
        </huddl:block>
        <huddl:block name="em96attitude">
          <huddl:field name="counter" type="huddl:u16"/>
          <huddl:field name="serialnum" type="huddl:u16"/>
          <huddl:field name="numentries" type="huddl:u16"/>
          <huddl:vector1d name="attitude">
            <huddl:blockType>em96att</huddl:blockType>
            <huddl:sizeField>numentries</huddl:sizeField>
          </huddl:vector1d>
          [… additional fields as in format specs …]
        </huddl:block>
      </huddl:blocks>

      <huddl:streams>
        <huddl:stream revID="revR" scope="rev19">
          <huddl:header refBlock="emhdr" discriminator="type"/>
          <huddl:topBlocks>
            <huddl:topBlock refBlock="em96attitude"
                            alias="attitude" identifier="0x41"/>
          [… additional top blocks as in format specs …]
          </huddl:topBlocks>
          <huddl:tail refBlock="emtail"/>
        </huddl:streams>
      </huddl:content>

    </huddl:format>
  </huddl:schema>
```

```
/* Huddler - Automatically generated code. NOT manually modify! */
#include "kongsberg_em_series.h"

static bool read_em96attitude( FILE* f, em96attitude_t* data,
                               int* tot_read, int* tot_bytes )
{
  int bytes_read = 0;

  if(fread(&(data->counter), 2, 1, f) != 1) return false;
  bytes_read += 2;
  [… additional code to read block fields …]
  for(i = 0; i < (int) data->num_entries; ++i)
    if( !read_em96att(f, data->attitude+i, &bytes_read, tot_bytes) )
      return false;

  bytes_read++;
  *tot_read += bytes_read;
  return true;
}

HdlrErrorType read_kng_revR( FILE* f, kng_revR_t* data,
                             u32* n_read, kng_validator_t vldtr )
{
  bool valid = false;
  int bytes_read = 0, bytes_tot = -1, resynch_distance = 0;
  HdlrErrorType rc;

  while( !valid && (resynch_distance < 1024) ){
    bytes_read = 0;
    if(!read_emhdr(f, &(data->header), &bytes_read, &bytes_tot ))
      if( feof(f) ) { rc = HUDDLER_EOF_WHILE; }
      else { rc = HUDDLER_READ_ERROR; }
      return rc;
    }

  switch( data->header.type ){
  case 0x41:
    if( !read_em96attitude( f, &(data->datagram.attitude),
                            &bytes_read, &bytes_total ) )
      return HUDDLER_TOPBLOCK_READFAIL;
    data->id = (kng_revR_id)0x41;
    break;
  [… additional code to read all the topblocks …]
  default:
    data->id = (kng_revR_id)data->header.type;
    return HUDDLER_TOPBLOCK_UNKNOWN;
  }
  *n_read = bytes_read;
  return HUDDLER_TOPBLOCK_OK;
}
```

Figure 8 *: On the left pane, a simple script that imports the HUDDLER-generated library that provides all the methods to open and access the attitude data. On the right pane, the output generated by the Python script that can be used to quickly inspect the attitude data before manipulation and/or use in processing algorithms.*




```
#!/usr/local/bin/python
import kongsberg_em_series as kng
import numpy as np
import matplotlib

class KngData:
    """ container for Kongsberg attitude data """
    roll = np.zeros(0, dtype=np.float)
    [… create all the required empty numpy arrays …]

def read_attitude(data_header, data_content, kng_data):
    """ read and store a Kongsberg attitude datagram """
    nr_entries = data_content.attitude.numentries
    loc_roll = np.zeros(nr_entries, dtype=np.float)

    for i in range(nr_entries):
        loc_att = kng.em96att_getitem(data_content.attitude.att, i)
        loc_roll[i] = loc_att.roll / 100.0
        [… read the other attitude measures: pitch, heave, etc. …]

    kng_data.roll = np.append(kng_data.roll, loc_roll)

def plot_attitude(kng_data):
    """ plot and save the read Kongsberg attitude data """
    fig, (ax0, ax1, ax2, ax3) = plt.subplots(ncols=1, nrows=4)
    plot0 = ax0.fill_between(kng_data.time, kng_data.roll,
                             linewidth=0.3, facecolor='green')
    ax0.set_title('Roll')
    [… use Matplotlib to plot all the attitude arrays …]

def main():
    kng_data = KngData()
    testfile_path = '0001_20120606_175918_ShipName.all'
    ip = kng.fopen(testfile_path, 'rb')
    [… check if the file was successfully open …]

    abort = False
    while not abort:
        rc, n_read = kng.read_kng_revR(ip, data, n_read,
                                       kng.validate_em96)
        if rc == kng.HUDDLER_TOPBLOCK_OK:
            if data.header.type == kng.KNG_REVR_ATTITUDE:
                read_attitude(data.header, data.datagram, kng_data)
        else:
            abort = True

    plot_attitude(kng_data)
```

version software to compile it, is available on the project website (https://huddl.ccom.unh.edu).

The compiler is accessible via the command line or through a GUI application (using Qt for cross-platform support), named HUSH (HUDDL Schema Handler), which provides additional tools and information to the user. The compiler has been demonstrated with a variety of data formats from sonar manufacturers (e.g., Kongsberg EM Series) and acquisition software companies (e.g., HyPack) both legacy and in active development, both binary and ASCII (Calder and Masetti, 2015).

4. Discussion

The HUDDL framework provides a simple and relatively low-effort solution to harmonize and catalogue the wide (and sometimes wild) range of hydrographic data formats, with their multiple revisions and releases. It also provides the opportunity to generate a catalogue of HUDDL Format Descriptions written in XML, each containing the description for a given data format (with its subsequent upgrades) that can be used as a set of instructions for an application on how to manipulate a data file in a specific format/version. The automation inherent in HUDDLER provides a low cost means of adding new data formats to an application, at least at the basic syntactic level of data access, leaving the coder to focus on the higher-level semantics of what to do with the data after the syntax problem is resolved.

The code in the HUDDLER project is only one means to translate an HFD into source code for use in a data reader: the HUDDL Core Schemas are available directly from the HUDDL community website, and can be used by anyone to develop additional services for HFDs. The code currently generated by HUDDLER is already relatively efficient for data handling, having derived in part from a cruder code generator that has been in use for over a decade. Many optimizations can still be made to improve the performance, and there is significant benefit to doing this in a community supporting a common code generator infrastructure. For example, if code to generate an index for files on first-read were to be added to HUDDLER, or if the frontend reader were multi-threaded, it would then be automatically available to every data format for the cost of a re-compile of the application software.

From the point of view of software manufacturers, HUDDL provides a new tool to build applications that are more data format independent. A single reader component could be developed in isolation and then these modules combined for the various data formats. If a sonar system manufacturer, or software developer, provided an HFD for their data format (which is the ideal case for a strong community), hosted either on their website or that of the project itself (***Figure 9***), it would significantly ease the effort involved in implementing the format in a data processing application. This would allow all readers to have the same understanding of the intended syntax and semantics of the data format. This will reduce some of the efforts required to maintain a set of data readers, usually one for each different format, during subsequent updates to the format, and will help to avoid problems with variant reading of data formats between different applications.

Wherever they are hosted, having the HFD web accessible has significant benefits. For example, when a new version is released, one of the commonly available mechanisms (e.g., RSS) may be used to notify interested users. This push notification allows for alerting of software maintainers as soon as a change is made, so that users do not have to search for changes when there is a sudden problem in reading a data format, or report this as a bug to software vendors. An HFD valid with respect to the HUDDL Core Schemas also allows for automated creation of standardized documentation through the use of XML style sheet technology. The HFD provides a single source for creation and documentation of code, always up to date and consistent.

These publicly available HFDs may be used as ‘trusted’ references for archived data. As long as a binary data file is paired with a HFD, the data content is described and the information can be recovered. The main benefit of this is that it is more likely that users will be able to read the data in the future, and have adequate

documentation, essentially for the price of a metadata link. As long as a version of HUDDLER is available, it can use these HFDs to (re-)create data format drivers to access the archive data with the same simplicity and consistency as for new data format versions. Additionally, once a valid HFD is constructed, the data in that format can be accessed on any platform regardless of the native configuration of the file system.

Having a separate description of the format has the potential for the description (e.g., an HFD) to become separated from its data. Future hydrographic data formats may choose to instead include the HFD as part of the binary file itself to avoid this risk. Another approach to this problem could be to use the first bytes of the file as an integer representing the unique ID reported in a future XML Hydrographic Formats Catalogue, or to store a URL referencing the HFD's location in a well-known place.

At present, the development of an HFD necessarily implies the creation of an XML descriptor for the format. Although there are many XML editors that support this, they are general tools rather than specific to HFDs, and development of the HFD for a complex format can still require significant effort. Of course, that effort is only required once, since the resulting HFD can then be shared by all members of the community so long as it is published at an appropriate URL and indexed, preferably at a clearing house such as the HUDDL community website. The structure of the HUDDL format is much more strict than a general XML file, and could be much more efficiently constructed, and checked, by a tool that reads the HUDDL Core Schemas. This provides the user with a customized interface that allows construction of XML for the HFD only within these bounds. Done graphically, this would significantly ease the burden of constructing the HFD in the first place, and their subsequent update. It is also possible to envision a graphical editing application where the HFD is rendered in diagrammatic form, and the user is able to drag-and-drop new fields and blocks, describing the structure of the data graphically before it is converted into an HFD for distribution.

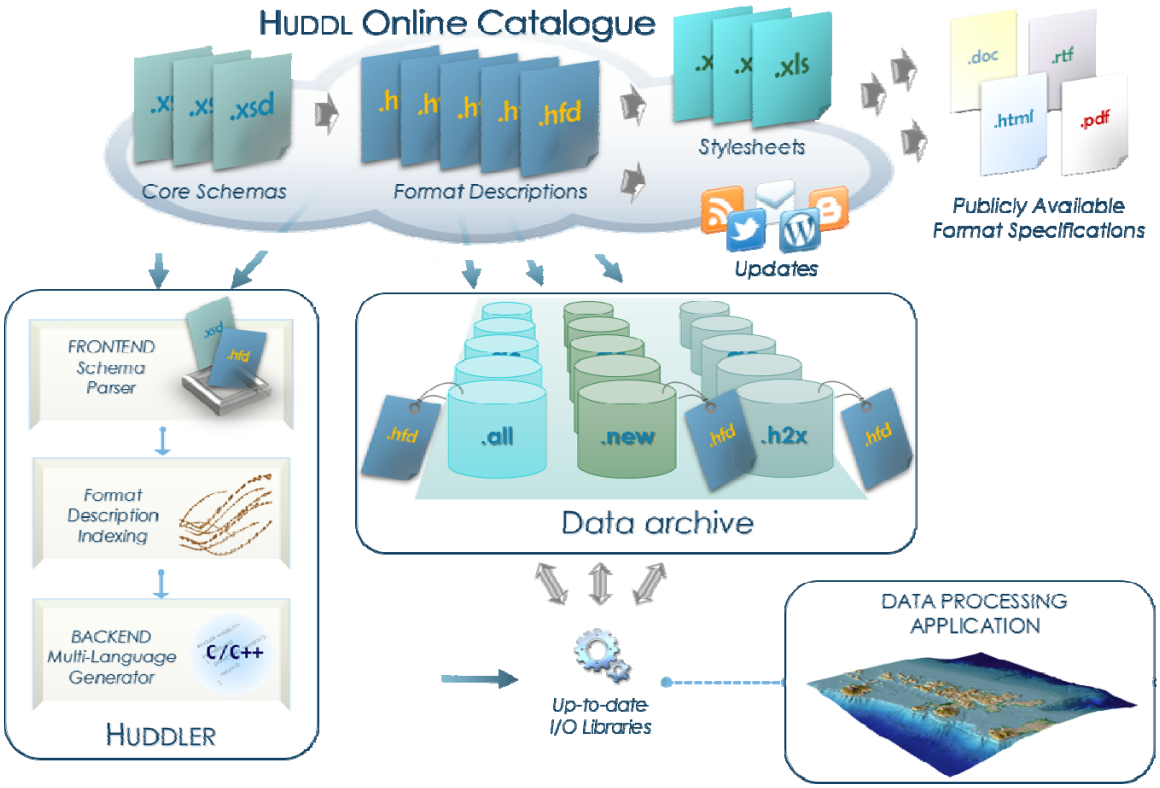


***Figure 9** : HUDDL framework: the online repository is used both for publicly providing format specifications (in different formats) and as a source for HUDDLER, which parses the descriptive schemas, serializes the information and creates an I/O library. Data processing applications can thus rely on this library for access the binary data.*

5. Conclusions

Currently, the hydrographic community has to deal with a multiplicity of data formats. Each format is home grown within its specialty and in many cases is based on manufacturer technology. Although generic data formats have been introduced, there has not been a sufficiently strong reason to rally around one particular method for containing data, nor is any one format general enough to accommodate everyone's needs. The result has been an invasive sea of data formats. At the same time, the temptation to convert all collected data to a selected data format does not appear to be the optimal solution, since many data formats are fundamentally incompatible with each other, and much metadata and information can be lost during this processing (or even worse, mistranslated).

The intent of HUDDL is not to describe every kind of binary data format that people have ever sent or will ever want to send from machine to machine. Rather, HUDDL focuses on the most commonly used hydrographic data formats in order to simplify the problem so that the solution can be efficient and sufficiently easy to use to make it an obvious choice for most users. It can support the hydrographic community on at least at three different levels (***Figure 10***):

- At the descriptive level, where the user simply takes advantage of the common format repository as well as the standardized templates for documentation;
- At the raw data level, where users can use the automatically created raw data parsers. That is, each parser is tailored for a given data format (with all the implicit data peculiarities) as described in the HFDs (as they are compiled by HUDDLER); and
- At the abstract data level, where an additional layer of homogenization is provided with the main aim of simplifying access to hydrographic data (e.g., the same function *getDepthData()* for obtaining the collected depth from various data formats). The Hydrographic Abstraction Layer (HABLA) may also be useful for researchers coming from fields not directly related to ocean mapping. HABLA features are currently in active development.

Many types of applications could benefit from this task-oriented approach: data explorers, conversion tools, metadata archives, etc. HUDDL represents a solution for both software and hardware manufacturers to providing a strong and universal mechanism for version control of hydrographic data formats.

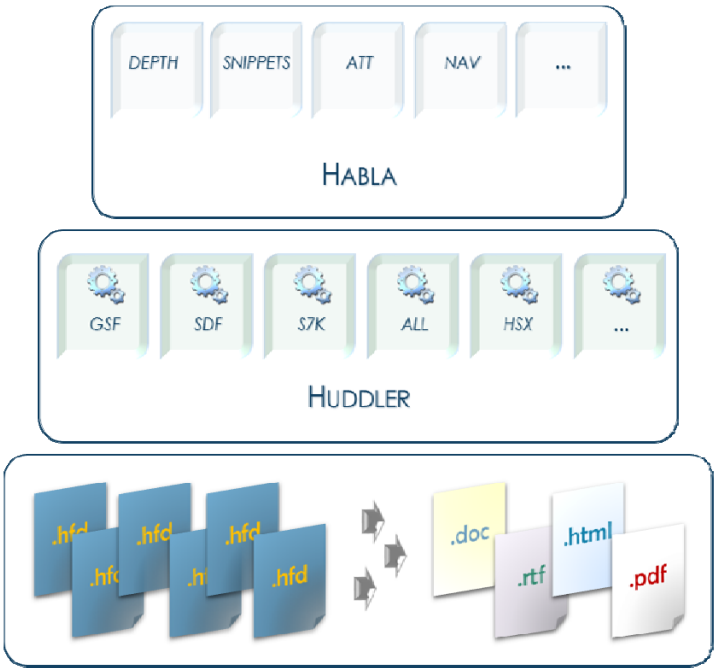


Figure 10 *: The three expected levels of users for the HUDDL framework. The basic Descriptive level provides only description services for different file formats, including documentation construction. The Raw Data level provides basic access to particular data formats through automatically generated data drivers. At the Abstract Data level, extra translations provide for conversion of the data into hydrographically understandable information, such as depth or backscatter, without information as to the underlying data format. The HABLA layer is currently under development.*

Developers will have an abstraction tool for development of binary data readers and converters. In addition, the HUDDL Format Description repository is a powerful solution for propagating the publication of a format update to all interested parties (using popular electronic mechanisms such as tweets, an RSS, or a mailing list).

Based on these considerations, we believe that HUDDL represents a concrete way to reduce, in a relatively short time, existing problems related to interoperability and access to hydrographic data.

6. References

1. Calder, B., 2013, Parallel & Distributed Performance of a Depth Estimation Algorithm, U.S. Hydrographic Conference, (US Hydro 13), New Orleans, LA, USA, pp. 11.

2. Calder, B., and Masetti, G., 2015, HUDDLER: a multi-language compiler for automatically generated format-specific data drivers, US Hydrographic Conference (US Hydro 2015), National Harbor, Maryland, USA.

3. Eisler, M., 2006, XDR: External data representation standard.

4. Georgieva, J., Gancheva, V., and Goranova, M., 2009, Scientific data formats, in Mastorakis, N.E., Demiralp, M., Mladenov, V., and Bojkovic, Z., eds., 9th WSEAS International Conference on Applied Informatics and Communications (AIC '09), Volume 9: Moscow, Russia, WSEAS Press, p. 19-24.

5. Hua, H., and Weiss, B., 2011, Strategies for Infusing ISO 19115 Metadata in Earth Science Data Systems, AGU Fall Meeting Abstracts, Volume 1, pp. 04.

6. ISO, 2008, ISO/TS 19139-2007 Geographic information -- Metadata -- XML schema implementation, International Organization for Standardization, pp. 111.

7. Kaur, G., and Fuad, M.M., 2010, An evaluation of Protocol Buffer, IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the, p. 459-462.

8. Kongsberg Maritime AS, 2013, EM Series Datagram formats - Instruction Manual, p. 126.

9. Masetti, G., and Calder, B., 2014, HUDDL for description and archive of hydrographic binary data, Canadian Hydrographic Conference: St. John's, NL (Canada), pp. 24.

10. Nurseitov, N., Paulson, M., Reynolds, R., and Izurieta, C., 2009, Comparison of JSON and XML Data Interchange Formats: A Case Study: Caine, v. 2009, p. 157-162.

11. Powell, A.W., Beckerle, M.J., and Hanson, S.M., 2011, Data Format Description Language (DFDL) v1. 0 Specification, Report of the Open Grid Forum. Retrieved from www. ogf. org/documents/GFD.

12. Ramachandran, R., Graves, S., Conover, H., and Moe, K., 2004, Earth Science Markup Language (ESML): a solution for scientific data-application interoperability problem: Computers & Geosciences, v. 30, p. 117-124.

13. SAIC, 2012, Generic Sensor Format Specification v.03.04, SAIC, pp. 151.

14. Triton, 2013, eXtended Triton Format (XTF) Rev. 35 Triton Imaging, Inc. , pp. 44.

15. Varda, K., 2008, Protocol Buffers: Google's Data Interchange Format.

16. Westhead, M., and Bull, M., 2003, Representing Scientific Data on the Grid with BinX–Binary XML Description Language: EPCC, University of Edinburgh.

17. Widener, P., Eisenhauer, G., and Schwan, K., 2001, Open metadata formats: efficient XML-based communication for high performance computing, High Performance Distributed Computing, 2001. Proceedings. 10th IEEE International Symposium on, p. 371-380.

18. Yongguo, J., Lianying, L., and Zhongwen, G., 2009, Design of Marine Information Metadata and Directory Service System Based on XML, Database Technology and Applications, 2009 First International Workshop on, p. 574-577.

Author Biography

Giuseppe Masetti received a MS degree in Ocean Engineering (UNH, USA) in 2012, and a Master in Marine Geomatics (2008) and a Ph.D. degree (2013) in System Monitoring and Environmental Risk Management (University of Genoa, Italy). His postdoctoral research at CCOM/JHC is focusing on signal processing and Bayesian hierarchical models for marine target detection. (gmasetti@ccom.unh.edu)

Brian Calder is an Associate Research Professor and Associate Director at CCOM (UNH, USA). He has a Ph.D. in Electrical and Electronic Engineering, completing his thesis on Bayesian methods in SSS processing (1997). He is currently focusing on statistically robust automated data processing approaches and tracing uncertainty in hydrographic data. (brc@ccom.unh.edu)