Copy No.________

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

Developing an eXtensible Markup Language (XML) Application for DFO Marine Data Exchange via the Web

Anthony W. Isenor
Defence R&D Canada – Atlantic

J. Robert Keeley
Marine Environmental Data Service

Joe Linguanti
Institute of Ocean Sciences

Prepared for:

DFO Science
Science Strategic Fund
Attention: Antoine Sioufi
Technology Transfer Advisor
Department of Fisheries and Oceans
200 Kent Street,
Ottawa, Ontario K1A 0E6

Defence R&D Canada

External Client Report
DRDC Atlantic ECR 2003-025
May 2003

Canada

Copy No:

Developing an eXtensible Markup Language (XML) Application for DFO Marine Data Exchange via the Web

Anthony W. Isenor
Defence R&D Canada – Atlantic

J. Robert Keeley
Marine Environmental Data Service

Joe Linguanti
Institute of Ocean Sciences

Prepared for:

DFO Science
Science Strategic Fund
Attention: Antoine Sioufi
Technology Transfer Advisor
Department of Fisheries and Oceans
200 Kent Street
Ottawa, Ontario,
K1A 0E6

Defence R&D Canada – Atlantic

External Client Report

DRDC Atlantic ECR 2003-025

May 2003

Authors

Anthony W. Isenor

Anthony W. Isenor | J. Robert Keeley | Joe Linguanti

Scientists

Approved by

J Kennedy

J.S. Kennedy

Head, Maritime Information and Combat Systems Section

Approved for release by

Kirk Foster

DRP Chair

The information contained herein has been derived and determined through best practice and adherence to the highest levels of ethical, scientific, and engineering investigative principles. The reported results, their interpretation, and any opinions expressed therein, remain those of the authors and do not represent, or otherwise reflect, any official opinion or position of DND or the Government of Canada.

DRDC shall have a royalty-free right to use and exercise any copyright information for its own internal purposes excluding any commercial use of the information.

Abstract

Generic data objects, or bricks, have been developed to describe the grouping characteristics of ocean data and metadata. Full definitions for the bricks were developed and the bricks where then implemented in an eXtensible Markup Language (XML) environment. The XML allows added format and content restrictions on XML elements. A full XML schema was developed for the structure representing ocean profile data.

A data exchange application was then developed by the Marine Environmental Data Service (MEDS), Institute of Ocean Sciences (IOS) and the Bedford Institute of Oceanography (BIO) (in cooperation with Defence R&D Canada) to convert profile data from the three ocean labs to the defined XML structure. Applications were developed in Java, Fortran and extensible stylesheet language transformations (XSLT) to port the XML structure to the local formats. Results show that the use of XML has merit in the ocean data community. However, considerable intellectual effort is required in defining the bricks, structure and definitions. Coding requirements are minimal in comparison and should not be considered an impediment to the development of an XML-based exchange language.

Résumé

Les objets de données génériques, ou séquences prédéfinies, ont été développés pour décrire les caractéristiques de groupage des données et des métadonnées sur les océans. Des définitions complètes de séquences prédéfinies ont été établies et les séquences ont ensuite été mises en application dans un environnement eXtensible Markup Language (XML). Le langage XML accepte le format ajouté et les restrictions de contenu dans ces éléments. Un schéma XML complet a été créé pour la structure représentant les données de profil des océans.

Une application d'échange de données a ensuite été élaborée par le Service des données sur le milieu marin (SDMM), l'Institut des sciences de la mer (ISM) et l'Institut océanographique de Bedford (IOB) (en coopération avec Recherche et développement pour la défense Canada) afin de convertir les données de profil provenant de ces trois laboratoires à la structure XML définie. Des applications ont été développées en Java, Fortran et XSLT (extensible stylesheet language transformations) pour adapter la structure XML aux formats locaux. Les résultats indiquent que l'utilisation de XML est réputée dans le monde des données océanographiques. Mais il faut fournir un effort intellectuel considérable pour établir les séquences prédéfinies, la structure et les définitions. En comparaison, les exigences en matière de codage sont minimales et il ne faut pas les considérer comme un obstacle au développement du langage d'échange basé sur XML.

This page intentionally left blank.

Executive summary

Background

The eXtensible Markup Language (XML) is being widely used as a basis for both dynamic web page development and more generally as a data exchange mechanism. The data exchange aspects of XML include the ability to define flexible data structures that utilise the terminology of the subject area. In this way, the data to be exchanged is packaged in a form more intuitive to the user. Add to this the extensive availability of free software for the manipulation and transformation of the XML data stream, and the developer is now in a position to easily develop, populate, exchange and transform data streams.

In this project, the XML environment is used in combination with the concept of data objects. In the context of ocean data, these objects are described as bricks, where each brick is a package of related data. These bricks can then be assembled to form data structures and applied in data transfer problems using XML.

Principal Results

As an initial project, this effort concentrated on defining those bricks important to ocean profile data. A full set of definitions for the bricks and content was developed. The bricks were combined into structures that were used to contain ocean profile data. The data were then exchanged between the project participants. Writing minimal software, transformations were developed to manipulate the XML data stream into local archive formats.

Significance of Results

The exchange of unambiguous and concise data streams is an important component of data sharing in a networked environment. XML is considered a possible transport mechanism for this exchange. This project exploited the XML environment and development tools freely available for XML. The project shows the potential of XML for the transfer of ocean data.

The national and international ocean data community can utilize this effort by considering the transport of profile data. The effort in developing the exchange mechanism concentrated on developing the bricks, structure and definitions. The community can take advantage of this intellectual effort by recognizing that understanding the process, and coding necessary data transformations, will require minimal effort as compared to the actual development process.

Future Plans

The use of XML in data exchange scenarios will continue to be explored. In particular, work is being conducted on the exchange of codes between data systems using XML. These exchanges have particular importance for Canadian naval operations involving multi-platform sharing of information.

Isenor, Anthony W., J. Robert Keeley and Joe Linguanti. 2003. Developing an eXtensible Markup Language (XML) Application for DFO Marine Data Exchange via the Web, DRDC Atlantic ECR 2003-025, Defence R&D Canada – Atlantic.

Sommaire

Contexte

Le langage XML (eXtensible Markup Language ou langage de balisage extensible) est largement utilisé comme principal outil à la fois pour la création dynamique de pages Web et, de façon plus générale, comme mécanisme d'échange de données. Les aspects de XML rattachés à l'échange de données comprennent la capacité de définir des structures de données flexibles qui utilisent la terminologie du domaine. Les données à échanger sont ainsi regroupées de manière plus intuitive pour l'utilisateur. Ajoutons à cela la disponibilité généralisée de logiciels gratuits destinés à la manipulation et à la transformation des données XML et le fait que le développeur peut maintenant créer, remplir, échanger et transformer facilement les flots de données.

Dans le cadre de ce projet, on utilise l'environnement XML de concert avec la notion des objets de données. Dans le contexte des données sur les océans, ces objets sont décrits comme des séquences prédéfinies, chaque séquence étant un regroupement de données connexes. Ces séquences peuvent être assemblées pour constituer des structures de données et pour être appliquées dans les cas de problèmes de transfert de données à l'aide de XML.

Principaux résultats

Comme il s'agissait du début du projet, on a surtout travaillé à définir les séquences importantes pour les données de profil des océans. On a élaboré un ensemble complet de définitions de contenu pour les séquences. Ces dernières étaient combinées en structures que l'on utilisait pour loger les données de profil des océans. Les données étaient ensuite échangées entre les participants du projet. Pendant la création d'un logiciel minimal, des transformations ont été conçues afin de manipuler les flots de données dans les formats d'archive locaux.

Importance des résultats

L'échange de flots de données clairs et concis est un élément important du partage de données dans un environnement réseau. On considère XML comme un mécanisme de transport possible pour cet échange. Ce projet se faisait dans un environnement XML et disposait pouvait utiliser sans contrainte les outils de développement s'y rattachant. Le projet démontre le potentiel de XML pour le transfert des données sur les océans.

Les spécialistes nationaux et internationaux du domaine des données sur les océans peuvent utiliser le travail déjà accompli en étudiant le transport des données de profil. Le travail de conception du mécanisme d'échange portait principalement sur le développement des séquences prédéfinies, des structures et des définitions. En reconnaissant le fait que comprendre le processus et coder les transformations de données nécessaires demanderont peu d'effort comparativement au processus de

développement réel, les spécialistes du domaine peuvent tirer profit de cette démarche intellectuelle.

Plans

On continue d'étudier l'utilisation de XML dans les scénarios d'échange de données. On travaille notamment sur l'échange de codes entre des systèmes de données utilisant XML. Ces échanges sont d'une importance particulière pour les opérations navales canadiennes mettant en cause de multiples plate-formes partageant l'information.

Isenor, Anthony W., J. Robert Keeley and Joe Linguanti. 2003. Developing an eXtensible Markup Language (XML) Application for DFO Marine Data Exchange via the Web, DRDC Atlantic ECR 2003-025, Defence R&D Canada – Atlantic.

Table of contents

Abstract.. i

Executive summary iii

Sommaire.. v

Table of contents vii

List of figures x

1. REVIEW PERIOD 1

2. PROJECT # 1

3. PROJECT TITLE 1

4. PROJECT LEADERS 1

5. RESEARCH TEAM.. 2

6. OVERVIEW OF THE PROJECT 3

6.1 Introduction 3

6.1.1 Relevance of Data Sharing 3

6.2 Defining a Brick 4

6.2.1 Hierarchical or Relational Bricks 5

6.2.2 Generic Vs. Specific Bricks 5

6.2.3 International Comment on the Bricks.. 6

6.2.4 The Current Bricks 7

6.2.5 Pure and Compound Bricks.. 10

6.3 XML 10

6.3.1 XML Fundamentals.. 10

6.3.2 Elements verses Attributes 12

6.3.3 Local Information.. 13

6.3.4 An Example XML Structure Using the Bricks.. 14

6.4 Construction Guidelines 14

6.5 Implementation Details 16

6.5.1 Granularity and Partitioning 16

6.5.2 Special Treatment of Specific Data 17

6.5.3 Date Time Format 17

6.5.4 Latitude, Longitude and Depth - Pressure Formats 18

6.5.5 Attribute pt_code 18

6.5.6 Element data_set 19

6.5.7 Rules for Compound Bricks 20

6.6 XML Application of the Bricks to Profile Data 20

6.6.1 File sizes 22

6.6.2 The Profile Data Structure 22

6.6.3 The Profile Schema 24

6.7 Mandatory Information in the Profile Brick Structure 25

6.7.1 Variables and Instruments 26

6.7.2 Use of pt_link 27

6.7.3 Use of pt_code 27

6.7.4 Variable Sets and Data Points 28

6.8 Specific Experiences 28

6.8.1 MEDS Ocean Data Format 29

6.8.1.1 Software Environment 29

6.8.1.2 Mapping from MEDS Terminology to XML 31

6.8.1.3 Mapping from MEDS Format to XML 32

6.8.1.4 Software Implementation 33

6.8.2 IOS Header 34

6.8.2.1 Computing Environment 35

6.8.2.2 The IOS Header Format 35

6.8.2.3 Mapping between IOS Format and XML 35

6.8.2.3.1 Stations and Profiles 35

6.8.2.3.2 Variables and Channels 36

6.8.2.3.3 Instrument Information 36

6.8.3 Bedford Institute – Ocean Data Format 36

6.9 Implications of XML Exchange 38

6.9.1 Specific Tag Names 38

6.9.2 Inadequate In-House Formats 38

6.9.3 Parameter Code Mapping 39

6.9.4 Null Values 39

6.9.5 XML Content for Output Format 40

6.9.6 Units 40

6.9.7 Brick Order 40

6.9.8 Quality in location_set 40

6.10 Future Work 41

7. KEY RESULTS ACHEIVED 42

8. IMPACT OF THE PROJECT 43

9. TRANSFER OF KNOWLEDGE/TECHNOLOGY 44

10. EXPENDITURES BY YEAR 45

11. GENERAL COMMENTS 46

12. References 47

Annex 1: Definitions of Current Bricks 48

Annex 2: Controlled Lists 57

Annex 3: Guidelines for Populating Profile Data Structure 64

Annex 4: Schema for Profile Data Structure 68

Annex 5: MEDS to XML Data Mapping 82

Annex 6: Example Profile Data in XML Structure 85

Annex 7: IOS to XML Data Mapping 89

Annex 8: ODF to XML Data Mapping 92

Annex 9: Presentation to IODE 96

List of symbols/abbreviations/acronyms/initialisms 99

Distribution list 101

List of figures

Figure 1. Specific descriptors that may be used to represent a collection of data grouped by cruise and station. The connecting lines indicate the hierarchy...........................5

Figure 2. Multiple profiles may be grouped under a single station, with many stations grouped under a cruise...6

Figure 3. The contents of the provenance brick. 9

Figure 4. Rearrangement of provenance and cruise bricks to form a data collection. 9

Figure 5. An example structure for profile data using the bricks. The text following the – are comments. 15

Figure 6. The profile data structure using bricks. The red text indicates pure bricks, while the green text indicates compound bricks. Compound bricks may also be identified by the name, which typically ends in the string "_set" (the exception being data_collection, which is also a compound brick). The [i,j] notation indicates the [minimum, maximum] occurrence of the brick. The arrows indicate an expansion of one element into subelements in the XML application. The details and definitions of the bricks may be found in Annex 1. 23

List of tables

Table 1. The list of bricks and definitions. Pure and Compound bricks are described in the text. 8

Table 2. The classification of variables into x,y,z,t space provides a perspective when constructing the XML structures from the bricks. This table outlines the four-dimensional space and identifies each dimension as being: C = constant; I = independent variable; D = dependent variable 21

Table 3. Expenditures by year. 45

Table 4. This table provides a general guide to placing content into the XML profile structure. 64

Table 5. MEDS to XML data mapping..82

Table 6. IOS to XML data mapping. 89

Table 7. BIO ODF to XML data mapping...92

Table 8. Slide presentation made to IODE by Bob Gelfeld, Associate Director, World Data Centre – A Oceanography. 96

This page intentionally left blank.

1. REVIEW PERIOD

May 2002 to March 31, 2003

This report is intended to meet the final report requirements for the Fisheries and Oceans (DFO) Science Strategic Funds (SSF).

2. PROJECT

90177

3. PROJECT TITLE

Developing an eXtensible Markup Language (XML) application for DFO marine data exchange via the web

4. PROJECT LEADERS

Anthony W. Isenor
Current Affiliation:
Defence R&D Canada – Atlantic
9 Grove Street, Dartmouth
Nova Scotia, Canada
B2Y 3Z7
anthony.isenor@drdc-rddc.gc.ca

J. Robert Keeley
Marine Environmental Data Service
Department of Fisheries and Oceans
W12082 - 200 Kent Street
Ottawa, Ontario, Canada
K1A 0E6
keeley@meds-sdmm.dfo-mpo.gc.ca

5. RESEARCH TEAM

Joe Linguanti
Institute of Ocean Sciences
PO Box 6000, 9860 West Saanich Road
Sidney, B.C., Canada
V8L 4B2
linguantij@dfo-mpo.gc.ca

Jeffery Jackson
Bedford Institute of Oceanography
1 Challenger Drive
PO Box 1006, Dartmouth
Nova Scotia, Canada
B2Y 4A2
jacksonj@mar.dfo-mpo.gc.ca

Naveenta Anand
Marine Environmental Data Service
Department of Fisheries and Oceans
W12082 - 200 Kent Street
Ottawa, Ontario, Canada
K1A 0E6
anand@meds-sdmm.dfo-mpo.gc.ca

Doug Gregory
Bedford Institute of Oceanography
1 Challenger Drive
PO Box 1006, Dartmouth
Nova Scotia, Canada
B2Y 4A2
gregoryd@mar.dfo-mpo.gc.ca

The research team was initially comprised of Fisheries and Ocean personnel. However, over the course of the project, personnel changes resulted in the project becoming inter-departmental, with contributions from Fisheries and Oceans (DFO) and Department of National Defence (DND). This collaboration successfully exploited the expertise of both Departments, utilizing the experience of Fisheries and Oceans in the full data management cycle and the existing XML knowledge within DND. This cooperative effort also had Departmental benefits, providing both Departments with enhanced knowledge on issues related to the network sharing of data and information in a distributed environment. We consider this inter-departmental effort an excellent example of cooperating departments exploring mutual interests to the benefit of both.

6. OVERVIEW OF THE PROJECT

6.1 Introduction

Extensible Markup Language (XML) provides a flexible way to deliver data and information between parties. XML is being used in many domains, including scientific and commercial. In oceanography, there has been a desire expressed to develop a form of XML suited to exchanging marine data and information. Development of a Marine XML requires some degree of standardization to be successful.

There is considerable interest in XML within the oceanographic community. The International Council for the Exploration of the Sea (ICES) and the Intergovernmental Oceanographic Commission (IOC), through the Intergovernmental Oceanographic Data and Information Exchange Committee (IODE), have a partnership Study Group on the Development of Marine Data Exchange Systems Using XML (SGXML). This group is focusing on the application of XML to issues of multiple parameter codings and metadata. The SGXML is also looking to this SSF project to provide information on application of XML to ocean data. One member of this project team (Isenor) is co-chairing the SGXML.

Other groups are also interested in this project. The Russian National Oceanographic Data Centre is actively promoting the use of XML in a distributed ocean data model. These efforts are coordinated through the Joint World Meteorological Organisation (WMO)-IOC Technical Commission on Oceanography and Marine Meteorology (JCOMM) Data Management subgroup, being lead by N. Mikhailov, subgroup chair. Mikhailov is also a member of the SGXML.

The European Union (EU) is also funding an investigation into developing a Marine XML. Although this SSF project has no formal links to the EU project, the IOC members on the SGXML do provide an informal flow of information between the parties.

6.1.1 Relevance of Data Sharing

Databases and information management systems represent a significant resource for both DFO and DND. Examining new and innovative ways to streamline the transfer of these data is critical to successful scientific and defence research. As well, data transfer is important for government initiatives making data and products available on-line to clients (e.g., Government On-line) or the sharing of information between cooperating parties.

Initiatives involving the monitoring of the marine environment or operational oceanography also require the efficient and unambiguous transfer of data and information. Exploring new data infrastructure and integration methods will help ensure the most efficient flow of data between parties.

The delivery of timely and reliable scientific advice requires proper data flow. Efforts to streamline data flow will help ensure the most current information is available when decisions and advice are required from the decision-makers.

The project being reported here is focused on taking an idea for a general definition of a Marine XML and testing its applicability on the specific case of profile data held in three regions of DFO. The strategy employs a relatively small number of generally defined data objects that can be assembled in different ways to reflect the structures of the data. These objects, called "Keeley bricks" [1], represent a basic building block for packaging data and metadata. Within this document, these objects will simply be referred to as “bricks”.

6.2 Defining a Brick

Measurements, regardless of the domain in which they are made, have many characteristics in common. We typically measure something as a function of something else. That is, we use independent variables such as water pressure, to measure dependent variables, such as water temperature. Depending on how many independent variables we use, the resulting measurements can be expressed as points, lines, planes or volumes.

When we make measurements, we always do so using some units of measure. Sometimes we may convert from one set of units to another through some simple or complicated formula.

Our instrumentation also has characteristics. Our measurement devices have limitations that can be expressed as limits to accuracy and precision. It is sometimes useful to quote detection limits. We often wish to record the make and model of the instruments for future reference.

In some disciplines, the measurement strategies and techniques are crucial in the interpretation of the actual measurements. Sample storage techniques, including environmental details and physical media, can be important.

It is often useful to record details about the origins of the data. For example, it is common practice at data centres to record information regarding the source of the data. This permits inquiries to go back to the originator.

All of the above examples represent information, either concerning the measurements themselves or about metadata surrounding the measurements. Much of this can be encapsulated into packages or objects of information that may apply to a data collection as a whole, or simply to discrete portions of a data collection. It is these packages that we are attempting to define and which we call bricks.

However, the brick analogy extends further. Just like building bricks can be used to construct different shapes of structures, we believe that a clever definition of bricks will allow us to build many different forms to describe the variety of data collected over a broad spectrum of disciplines.

6.2.1 Hierarchical or Relational Bricks

One of the useful characteristics of the bricks is that they may be assembled in many different ways to construct different structures. For example, we may consider assembling the bricks in a hierarchy. Suppose we have a collection of bricks representing cruises and stations. We could envisage a structure as shown in Figure 1. Here, the cruise brick is considered higher in the hierarchy than the station brick.

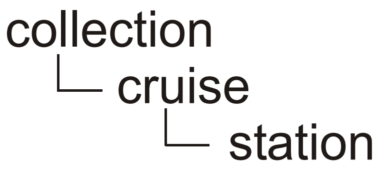


Figure 1. Specific descriptors that may be used to represent a collection of data grouped by cruise and station. The connecting lines indicate the hierarchy.

In other cases, where information relationships are many to many (in a relational database sense), the encoding may take one of many views. The person encoding the information will need to choose the "viewpoint" and repeat information.

For example, a relational database may consist of information relating people to projects. One person can be a member of many projects, and one project can involve many people. To convey this requires a choice of viewpoints. One viewpoint would list all people and for each person, list the projects they are connected to. The alternate viewpoint is to list all projects and for each, list the people involved. In either case, data is being repeated either in the form of names or projects.

6.2.2 Generic Vs. Specific Bricks

Now consider the case of water column data from an oceanographic cruise. Such data collections are typically structured as profiles collected at stations within the cruise. An obvious way to represent this data with bricks would be to use the very specific descriptors shown in Figure 2.

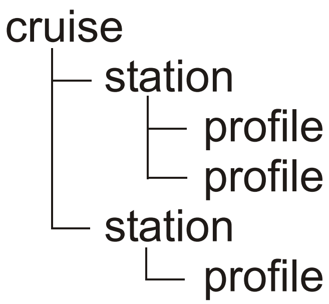


Figure 2. Multiple profiles may be grouped under a single station, with many stations grouped under a cruise.

This is a perfectly good strategy that exploits the vocabulary used in the oceanographic domain. Suppose now we have another data collection, but this time it is a time series of current speed and direction and a set of depths on a mooring. In this case, we could use descriptors such as "mooring", "instrument" and "series". Again, the descriptors use the common vocabulary.

If we consider more closely the two data collections, we see that they really have the same structures. The data are simply one (or more) dependent variables measured as a function of an independent variable (depth for profiles and time for the time series). If we can find appropriate descriptors for this structure then we can use the same descriptors to describe both.

The advantage of using the more general descriptors is that we do not become overwhelmed by the myriad of descriptors that may be used. If the number of descriptors becomes large, we risk confusing matters. In all cases, it will be essential to have clear descriptor definitions.

6.2.3 International Comment on the Bricks

An initial version of the bricks was presented to the ICES-IOC Study Group on XML in April 2002. Although no official comments made it to the final report [1], several comments were made unofficially.

Following the presentation of the Keeley brick application to an XML environment, there were mixed opinions within the SGXML membership. Unofficial comments and discussion after the presentation centred on the following points:

- There is a need to provide a unit override mechanism within any defined structure. This will provide people with a mechanism to use a particular parameter dictionary, but not use the units associated with the parameter as defined in the dictionary.

- There must be an allowance for undeclared objects

- The bricks should use more mandatory elements (e.g., latitude, longitude, date, time)

- The format of key metadata (e.g., latitude, longitude, date, and time) should be defined

- The bricks should use metadata standards where possible

These comments were considered and steps taken to address every comment in the current version of the bricks presented here. An investigation of metadata standards [2] did result from the last comment, however, it was completed too late in the development process to be incorporated into the results of this project. This report provides the details that address the remaining comments.

6.2.4 The Current Bricks

The working list of bricks developed from this project is shown in Table 1. The complete list of bricks and content is provided in Annex 1.

Annex 1 provides the definitions of the bricks. As well, the elements and attributes within the bricks are also defined. The occurrence of these elements and attributes are included. Any content constraints on the attributes are noted by referring to a list constraint. All list constraints are provided in Annex 2.

As an example, we consider the provenance brick in more detail (see Figure 3). In a data management capacity, it is always useful to know where data originated. So, knowing the person, the agency, perhaps what the original identifier was at the agency, and the date of dataset creation, are all items that apply to the origins of the data. We have built a provenance brick to hold this information.

Table 1. The list of bricks and definitions. Pure and Compound bricks are described in the text.

BRICKS	BRICK TYPE	DEFINITIONS
analysis_method	Pure	Stores information about any physical, chemical or biological analyses carried out on the data
availability	Pure	Stores information about the possible release of the data to the public
calibration	Pure	Stores calibration information on the instrument, sensor or variable
comment	Pure	Stores general textual information not intended to be used in data retrievals
data_collection	Compound	Used to encapsulate the entire XML file
data_dictionary	Pure	Indicates the specifics of the data dictionary being used within the collection
data_point	Pure	Used to store any type of data or metadata value
data_set	Compound	Used with data_set_id to define the granularity of the collection
data_set_id	Pure	A numeric or text identifier for a particular data set
depth_pressure	Pure	Store the z coordinate of the data
history	Pure	Processing history of the data
history_set	Compound	Used to encapsulate history information
instrument	Pure	Information about the instrument used to make the measurements
latitude	Pure	The y coordinate of the data
ldate	Pure	The time coordinate of the data
location_set	Compound	A compound brick to record the x, y, z, t values
longitude	Pure	The x coordinate of the data
previous_value	Pure	Information about the value before it is changed
provenance	Pure	The originator of the data
quality	Pure	A marker providing an assessment of data quality
quality_testing	Pure	Information about how the data quality assessment was made
sampling	Pure	Information about the sampling methods used
sensor	Pure	Identifies sensor specifics
units	Pure	The units of measurements
variable	Pure	Information about the variables measured
variable_set	Compound	A compound brick used to declare a variable

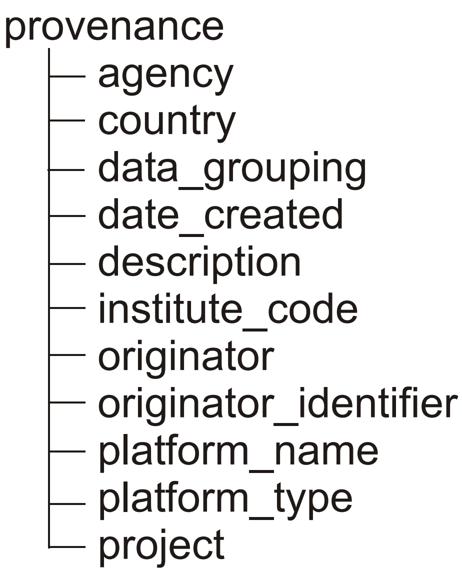


Figure 3. The contents of the provenance brick.

For some collections of data, there may be a single provenance brick for the entire collection. However, we can imagine that a collection may be assembled from the data collected by a number of principle investigators and if we wish to record all of the names, we may have multiple occurrences of a provenance brick in a single data collection. This should cause no difficulty, as the basic structure should have the form shown in Figure 4 (where cruise represents all of the information about individual cruises from the same provenance. Note that cruise is not a brick).

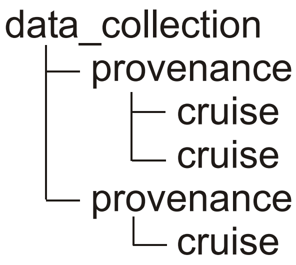


Figure 4. Rearrangement of provenance and cruise bricks to form a data collection.

The list of bricks shown in Table 1 does not include those that we expect will be needed for other data types. For example, we have no brick in which to store information about species taxonomy. This is important information for biological data.

6.2.5 Pure and Compound Bricks

The list of bricks in Table 1 indicates the brick type as being either pure or compound. *Pure bricks* may be considered those bricks that contain subcomponents (only one sub-level in the hierarchy) with data or information content.

Compound bricks can be thought of as simple containers in which bricks are assembled. A compound brick contains no content other than bricks (pure or compound). Its sole purpose is to provide internal structure to the XML document.

6.3 XML

To illustrate how we envisage using bricks to build XML data files we will first review some fundamentals of XML. Then, we consider some of the basic questions and decisions surrounding the application of the bricks to XML.

6.3.1 XML Fundamentals

This project report will not attempt to explain all aspects of XML. Rather, a brief introduction is provided to give the reader enough information to understand the remaining report sections dealing with XML.

XML was developed by the World Wide Web Consortium (W3C) with the release of the specification in 1998. XML is actually a language used to develop other languages. The developed language is thus based on XML, but is not properly named XML.

In the simplest of terms, XML may be used to construct a language using any known computer based character set. XML provides various structures that may be used to capture the data. The simplest XML structures are elements and attributes.

An XML *element* is similar to a data object. It may contain other elements or attributes. XML syntax used to identify an element is the angle bracket, < and >. For example, the element named cruise would be written as

<cruise>

The actual text and angle brackets represent the *tag*. To close an XML element, the / is included in the trailing tag. For example:

<cruise>
</cruise>

Alternately, an empty tag may be shortened to be:

```
<cruise/>
```

To encapsulate another element inside the cruise element, the syntax would be:

```
<cruise>
  <station>5</station>
</cruise>
```

Here, the leading spaces on the station element are included for clarity. The numeric 5 is the content of the station element.

An *attribute* for an element is included within the starting tag. For example, if the station element had an attribute "name" and the name of the station was "Bravo", then the syntax would be

```
<cruise>
  <station name="Bravo">5</station>
</cruise>
```

In the text, attributes will be indicated using the braces { } such as {name="Bravo"}.

A *namespace* may also apply to the developed language. Although namespaces will not be dealt with in detail, they do appear in the schema elements within this report. A namespace may be considered a specifically named topic area for the developed language. For example, if the developed language for this project were “Ocean Data XML” and the namespace “odax” was declared to represent this language, then the namespace addition would be

```
<odax:cruise>
  < odax:station name="Bravo">5</ odax:station>
</ odax:cruise>
```

Finally, in XML terminology, there are well-formed documents and valid documents. *Well-formed* means the start and end tags occur in the proper sequence, similar to a stack first-in-last-out feature. *Valid* means the document agrees with a structure defined in a schema.

There are many more syntactic rules for constructing XML based languages. These rules will not be reviewed here. Those interested are referred to the many on-line resources or published books on the subject (see [3], [4]).

6.3.2 Elements verses Attributes

As described above, there are two basic XML constructs for capturing data content: elements and attributes. An *element* may be defined as any construct that can encapsulate other attributes or elements. An element may be considered a data object. An *attribute* is loosely defined as a characteristic of an element. Attributes do not encapsulate data, but may contain data.

The ICES-IOC Study Group on XML summarized the characteristics of attributes [1]. To summarize, the SGXML noted the following attribute related points:

- You cannot have an attribute name without defining a value
- No two attributes can have the same name in a single tag
- There is no order dependence to attributes
- You cannot expand attributes to contain other attributes within the same document
- You can limit values for attributes to predefined lists

To illustrate the use of attributes and elements within an ocean context, consider the following example using a date. We may write a date using only XML elements

```
<date>
   <type>start</type
   <value>19981022</value>
</date>
```

Alternately, the type of date may be indicated using an attribute of the date element.

```
<date type=”start”>19981022</date>
```

The second formulation is more compact and in this particular case may be a viable solution.

In a number of XML tag formulations, it would seem that attributes have been overused to the extent that there is no longer a clear division of when to use an attribute. In an attempt to define clear conditions for the use of attributes, during this project attributes are used when the content can be described in a finite list. The example above illustrates such a finite list for a date type. We can imagine dates to represent the start of an event, the end, and an instantaneous value. Since this list is finite, we would choose the second formulation over the first.

6.3.3 Local Information

Considerable effort has gone into the formulation of the brick structure and implementation in the XML environment. However, we realize that the bricks and structure we have developed cannot meet all possible issues involving ocean data transfer.

The flexibility to modify the XML structure to meet local needs is very important. Many advocate developments that account for such local specialization of the structures to meet these local needs [5].

In the development of the bricks, the local aspect of any implementation was initially accounted for via the tag “local_tag”. However, as development proceeded the implementation of this local_tag became difficult and investigations began with regard to the XML schema definition (XSD) specification for the “any” tag.

The XSD any tag may be added to any developed schema. By specifying the use of all namespaces other than the one currently being used, the any tag may be added to the first position of any XSD sequence without violating the unique particle attribution rule [6]. For example, the following XSD code adds the any tag to a schema tag sequence.

```
<xsd:any namespace="##other" processContents="skip" minOccurs="0"
maxOccurs="1"/>
```

By including the above XSD snippet in a schema at the appropriate location within a defined sequence, the local user would be allowed to include any tag from another namespace in their local instance document. The local user would be required to add a namespace specification in their instance document, such as:

```
xmlns:rev="http://www.meds-sdmm.dfo-mpo.gc.ca/meds"
```

This specification is added as an attribute in the root element in the local instance document. Then, the local instance document would be able to contain local tags such as:

```
<rev:local_addition>
    <rev:species>calanus finmarchicus</rev:species>
</rev:local_addition >
```

Such an addition would pass all validation checks against the schema.

Although the flexibility of local additions is useful, we must act cautiously in using such an open-ended facility. If used extensively, this flexibility could defeat the basic idea behind the bricks. Additions made in this way should be monitored for possible inclusion within the brick framework.

6.3.4 An Example XML Structure Using the Bricks

Having defined the list of bricks (Table 1) and illustrated the sub-elements for some of these bricks (Figure 3), we now present an example of applying these bricks to an ocean data set. The strategy here uses a common oceanographic data collection unit, the cruise. A cruise is considered to consist of a number of stations at which one or more variables are measured over the water column.

An XML document assembling this cruise, station and data value information might take the form shown in Figure 5.

Figure 5 shows the XML document structure down to the station level. The actual implementation of this structure would cascade down to the individual pressure level.

Note that each level is contained inside a data_set element and whenever the compound brick data_set is used, the same set of tags may be used. This example illustrates how the encapsulation of the data values inside data objects can be managed in the XML environment.

6.4 Construction Guidelines

Using a basic set of bricks, it is possible to build an XML document in many different ways for a single collection of data. This project, which concentrates on oceanographic profile data, first needed to define some construction rules for the bricks and structure. This section presents some guidelines for the construction process.

1. Data Hierarchy should be exploited

Oceanographic datasets often contain a natural organisation that results in a hierarchical structure. The example of a cruise containing many stations is one example of this hierarchical structure. The bricks, both pure bricks and compound bricks, must be capable of exploiting this natural internal organisation of datasets.

2. Attributes are restricted.

Some guiding principle needed to be defined for the use of attributes. For this project, attributes will be used when the required content can be selected from a finite list of possibilities. Such lists are similar to domain tables in data modelling and as such, attributes are then similar to categorical variables.

```
<data_collection>     -- the compound brick acting as container for the data
   <comment/>         -- comments relevant to the entire collection
   <data_dictionary/> -- describes the dictionary used in the XML document
   <provenance/>      -- origins of the collection
   <location_set/>    -- a compound brick to hold x,y,z,t information of the data
   <data_set>         -- a compound brick to hold the data
     <availability/>  -- distribution privileges for the cruise
     <comment/>       -- comments relevant to the cruise
     <data_point/>    -- data that pertain to the cruise
     <data_set_id/>   -- an identifier for the cruise
     <provenance/>            -- origin of the data if different from the data_collection
     <quality/>               -- quality indicators pertaining to the cruise
     <quality_testing/>       -- description of QC undertaken
     <variable_set/>          -- variables reported
     <location_set/>          -- a compound brick to hold x,y,z,t information
     <history_set/>           -- a description of processing history for the cruise
     <data_set>               -- a compound brick to hold the data
       <availability/>        -- distribution privileges for the station
       <comment/>             -- comments relevant to the station
       <data_point/>          -- data that pertain to the station
       <data_set_id/>         -- an identifier for the station
       <provenance/>          -- origin of the data if different from the cruise
       <quality/>             -- quality indicators pertaining to the station
       <quality_testing/>     -- description of QC undertaken if different from cruise
       <history_set/>         -- a description of processing history
     </data_set>
  </data_set>
  <data_set>          -- a compound brick to hold the data
    <data_set_id/>    -- an identifier for another cruise
  </data_set>
</data_collection>
```

Figure 5. An example structure for profile data using the bricks. The text following the – are comments.

3. Generalization of brick contents

Each brick has a number of tags or attributes that may not appear to be important in one context, but will be useful in another. The internal structure of the bricks has been built to offer all information relevant to the brick, to an assortment of users. This generalization is critical to the wide use of any XML structure. In fact, one key benefit to XML is that ignoring superfluous information is trivial.

4. Each XML document should have sufficient information to be self-describing.

All descriptive information should always be included in the XML document so that it is completely self-contained. At any time, a user should find all required information in the file to be able to know exactly what is the content.

5. There are a limited number of XML schemas.

It is expected that another kind of data may well have a different structure. In that case, the bricks will be assembled in the most appropriate way to record the information. However, it is important to have a limited number of data structures. This will permit XML parsing software to exploit the limited number of schemas for checking the XML document for validity.

6. Mandatory elements or attributes

The elements and attributes within any brick were defined to meet a multitude of needs, from data centre to data centre exchanges to the local user requesting data. In this framework, the elements and attributes for a brick become a complicated assortment only related through the common topic of the brick. However, it is important to identify the core component or components of a brick that would or should apply to all users of that brick. Within each brick, these components are declared mandatory.

6.5 Implementation Details

This section describes the many details of this particular implementation.

6.5.1 Granularity and Partitioning

The definition of the brick elements and attributes represents a considerable effort. As with all data modelling exercises, a firm understanding of the problem space helps define the resulting granularity of the data objects. Here, granularity describes the level of content decomposition.

A date may be used in an example illustrating granularity. A date is typically composed of a year, month and day. One situation may result in this information being stored in forms such as yyyy-mm-dd or dd-mmm-yyyy, where yyyy represents the four digit year, mm the two digit month and dd the two digit day. However, if the date were stored in this form, parsing would be required to extract the individual components (e.g., month) from the data content.

The processing cost of the parsing may be acceptable in those cases when parsing the date content represents an exceptional case. However, if in another situation parsing is often required, then the date may be better stored as individual content in a set. In the set, each component of the date is accessible without parsing.

In the second situation, the date is stored at a higher granularity. Although parsing time is reduced, storage requirements increase.

For data modelling within an XML environment, granularity has direct implications to file size. Tag names occupy considerable space within an XML document and increasing the granularity also increases the number of tags. Another important point is that increased granularity within an XML environment provides more stringent control over the content. This is because the increased granularity may be used to increase specific content checking in the XML schema. This provides more control over the content through increased validation.

The partitioning of the data into objects is an important function of the XML environment. For this project, the data partitioning was conducted relative to the natural structure of the data. For profile data, this results in levels within the XML document that describe the cruise, the station, the profile and the record. All of these levels are described by the data_set and data_set_id elements.

6.5.2 Special Treatment of Specific Data

In the initial development of the generic bricks, specifically named bricks to store space and time coordinates of the measurements were not incorporated. Comments on the initial bricks emphasised the importance of these variables to any geospatial dataset. Thus, it was decided to emphasis this importance by constructing specific elements to hold the spatial-temporal information. These bricks are latitude, longitude, depth_pressure and ldate to emphasize their importance. An assessment of this decision will follow in the section dealing with the implications of XML exchange.

6.5.3 Date Time Format

There are many elements within the bricks that contain a date or time value. The formats of these values were set to comply with the date and time data type definitions from the W3C specifications for XML. This choice allows automatic checking of the date values using validation built into the XML data types.

For all element date values, the format was set to yyyy-mm-ddZ. Both the dashes, "-", between digits and the "Z" must be present. The Z indicates that dates are reported in GMT.

For all element time values, the format was set to hh:mm:ss.ssZ where hh represents the two-digit hour, mm the two-digit minute and ss.ss represent the seconds to two decimal places (the decimal part of the seconds may extend to any number of decimal places). In the time format, the colons, ":", and "Z" must be present. Again times are reported in GMT.

6.5.4 Latitude, Longitude and Depth - Pressure Formats

The latitude and longitude bricks record the position on the earth at which the observations were made. For this project, the convention for latitude allows a range from -90 to +90 degrees with positive values being north of the equator. The longitude value must be within the range of -180 to +180 degrees with positive values measured east of Greenwich.

The depth_pressure brick uses pt_code to identify a code that indicates if a depth or pressure is being recorded. In either case, the vertical coordinate is measured with the sea surface being zero and values positive downwards.

6.5.5 Attribute pt_code

For the specification of a parameter code within the XML structure, there exist three possibilities:

1) the code as a tag name,
2) the code as tag content, and
3) the code as attribute content.

The parameter code existing as a tag name was not considered viable. In essence, this option would allow for many thousands of tags, as there are many thousands of oceanographic parameters. This option would also eliminate the usefulness of the schema validation process. This is because one would not be able to create a schema that identified all possible combinations of parameters.

Considering option 2), the parameter code could easily exist as content for the tag. However, if the data value is also stored in tag content, then the direct connection between the code and the data value is lost. This connection could exist if the attribute of the tag contained the data value.

Option 3) was selected for this project. This option packages the code with the data value in a single XML element. As well, the option leaves open the possibility of using a list of allowable parameters for the attribute content.

The pt_code attribute is used to store parameter code information in various bricks. As well, a pt_link attribute is used to provide a counter on the pt_code. The pt_link attribute allows for the same parameter occurring more than once in a dataset.

6.5.6 Element data_set

The definition of a dataset is sometimes contentious. A dataset may imply many different things to different people. Nevertheless, we attempt to define a generic dataset that can meet the diverse ocean data community and the application of profile data within this project.

In this regard, a *dataset* (note: don't confuse *dataset* with data_set. The first is a collection of data in various forms. The latter is a compound brick) should contain some very basic information. A dataset may contain:

- a basic identifier either by name or number
- a history of processing, including processing related to quality testing and results of this testing
- a definition of the level of availability for the dataset
- parameters or variables
- variables that can be represented as data points within the dataset
- identification of the dataset owner, and
- other datasets.

Using this definition, we constructed the compound brick named data_set. This compound brick contains other compound and pure bricks that address the above requirements. There are no mandatory bricks within the data_set compound brick.

Dataset levels were also defined for this implementation of profile data. This exercise defined an appropriate use for each compound or pure brick at all levels within the profile structure (see Annex 3). This was important for identifying the applicability of reoccurring compound and pure bricks within the structure.

6.5.7 Rules for Compound Bricks

Before declaring and defining numerous compound bricks to meet the immediate needs of the project, a set of rules was established for defining and using compound bricks. The rules are as follows:

- Compound bricks will have no attributes

- Compound bricks will contain no natural subelements of their own (i.e., they are not defined with any internal structure or content)

- Designers may declare subelements to be contained within a compound brick. These subelements shall be:

    - Bricks first, in alphabetical order

    - Compound bricks second, in a once defined but arbitrary order

    - data_set compound brick last (if required)

These rules have resulted in some interesting effects. For example, consider the information within the data_set_id brick. This brick contains identifying information for the data encapsulated within the data_set compound brick. However, the identifying information is at a lateral level to the information itself as opposed to the object-oriented approach that would result in the information encapsulated below the identifying data. Note that this could easily be changed so that the identifying data would be part of the encapsulating wrapper around the information, by moving the attribute within data_set_id to the data_set compound brick. However, having established the rules it was decided to evaluate the consequences rather than continually modifying the rules during the investigation.

A similar effect exists with the location_set compound brick. This compound brick contains many pure bricks with the same attributes. This situation could be simplified by moving these attributes to the location_set compound brick.

6.6 XML Application of the Bricks to Profile Data

The application of the bricks to a specific type of data is a process of turning the more abstract ideas into real and usable solutions. This process requires a number of steps. The details concerning these are developed in the sections below.

The contents of the generic bricks shown in Annex 1 were strongly influenced by our more immediate goal of focusing on profile data. Throughout the process we have tried to keep in mind the broader goal of designing bricks that are capable of representing other kinds of ocean data.

In that regard, one should recognize that profile data is really a subset of ocean data, where the data are described by one independent variable and many dependent variables. Table 2 expands this examination of independent and dependent variables to illustrate examples for (x, y, z, t) space.

In considering Table 1, we note that the number of constant values defines the "class" of data. For example, the vector class has three constant values while the "plane" class has two constant values.

This application of bricks to profile data represents a vector class investigation.

Table 2. The classification of variables into x,y,z,t space provides a perspective when constructing the XML structures from the bricks. This table outlines the four-dimensional space and identifies each dimension as being:
C = constant;
I = independent variable;
D = dependent variable

CLASS	X	Y	Z	T	EXAMPLES
4 D	I	I	I	I	AUV data
3 D	I	I	C	I	thermo-salinograph (or underway TS), drifters
	I	I	I	C	Tomography, Net Tow
Plane	I	I	C	C	Satellite image
	I	I	D	C	Altimeter
Vector	C	C	C	I	Time series (e.g., current meter)
	C	C	D	I	Water level time series
	C	C	I	C	Profiles (e.g., CTD)

6.6.1 File sizes

An immediate consideration will be the size of the XML documents that are created. There will be a natural tendency to manipulate any developed structure to reduce resultant file sizes. Considering the general bricks, it is evident that even for a relatively small amount of data to be stored in a data collection, the resulting XML document will be considerably larger than the original file from which the data were created. The tendency to adjust the structure to be compact has been resisted. We have, instead, chosen to exploit widely available tools for compression of files.

The XML documents generated within this project provided an opportunity to investigate file size issues. The BIO Ocean Data Format (ODF) files, which are ASCII, were compared to the XML documents generated from the ODF content. When comparing six ODF files (bottle, Conductivity-Temperature-Depth (CTD), expendable bathythermograph (XBT), float, moored current meter, and underway TS data) to the XML equivalents, the XML representation occupied 600% of the disk space as compared to the ODF files. However, when compressed using zip compression software, the XML representation was 140% as compared to the compressed ODF files (using default zip settings). Other compression software (bzip) actually improved compression, producing bzipped files totalling 75% of the size of the zipped ODF files. Note that the 75% indicates that the bzipped XML files are actually smaller than the original zipped ODF files. This indicates that compression of XML files may alleviate XML file size issues during transport of the data.

6.6.2 The Profile Data Structure

Figure 6 shows a pictorial representation of the profile structure developed for this application. In it, each box surrounds a compound brick and the lines and arrows represent the cascading internal structure. In Figure 6 we see the data_collection brick, containing comment, data_dictionary, provenance, location_set and data_set bricks. Compound bricks are indicated using green text while pure bricks are indicated using red text. Compound bricks also end in the string “_set”, the exception being data_collection.

Connected by arrows to the data_collection are two data_set bricks. Each of these has an attribute designator {"level=cruise"} in the data_set_id and so each of these contains the data and information pertaining to an individual cruise. The Figure shows two cruise datasets in the data_collection, but any number may be added.

Connected to the data_set {"level=cruise"} is a data_set {"level=station"}. There will be as many of these data_set compound bricks as there are stations in the cruise. This has further connected data_set compound bricks for individual profiles and then individual records (depths or pressures) in each profile. These data_set compound bricks repeat as required to deliver all of the record data for each of the many profiles that may constitute a station.

Also within the data_set compound brick are four other compound bricks called location_set, variable_set, history_set and a data_set {"level=related"}. These compound bricks provide information about the space and time location of the data set in which they are nested, the variables, and any processing history that is available. The data_set {"level=related"} is intended for those measurements made at a station that accompany ocean profile observations (e.g., wind observations, sounding). A location_set brick is included for those cases when these additional observations are collected at a different space or time coordinate than the main station data (e.g., winds measured at some height above sea level).

Beside the name of each brick has been placed an indicator of the occurrence of the brick. The syntax used is [j,k] where j indicates the fewest number of occurrences that may be found and k represents the maximum. A value for k of n means as many as required. The mandatory bricks are indicated by [1], meaning they must be present but can only occur once.

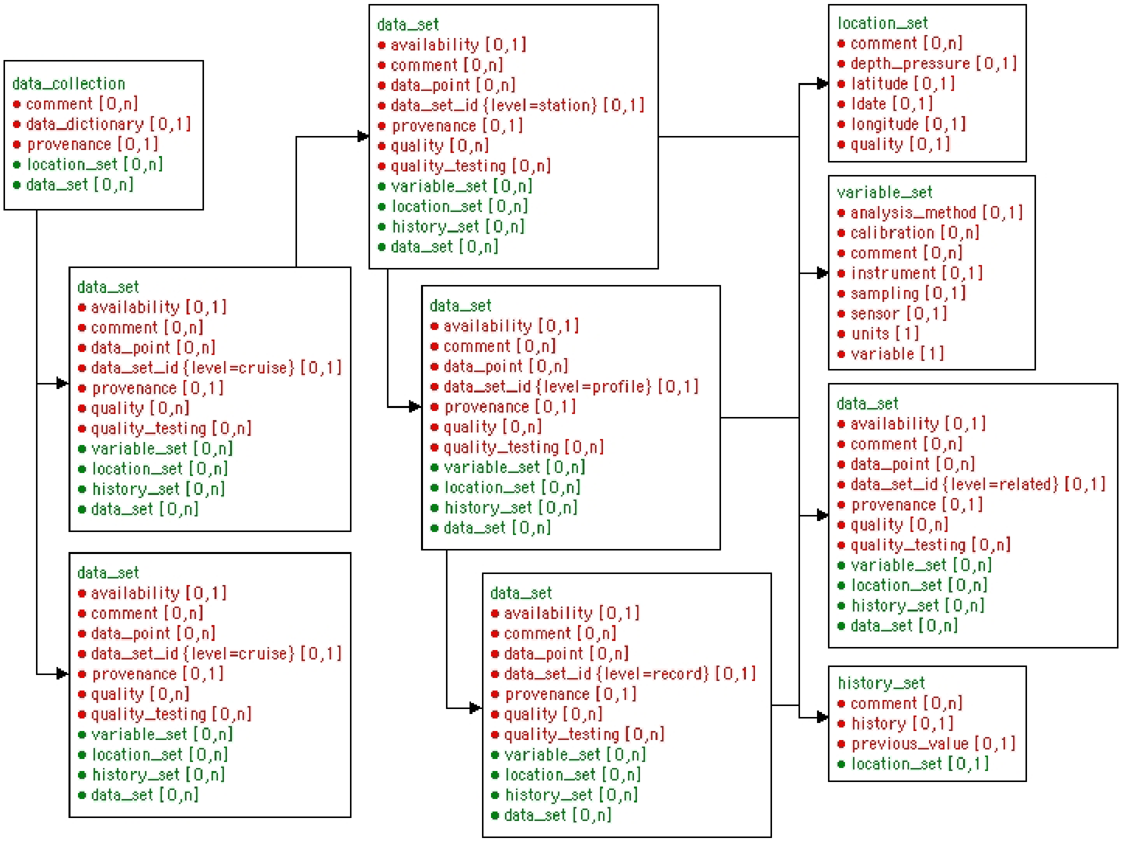


Figure 6. The profile data structure using bricks. The red text indicates pure bricks, while the green text indicates compound bricks. Compound bricks may also be identified by the name, which typically ends in the string "_set" (the exception being data_collection, which is also a compound brick). The [i,j] notation indicates the [minimum, maximum] occurrence of the brick. The arrows indicate an expansion of one element into subelements in the XML application. The details and definitions of the bricks may be found in Annex 1.

Several features of the structure presented in Figure 6 are worth noting. First, is the difference between optimization and compliance. Consider the data from two profiles at two different station locations. Consider that each profile has temperature measured as a function of pressure. The structure presented in Figure 6 would allow the inclusion of the variable information at the profile level, the station level or the cruise level. If included at the profile or station level, the variable information would need to be repeated within the XML document. If included at the cruise level, the variable information would only occur once. In all cases, the data in the XML document comply with the structure in Figure 6. However, only by including the variable information at the cruise level is the data optimized in the structure.

This difference between compliance and optimization is important for those constructing the XML instance documents. In all cases, every effort should be made to optimize the content within the structure.

The second noteworthy point is related to the hierarchical nature of the structure. The compound brick data_set is repeatable at many levels, and within this compound brick are other compounds, for example, variable_set and location_set. The hierarchy allows the specification of these compound bricks at high levels of the structure (e.g., the {level="cruise"} in the data_set_id brick). However, overriding mechanisms, similar to object programs overriding methods, allows the inclusion of these compound bricks at lower levels.

An example of this overriding is useful for our understanding. Consider a profile dataset of water samples. Suppose the time spent between collection (e.g., the closing of the bottle) and the chemical analysis of a subsample is important. Obviously, the profile has a start time, when the cast began. This time would be recorded at the {level="profile"} in the data_set compound brick. However, the time of the bottle closure must also be recorded, as this will be used to determine the time between collection and analysis. The collection time will be stored at the {level="record"} in the data_set compound brick. The record time will override the time provided at the {level="profile"}. In the case where no time is provided at the {level="record"}, the time at the start of the profile will be considered the collection time of the data in that profile.

6.6.3 The Profile Schema

The translation of the generic bricks into a structure to represent profile data involves a process of setting constraints on how bricks may be assembled. In one sense, this goes against the principal of generic bricks that can be assembled as required. However, by imposing more rigidity, we are able to exploit widely available tools built to deal with XML documents.

The specification of an XML structure is made via an XML schema definition. A *schema* contains the encoding and rules that are used to verify that the XML document is properly constructed. In XML terminology, the XML document may be validated using the schema.

There is enormous advantage to defining a schema. The schema language, which itself is based on XML, provides the parsing software with the rules of structure and allowable content. These parsers, which are freely available on-line, remove a considerable effort from the process of implementing a data exchange system.

From the structure in Figure 6, a schema description can be written (see Annex 4) and used to validate that an XML document conforms to this structure. Indeed, there are web sites (http://www.gotdotnet.com/services/xsdvalidator/) where you can provide the schema and the file to be checked, and a report is generated on-line showing if there are any problems in the file.

Such a checker can only determine if an XML document meets the structure rules defined by the schema. Content rules based on the schema are also validated, but such rules typically only define variable types (e.g., integer, decimal values), formats (e.g., date or time format) or restricted list content (e.g., some content is restricted to a predefined list of allowable content). The validation process cannot verify that the provided content makes sense. Still, it provides a first level check that a file is correctly built.

The schema provided in Annex 4 uses the Venetian Blind technique [7]. This technique for building a schema defines objects that may be used to construct the structures. This technique is perfectly suited for using the bricks to define structures. The technique also allows expansion into other namespaces if required.

6.7 Mandatory Information in the Profile Brick Structure

Another requirement in going from the generic bricks to a profile implementation is to determine what bricks, compound bricks, sub-elements and attributes (we will term all of these *components*) must be present. In the previous discussion of structures and schemas, we noted the occurrences of bricks within the structure. In defining the occurrence of components, we have attempted to consider what must be present if a component is used. By examining each component, we identified the lowest set of information that would be present if that component were to be used in a structure.

For example, consider the variable_set compound brick (Figure 6). The use of this compound brick in a structure would require some specification about the variable. There is no point using a variable_set compound brick if no information is given about the name of the variable or what units of measurement were used. Thus, the variable and units brick occurrence is set to one in the variable_set.

Turning to the contents of bricks, attributes and sub-elements, the same criteria were used. To extend the variable example, the variable brick (see Annex 1) must have the varaiable_name sub-element, as well as the typing and pt_code attributes. These attributes specify the type of variable (e.g., string, numeric, date, etc.) and the parameter code associated with the variable.

Annex 1 provides the text description of the information that must be provided whenever a brick is used. Annex 4 provides a schema description of the same information. All requirements built into the schema are used to validate the content of the XML document. The specification of mandatory components identifies the minimum information set of the profile data structure.

Schemas are constructed to formalize the XML document structure and to validate an XML document against the specified content within the schema. This level of checking was used for some attribute content within the profile data structure. For example, the latitude, longitude and ldate bricks were defined to contain an attribute named “property”. This attribute defines a qualifier that indicates a specific event. The content of the property attribute was restricted to be: start, bottom, end, creation or original. Only those strings may be present within the property attribute.

In XSD, this type of restriction on the content is termed an *enumerated list*. In a database structure, the same functionality is obtained using foreign key links to domain tables.

Most of the attributes and sub-elements within the profile data XSD do not have restrictions. Thus, much of the content within the XML document is not checked against predefined content. This lack of checking means it is possible to insert nonsense values into an XML document and still have the file pass validation against the schema.

If a brick is mandatory, it must be present in the XML document even if there is nothing to report. Likewise, if an attribute or element is mandatory in a brick, they must be present. The variable_set brick provides an example. If there is a variable_set brick, there must be units and variable bricks. The units brick must have the pt_code and stored_units attributes present. Since neither of these has restrictions, they can appear as pt_code="" and stored_units="" although this is not advised.

Optional components need only be present if there is content information available. If there is not, they should not be written to the XML document. For example, if there is no comment to be made in a data_set, the form <comment/> should not be used.

6.7.1 Variables and Instruments

In writing profile information into the XML document, the variables measured and the instruments used may be described. Some instruments, such as a CTD, can be used to measure a variety of variables, and in this case, being able to report multiple variables for a single instrument would be an advantage. However, there are other occasions when the same variable is measured by multiple instruments, such as temperature profiles measured by XBT, bottle, CTD, or thermistor chain. In this case the opposite construction would be better suited.

In this project, we chose to construct bricks where one variable is associated with one instrument. So, each measurement of a temperature requires a compound brick to describe the name of the variable. The instrument used may also be detailed within the variable_set compound brick. This means repeating information, but it also means there is a clear link between the variable, instrument and other information that applies to that measurement.

6.7.2 Use of pt_link

This attribute appears in a number of bricks and may be used to distinguish different measurements of the same variable. This attribute could be used to distinguish two different measurements from the same instrument (e.g., dual sensors on a CTD) or replicate water sample measurements.

As an example, consider the case of a temperature measured by reversing thermometer and a temperature measured by CTD, in the variable_set brick we would report the variable to be temperature in both cases. In the variable_set brick used to describe the measurements made by reversing thermometer, the attribute pt_code in the variable brick might be TEMP for temperature and pt_link could be set to "1". In the variable_set for temperature measured using a CTD, in the variable brick the attribute pt_code is again TEMP but pt_link could be "2". Now, when reporting the values using the data_point brick, a relationship may be formed between those temperatures measured with a reversing thermometer by using pt_link="1" and for CTD temperature using pt_link="2".

6.7.3 Use of pt_code

The attribute pt_code is identified as a kind of alias for the name of the variable described in the variable_set compound brick. For example, we can use pt_code="TEMP" when the variable_name is "water temperature". However, there is often information recorded about the cruise, station, etc., that are not measurements of physical variables but which are valuable to record. For example, information such as the reliability of the position information, or more details about how the data reached the archive. In this case, variable_set should be used to describe the information. In the case of more information about a station, there is no variable or units information to report. Still, because these are mandatory bricks, they must be present in the variable_set brick and must have at least the mandatory attributes and elements present. In this way, the pt_code attribute allows internal definitions of variables that extend beyond the declared data dictionary.

This extension of variables beyond the data dictionary has implications. Software developed to manipulate the documents may not know if a code originates within the declared dictionary or is defined internally. The software cannot distinguish the codes without a link to a defined dictionary. However, this does allow flexibility for the data provider.

Information relating to internally defined variables may also be stored in the comment brick. In some cases this is a legitimate solution. The advantage of placing such information into the greater formalism of a variable_set compound brick is that it allows software to manage the information and permits manipulation of data based on the content. If the information is stored in a comment brick, there is no restriction on what form it can take and so no way to guarantee to software that certain information may be present or not. The choice of whether to use a comment or a variable_set brick is up to the creator of the XML document. However, the guiding principle should be that if software will be used to manipulate or retrieve information in the XML document, then the information should be placed in a variable_set brick and not in a comment.

6.7.4 Variable Sets and Data Points

There are two bricks within the profile structure that are linked, these being the variable and the data_point bricks. Within the profile structure, every data_point brick as defined by the pt_code and pt_link attributes, must have a corresponding variable brick. The variable and data_point bricks do not need to exist at the same levels in the structure. However, the variable_set brick must exist at a level equivalent to or higher than the corresponding data_point brick. This is a 1:1 relationship, so the reverse is also true where each variable brick must have a similar data_point brick somewhere within the structure.

This requirement addresses the issue of variable accountability. This is not a requirement of the XML structure, but rather a requirement of some formats being created from the XML structure. Many in-house formats need to define the variables in the data file in some type of header. Thus, a complete list of variables needs to be formed before the content of those variables is identified. This 1:1 relationship between variable_set and data_point provides a mechanism to make this linkage.

6.8 Specific Experiences

Each of the three organizations involved in this project have their own in-house formats for profile data. At BIO, the standard format for profile data is called the

Ocean Data Format (ODF). At IOS, the standard form for profile data is the IOS Header. At MEDS, the format is referred to as MEDS ASCII.

The specific experiences covered in this section deal with the transformation from local in-house formats to the XML structure and also from the XML structure to the in-house formats. Both aspects are necessary for the proper exchange of data through this mechanism.

6.8.1 MEDS Ocean Data Format

The work to transform data from MEDS ocean data format to and from XML was carried out under contract with Matthew Nicoll of Cypher Consulting in British Columbia. He also did the work for the IOS data format. Parts of his design document [8] are included here to explain the work carried out.

It was recognized by MEDS that there were two results that would stem from this contract. The first, and perhaps most important at this stage, was a third party skilled in IT would be reviewing the idea of XML bricks and working to realize a practical implementation. In this respect, the contract was very successful with many cogent comments and questions received.

The second result was the software that would accomplish conversion of MEDS data to XML and back. Due to time constraints, the complete mapping was not achieved. However, the software to transfer most of the information was completed. Some additional work will be required to complete this software.

6.8.1.1 Software Environment

MEDS conducts software development in Fortran. MEDS operate on HP Alpha/VMS networked to Intel/Windows. Under VMS, MEDS has Fortran 90. MEDS does little or no Fortran development in Windows.

MEDS has Fortran structures defined for the in-house file format, and Fortran code for reading and writing MEDS Ocean Processing ASCII and Binary files. There are also programs for converting back and forth between binary and ASCII formats on the Alpha.

Microsoft Windows comes with an XML Parser in the form of a DLL. The latest version is called MSXML4.DLL, and is available as a free download from Microsoft. The parser:

- implements the W3C Document Object Model (DOM) for XML,

- implements the Simple Application Programming Interface (API) for XML (SAX),

- provides XSD support,

- provides additional methods to support XSLT, XPath, namespaces, and data types,

- exposes its objects as Common Object Model (COM) interfaces (COM is a software architecture that allows the components made by different software vendors to be combined into a variety of applications), and

- also exposes its objects as Automation objects, via the IDispatch interface (an Automation object is in fact a COM object that implements the interface IDispatch).

Compaq Visual Fortran (CVF) 6.6 comes with a utility called the Module Wizard, which analyses a COM-enabled file such as MSXML4.DLL, and produces a Fortran 90 Module containing all the type declarations, routine Interfaces and wrapper routines to enable the DLL and its objects to be accessed fairly easily from Fortran. There are a number of options in the Module Wizard for the "source of OLE Type information". Initially the "Type Library containing COM interface information" option was chosen. When run on MSXML4.DLL with this option, the wizard created a Fortran source file containing about 22,000 lines of code. It compiled with a few additional hours of work. The COM interface worked fine for the initial testing, but later began producing "Out of Memory" errors. So (after much head scratching and web scanning) the Module Wizard was run again, using the "Type Library containing Automation information" option. The generated module required only a minor modification to compile and has since worked without problems.

XSLT is an XML-based language that enables the transformation of one class of XML document to another. It can also be used to transform an XML file to a text file.

Basically, an XSLT file is coded as a template of the target ASCII file, and embeds instructions for extracting data from an XML file. Then, in Windows, the MSXML4.DLL can be used to do the transformation. Other implementations of the XSL standard could be used on other platforms.

Microsoft provides an XSLT Command Line Utility, MSXSL.EXE that performs command-line XSL transformations using the Microsoft XSL processor. MSXSL is a small (~11K) command-line utility that invokes MSXML4.DLL to perform the actual work of the transformation.

XSLT would be quite feasible for simple cases where there was one XML file for each output file, but in complex cases, with multiple cruises and stations, using a full-featured programming language is likely to be more efficient. To create the rigidly structured MEDS ASCII file would at best be a tedious process with XSLT. It makes sense to use the existing Fortran code for that task. Since XSLT is designed to transform from XML files, it is of no help in transforming from IOS or MEDS ASCII files to XML.

Given that DIGITAL Fortran 77 will compile on both the VMS platforms and on Intel/Windows, a case could be made for writing the programs exclusively in that language, to maximize portability. Such a solution would also enable usage with MEDS binary files on the Alpha. An Internet search for XML parsers written in Fortran yielded nothing, so an all-Fortran, highly portable solution would entail writing code to read, parse and write XML files. This would greatly increase the cost of the project.

Having verified that the MSXML4 DLL can be used from Fortran, and having been assured that converting MEDS binary files to ASCII for XML conversion is not a problem, the recommendation was to create Intel/Windows/MSXML based programs, using Compaq Visual FORTRAN 6.6.

6.8.1.2 Mapping from MEDS Terminology to XML

Considerable topic specific terminology is used within any organisation. Within an oceanographic data centre one frequently used term is "station". MEDS uses the term "station" as:

- the place where a single-location event occurs,
- the fixed portion of each station record, which contains the latitude and longitude, is called the "station" section, and
- the "station" is identified uniquely within a cruise with a "station number" (stn_number), which is actually a consecutive number of the event within the cruise.

The brick definitions also frequently use "station". In the XML implementation of the bricks:

- a station-level data_set element is simply a set of all data from the same latitude and longitude (thus "station" has the same meaning as in a MEDS file),
- the word "station" is used as an attribute value in the data_set_id element,
- a station-level data_set, containing location information, will be created in a target XML file for each source MEDS station or IOS file, and
- the text of the station-level data_set_id element will be the name of the station, if the station has a name.

An example of “station” usage within the XML profile structure is:

```
<data_set>
        <data_set_id level="station">Bravo</data_set_id>
```

If the data comes from an un-named location, or if there is no name available (as in a MEDS file), then there will be no station-level data_set_id text. In this case, the XML structure will look like:

```
<data_set>
        <data_set_id level="station"/>
```

6.8.1.3   Mapping from MEDS Format to XML

The mapping of one data structure to another is a difficult task. The details of this task are often important for future modifications and as such those details for the MEDS mapping are presented in Annex 5.

For example, some characteristics of the input MEDS files include:

- each file contains one or more profiles, one variable per profile,
- variables are called "profile type" , "parameter" or "type of data",
- each variable is identified with a 4-character code (prof_type) which, via a lookup-table, implies the data type and its units,
- each profile consists of a depth (or pressure) vector, and a vector of corresponding dependent variable values,
- the depth (or pressure) vectors are usually, but not necessarily identical over all profiles,
- if all depth (or pressure) vectors are identical, the file could be said to contain a single multi-variable profile, and
- the cruise, date and time information in each profile record can be assumed to be the same as in the fixed Station record.

Where as in the XML bricks:

- each station data set may contain zero or more profile data sets,

- location (latitude and longitude) and variable information will be looked for at the station and the profile levels,

- variables are identified by the pt_code attribute, and optionally the pt_link attribute. If two or more variables have the same pt_code, the pt_link can be used to uniquely identify them,

- use the pt_code unmodified as the target variable/channel identifier. (i.e., any translations must already be done),

- The station-level data_set will contain a single, multi-variable profile data_set, except for the case where a MEDS file has profiles with different depth vectors, in which case there will be multiple profiles,

- Location information will be stored at the station level,

- Variable information will be stored at the profile level,

- The MEDS profile type will become the XML pt_code, and

- The profile-level data_set_id text will contain the event number (MEDS stn_number). Thus in multi-profile cases each profile will have the same data_set_id.

6.8.1.4 Software Implementation

The MEDS programs will have simple command-line interfaces:

To convert a single MEDS file to XML:

```
meds2xml meds_ascii_file.txt  output_file_name.xml
```

To convert multiple files named one per line in file meds_file_list.txt:

```
meds2xml @meds_file_list.txt  output_file_name.xml
```

Log information, by default, will be written to standard output, but if a log file name is specified as a third argument, log information will be written to that file instead.

A scheme could be devised for creating a default output xml file name from cruise or other information in the input files.

To convert from XML:

```
xml2meds input_file_name.xml
```

The output file names will be generated using a combination of the input file name and/or cruise and profile number information found in the XML file.

All XML file I/O, parsing and creating will be done using the freely available Microsoft MSXML4.DLL XML parser. An example XML document produced from data in the MEDS archive system is shown in Annex 6.

If information is to be stored at the highest possible XML data_set level, then one of the following strategies would be required when converting multiple input files to a single XML file:

1. Examine header information from all input files before doing any XML data_set creation, and put information which is uniform across all files at the cruise or collection level.

2. a) make the XML structure mirror the input structure where possible, duplicating information as it is duplicated in the input files, then,

   b) (optionally) modify the XML hierarchy to promote duplicated elements.

The second method was selected because:

- it separates problems, reducing complexity, thus enhancing reliability and maintainability,
- when step 2b is not desired, having it as a distinct step makes it easy to program as an option, and
- the same code for step 2b could be used by both the IOS and MEDS to XML programs, and conceivably, by other programs written in the future.

6.8.2 IOS Header

The work to transform data from IOS Header data format to and from XML was carried out under contract with Matthew Nicoll of Cypher Consulting in British Columbia. The comments in the MEDS section above relating to choice of programming tools apply in the most part to the IOS situation. A Windows-based solution was an easy choice because of the large amount of software development and data processing that is done using Windows and Fortran at IOS.

6.8.2.1 Computing Environment

IOS operates on Intel/Windows, Alpha/VMS, and VAX/VMS. Under VMS, IOS has DIGITAL Fortran 77 V6.5-199. IOS uses Compaq Visual FORTRAN V6.6 in Windows.

6.8.2.2 The IOS Header Format

An IOS Header file consists of an ASCII header, followed by ASCII data columns. Alternatively, the data can be in a separate file, in which case it can be binary. For profile data, all the data at one depth is in one record. The header is designed to be readable by both human and computer. A header may be as small as 10 lines, or may, if extensive comments are included, contain thousands of lines. Each column of data is called a channel, and each channel is defined in the header. Data may be of various types, including Real, Integer, Date, Time, and Character.

There is a library of Fortran routines called the IOS Header Library, which can be used for reading, writing and manipulating files in the IOS Header format. The library is written in Fortran, and is implemented as an object-module library with structure definitions, in both VMS and Windows. There is also a Windows DLL version of the library, and Visual Basic versions of the include files, which allow the code to be used from Visual Basic (see [9]).

6.8.2.3 Mapping between IOS Format and XML

IOS has separate data files for each profiling event (e.g., CTD cast). Thus, many IOS files can be placed into one XML file. For the details of this mapping from IOS to XML structures, see Annex 7.

6.8.2.3.1 Stations and Profiles

Each IOS file contains a multi-variable profile, located with a latitude and longitude. Hence, the station/profile XML dataset hierarchy was not utilized. One XML station dataset, containing just one profile dataset, was created for each IOS file. When converting from XML to IOS, one IOS file was created for each profile dataset. The station dataset was treated as just another level to search for information not present at the profile level. When converting from IOS to XML, location information was placed into the station level, and variable information into the profile level.

Theoretically, IOS files could be grouped by station so that location information could be raised to the station level in XML. However, different casts of data collected at one "station" are not likely to have exactly the same latitude and longitudes.

6.8.2.3.2 Variables and Channels

This is an area where the IOS files are more programmer-friendly than the XML. IOS channel definitions are one-to-one with each type of profile data. By examining the channel table in the IOS Header, a program can determine exactly how many channels of data there are to read, the type of data, ranges, pad values, etc. In the XML file, variable bricks describe the profile data to be found at the record level, but there may also be variable bricks describing single data values stored at the profile or station levels. There is also less data formatting detail in the XML variable brick. It was necessary, therefore, to scan all the record datasets in each profile twice - once to find out what variables are present and to get details about the data; and then again to actually read the data into the storage prepared for them. Late in the investigation, it was realized that the ancestor-or-self functionality could have been used to avoid this problem.

6.8.2.3.3 Instrument Information

This is an area where the XML design is more programmer-friendly than the IOS files. In XML, each variable can be linked to detailed instrument information. The IOS file design grew from a scheme where there was one file for one (multi-sensored) instrument. Hence, there is a general instrument section in the header, where individual sensors are named, but are not linked directly to channels. A human reader of the header can usually make this link. If and when data from different instruments are combined into a single IOS file, there is increased likelihood of losing track of the instrument sensor that is related to a particular channel.

When converting to XML, an attempt is made in the Fortran code to match sensor names to channel names, but this will not be reliable in all cases.

6.8.3 Bedford Institute – Ocean Data Format

The investigation to transform ODF profile data to XML was conducted using the Ocean Sciences Division (OSD) Matlab based set of tools called the Oceans Data System (ODS) Toolbox. This suite of tools has been developed by OSD over the past decade and constitutes the primary analysis tool within the Division. The Toolbox is capable of reading ODF files and creating a memory resident ODF cell structure within the Matlab environment.

Matlab scripts and functions were written to take the contents of the cell structure and produce the XML structure for profile data. These scripts were developed in Matlab 5 and incorporated into the ODS Toolbox.

The transformation from XML to ODF was conducted using two different methods. The first method used XSLT, an XML-based language designed to manipulate the structure of an XML document. This method was developed as an exercise in understanding the usefulness and versatility of XSLT and was not considered the most practical method of dealing with the brick objects.

The second method for transforming the XML to ODF utilized Java technology. This method is also quite economical, as many Java tools and software components are freely available.

There are many Java packages available for the manipulation of XML documents. Java implementations of the DOM and the SAX are available on the Internet. As well, data binding technology is particularly well suited to XML manipulation. Data binding was chosen for this investigation.

Data Binding is a technique for linking, and automatically creating, Java classes based on a constraint file. In the case of XML, the constraint file is the document type definition (DTD) or Schema. Available software will read the constraint file and generate the Java code for the Java classes. The classes are based on objects, and those objects are represented in the constraint file. In the case of the bricks, the generated Java objects are the bricks.

The class generation also provides access methods to the content of the bricks. These access methods are named based on the constraint file brick definitions. The result of the binding is the removal of complicated data access methods via calls to nodes in the XML structure. Instead, the binding creates more descriptive classes based on the brick names. So, classes were created like DataCollection. As well, binding results in method calls such as object.getProvenance().getDescription() (where object is a DataCollection or a DataSetCbrick, see Annex 4) to obtain the description data within the provenance brick (see Figure 3 for contents of the provenance brick). This provides the programmer with a more intuitive set of classes and methods for accessing the content of the XML document.

Java based data binding frameworks are also freely available on the Internet. In particular, two products were examined: Zeus and Sun's Java Architecture for XML Binding (JAXB). In the brief review, the Zeus documentation identified support for binding with a DTD, but not a schema. This product was not considered further (later we discovered that the documentation had not been updated, and in fact the product did support binding from a schema).

At the beginning of this development, the Sun JAXB product had passed Beta testing. Over the course of the development, JAXB was finalized.

After the classes and methods were created in the binding process, a wrapper program, called ProcessDataCollection, was created. ProcessDataCollection controlled the transfer (unmarshalling) of the XML document into Java objects. ProcessDataCollection also controlled the access of the Java objects to create a new file in the in-house ODF format.

6.9 Implications of XML Exchange

We conclude with several interesting points discovered during the development process. Ideally, these points may be used to refine the bricks and structure developed from the bricks. Also, some points may provide useful insight for those wanting to map their in-house formats to the XML structure.

6.9.1 Specific Tag Names

The BIO XSLT investigation uncovered some interesting aspects of the structure design. In particular, one design issue that resulted from SGXML comments on the initial profile data structure and bricks, dealt with the generalization of the data values. Initially, the generalization of data points resulted in (x,y,z,t) values having no special consideration. In this way, all positional information was included in the data_point brick by specifying the (x,y,z,t) particulars using the pt_code attribute. However, comments from SGXML resulted in special emphasis being placed on these variables, and the subsequent creation of the longitude, latitude, depth_pressure and ldate bricks.

The latitude, longitude and ldate bricks were sufficiently specific to remove the requirement for the pt_code attribute. Although seemingly innocuous, the removal of the pt_code attribute caused problems with the XSLT code for transforming the XML to ODF. Specifically, the pt_code (and pt_link) attribute permitted the sorting of the elements. This was very important for the output ODF structure, as ODF requires the header variable definitions in the ODF file to be in the same order as the variable columns. Using a sort on pt_code for both variable and data_point information resulted in a consistent ordering of the variable information in the ODF file. The removal of the pt_code attribute in the latitude, longitude and depth_pressure bricks, complicates the XSLT code immensely.

6.9.2 Inadequate In-House Formats

Another realization during the process dealt with the numerous inadequacies with the in-house formats. These formats were developed to maintain local processing and archive systems. The data model, from which many evolved, was not oriented towards general ocean data, but more toward specific ocean data types. This has caused problems.

As an example, the ODF format does not have the ability to store information on more than one instrument. This implies that the data in a single ODF file must originate from a single instrument. However, this is often not the case. In a water sample file, pressure is often included from the CTD data stream, while other variables are true water sample values. The XML structure allows unique data types to identify unique instruments. This more generalized structure does not map well to the ODF format.

Other examples exist where the source file is lacking explicit information to map to the XML structure. For example, in the IOS file the distinction between an oxygen measurement derived from a water sample and one from an oxygen probe is encoded in the name of the variable but nowhere in the file (or any file) is this encoding written. Another example deals with serial numbers, where in the IOS file such numbers are typically embedded in free text. This is not something that software can find, even though the XML file has a place for the information.

6.9.3 Parameter Code Mapping

The development presented here only provides a data structure for encapsulating ocean data. When exchanging data within a common structure, the problems associated with parameter codes are still present. For example, one institute may refer to particular organic carbon as CPX1 in milligrams/litre while another institute refers to the same data as Carbon:Particulate:Organic in micrograms/litre. Developments are ongoing to convert these code sets using XML [10].

6.9.4 Null Values

Null values are not XML friendly. In the old paradigm of data formats, null values were defined to fill the space of the "missing" data value. These null values were typically outside the space of realizable data values. For example, latitude stored in degrees might have an associated null value of –99. The –99 is not in the realizable range of latitude values expressed in degrees. In XML, one may define restrictions to set the allowable limits of data values. If such a schema restriction were placed on latitude for the range –90 to 90, the null value of –99 would not pass validation.

Also, implementing this restriction would not allow an empty tag to be present in the XML document. In XML, all tag content is used in validation, including empty content.

This functionality has consequences for the mandatory set of tags defined for the bricks. If, in the schema, a tag is declared as mandatory with restrictions, then the tag must be present with valid content in the XML document. The input data streams therefore must have that content available.

6.9.5 XML Content for Output Format

The developed XML profile structure must allow for the final output formats being created based on the XML content. In some cases, this resulted in the bricks being modified and thereby containing some components that are not, strictly speaking, part of the content that one would expect to be exchanged. For example, the variable brick contains a sub-element named decimal_places. This sub-element contains the number of decimal places in the specific variable. This formatting type information is not a requirement of the XML structure nor is it a natural component of a definition for a variable. Rather, it is required to meet the needs of the formats being created from the XML document. Another example is the typing attribute.

6.9.6 Units

Within the units brick is an attribute named stored_units. stored_units is presently mandatory within the units brick. In XML, this means it must be present. Since there are no restrictions on this attribute, it may be present with empty content (e.g., stored_units=""). If content within stored_units is mandatory, then those variables that do not have units (e.g., indexes, sample ID numbers) will require a designator that indicates no units. The designers must accept a condition that allows no content, or a unitless variable having a null value for the unit.

6.9.7 Brick Order

The order of the bricks within the profile structure was arbitrary. However, having data_set_id within an alphabetical listing results in its placement after the comment brick. In this ordering, it is easy to mistake at which level the comment occurs. Depending on how strongly people feel about being able to open the XML and read it manually, as opposed to letting software do this, we may want to break the alphabetical ordering rule to insist data_set_id brick occurs first in the data_set brick.

6.9.8 Quality in location_set

Quality in location_set is a problem. The MEDS archive structure allows separate quality flags for position, time and record. In location_set, the profile structure only allows a single quality code for all the location_set content. This is not adequate and should be revised.

6.10 Future Work

There are many possible avenues for future work. Some initial topics that we plan to investigate include:

- Bricks applied to object-oriented databases (OODB) – OODBs represent a storage mechanism that can utilise the hierarchical nature of datasets. These databases may be used to construct a database that reflects the encapsulation of bricks within bricks.

- Extension to other data types – The usefulness of the bricks in other data types needs to be addressed. Examining whether or not the bricks can be applied to other types will help assess the principle behind assembling bricks into data structures for other ocean data types.

- Standardization for data exchange – The XML structures that we have proposed here go part way to providing needed standards. In this case, we have proposed a standard formatting scheme that is very capable of carrying both data and information about the data. Having such a standard simplifies the processing needed when data are received since now a single program can read the file created by any number of sources. A huge gain in efficiency will result as the XML is adopted.

- Formal definitions for data exchange – The formally defined structures in the XML file allow for detailed information that describes the data and collection process. This formalism in description is one step in reducing the ambiguity and imprecision that can be found in exchanged data files. In this way, the XML structure is furthering the reliability of the correct transcription of data from one form to another.

- Simplification and fidelity of the exchange – As part of the definition process of the XML file, we quickly realized the partners in the study use more than one data dictionary. We chose to build in the capability to deal with more than one dictionary, rather than resolve differences during the study. It is clear that ambiguity is reduced if partners can reach agreement on names for variables, preferred units, quality indicators, etc. Though the desirability of such standardization has long been recognized, it is only now when data exchange has become so easy, quick and ubiquitous, that the pressure may finally be sufficiently high to force agreement. We need to capitalize on this pressure.

7. KEY RESULTS ACHEIVED

There are several key results from this work:

- We have designed and tested a method of exchanging ocean profile data in an XML environment.

- The design has developed the concept of a data object or brick, for oceanographic data. These bricks, if carefully defined, can represent common components across multiple data types. We expect this technique can be applied to other ocean data structures such as imagery, time series, etc.

- The testing included an examination of the design feasibility using actual data and common programming languages. Three different programming languages were used to implement the system including Fortran using MSXML, Java data binding objects and XSLT.

- There has also been progress on the implementation of the SGXML parameter dictionary structure. The SGXML structure was expanded slightly to account for direct mapping between dictionaries used at the three labs involved in this project. The expansion allows an XML based translation from one organizations parameter dictionary to another. For details, see [10]

- Another finding deals with the development of the required software. Developing the software required to transform in-house datasets to and from the XML structure is not a difficult task, nor is it expensive. Thus, once the intellectual effort is expended in the mappings, the dollar expense of joining the community using the XML structure is not prohibitive.

8. IMPACT OF THE PROJECT

This project will have major influence in the field of ocean data management. Specifically, the project has impact in the following areas.

- The project has advanced the exchange of data by showing that the XML can be used to define a general structure capable of reliability and efficiently transporting data between diverse labs. The general structure could be applied to any ocean profile data. Thus, interest in this work will exist within organizations interested in data exchange, such as IODE, ICES and JCOMM.

- The project will also have an impact on the planning for a distributed ocean data system. In the global ocean data community, some proposals for a distributed ocean data management system are being considered. Such a system will need to query diverse data holdings, generating a common output structure for the data stream and rationalizing the code system on which the different data output are based. The XML structure developed in this project could be used for the data streams. As well, the code mappings [10] that were briefly explored in this project could form the foundation for a fuller investigation.

9. TRANSFER OF KNOWLEDGE/TECHNOLOGY

This work was first presented to the IODE at the IODE XVII meeting in Paris, March 3-7, 2003. We are grateful to Bob Gelfeld (US National Oceanographic Data Centre), co-chair of the ICES-IOC SGXML, for presenting our work to the IODE Committee. A copy of the presentation is included in Annex 9.

The results of this work have also been presented to Defence R&D Canada – Atlantic as part of the DRDC Atlantic Seminar series (March 28, 2003).

The ICES-IOC SGXML will also receive a briefing [11] on this activity at the annual meeting in May 2003 (Sweden). The SGXML have provided critical review of past efforts. As well, some of this work uses results from SGXML activities.

A web site has been established to direct people to both key results and details of this effort. The web site is hosted by MEDS, and can be accessed through the existing MEDS site [12].

Finally, copies of this report have been distributed to key individuals within the ICES and IODE communities.

10. EXPENDITURES BY YEAR

Table 3. Expenditures by year.

CLASS		2002/03	2003/04	TOTAL
SSF 2001	SAL ($K)	20		20
	O&M ($K)			
	CAP ($K)			
	Shiptime ($K)			
	Total ($K)	20		20
Regional A-Base Invested	SAL ($K)	20		20
	O&M ($K)			
	CAP ($K)			
	Shiptime ($K)			
	Total ($K)	20		20
Funding Provided by Partners	SAL ($K)	15*	15*	30
	O&M ($K)			
	CAP ($K)			
	Shiptime ($K)			
	Total ($K)	15	15	30
Total Cost of Project	SAL ($K)	40		40
	O&M ($K)			
	CAP ($K)			
	Shiptime ($K)			
	Total ($K)	40		40

* This represents contributions from international colleagues within the ICES Study Group now considering XML for ocean data transfer.

11. GENERAL COMMENTS

Prepared by: Anthony W. Isenor, J. Robert Keeley and Joe Linguanti

Date: March 2003

12. References

1. SGXML, 2002. Report of the ICES-IOC Study Group on the Development of Marine Data Exchange Systems Using XML, Helsinki, Finland, 15–16 April 2002, ICES CM 2002/C:12.

2. Isenor, Anthony W. 2002. The Importance of Metadata in System Development and IKM, DRDC Atlantic TM-2003-011, Defence R&D Canada – Atlantic.

3. The Annotated XML Specification at http://www.xml.com/axml/testaxml.htm

4. Walsh, Norman. A Technical Introduction to XML, http://www.xml.com/lpt/a/98/10/guide0.html

5. Mitre. 2002. XML Schemas: Best Practices, Creating Extensible Content Models, The MITRE Corporation and the xml-dev list group, http://www.xfront.com/BestPracticesHomepage.html

6. Vlist, Eric van der. 2002. XML Schema, O'Reilly and Associates, Inc., ISBN 0-596-00252-1.

7. Cagle, Kurt, Michael Corning, Jason Diamond, Teun Duynstee, Oli Gauti Gudmundsson, Michael Mason, Jonathan Pinnock, Paul Spencer, Jeff Tang, and Andrew Watt. 2001. Professional XSL, Wrox Press Ltd.

8. Nicoll, Matthew. 2003. Requirements and Analysis Design, Cypher Consulting, January 2003.

9. For details see \\paciosfp2\osapshare\libraries\ios_header\doc\UserManual.doc

10. Isenor, Anthony W. 2003. XML Based Manipulation of Codes Exchanged Between Data Systems, Technical Memorandum in preparation, Defence R&D Canada – Atlantic.

11. Isenor, Anthony W. 2003. Canadian Investigations of the Keeley Bricks With Application to Profile Data Transfer Using XML, DRDC Atlantic SL 2003-029, Defence R&D Canada – Atlantic. Published in Meeting Report for the ICES-IOC Study Group on the Development of Marine Data Exchange Systems Using XML, Gothenburg, Sweden, May 26-27, 2003.

12. For details see www.meds-sdmm.dfo-mpo.gc.ca/meds/Prog_Int/ICES/ICES_e.htm

Annex 1: Definitions of Current Bricks

Current Bricks and Elements

Note: If a brick has no elements, then the brick name is considered the only element. For example, latitude.

Brick: analysis_method

Stores information about any physical, chemical or biological analyses carried out on the data

Version: 2

Element	*Atttribute*	*Constraint*	*Definition*	*Occur*
analysis_date		none	The date on which the analysis occurred	1
analysis_id		none	A sample identifier used by the analyst	0,1
analyst_name		none	The name of the person conducting the analysis	0,1
method		none	The instrument or method used to carry out the analysis	1

Brick: availability

Stores information about the possible release of the data to the public.

Version: 4

Element	*Atttribute*	*Constraint*	*Definition*	*Occur*
	indicator	see list Table	The flag that indicates the availability of the data.	1
avail_date			The date the indicator was set.	1

Brick: calibration

Stores calibration information on the instrument, sensor or variable

Version: 6

Element	*Atttribute*	*Constraint*	*Definition*	*Occur*
algorithm_type			Defines the algorithm used with the coefficients (e.g. linear, polynomial)	1
application_date			The date the calibration was applied to the data.	0,1
calibration_date			The date the calibration procedure resulted in the determined coefficients.	0,1
coefficients	name		The coefficients used in the calibration	0,n

			procedure. Name corresponds to the name of the coefficient used in the calibration expression.	
number_of_coefficients			The number of coefficients used in the calibration procedure.	0,1
process			The name of the process used in calibration of the sensor.	0,1

Brick: comment

Stores general textual information not intended to be used in data retrievals

Version: 6

Element	Atttribute	Constraint	Definition	Occur

Brick: data_collection

Compound brick used to encapsulate the entire XML file

Version: 1

Element	Atttribute	Constraint	Definition	Occur

Brick: data_dictionary

Indicates the specifics of the data dictionary being used within the file.

Version: 3

Element	Atttribute	Constraint	Definition	Occur
dictionary_name			The name of the data dictionary used in the data file.	1

Brick: data_point

Used to store any type of data or metadata value

Version: 9

Element	Atttribute	Constraint	Definition	Occur
	pt_code		A unique code for the variable.	1
	pt_link		Used to allow links to be made between 2 different measurements made from the same instrument, or two of the same measurements of variables made with different instruments	0,1
	statistic		Indicates if the value is an instantaneous or	0,1

			statistical measurement	
	typing	see list	Indicates the computer typing (real, integer, string)	0,1

Brick: data_set

Compound brick used with data_set_id to define the granularity of the data_collection

Version: 4

Element	*Atttribute*	*Constraint*	*Definition*	*Occur*

Brick: data_set_id

This brick allows for a text or numeric identifier to be attached to a particular data set.

Version: 2

Element	*Atttribute*	*Constraint*	*Definition*	*Occur*
	level	see list	Used to indicate the grouping level of the data set.	1

Brick: depth_pressure

Used to explicitly store the depth or pressure for the z coordinate

Version: 7

Element	*Atttribute*	*Constraint*	*Definition*	*Occur*
	kind		Indicates if the variable is used as an independent, or dependent variable.	0,1
	property	list	A qualifier on the position to indicate specific events like beginning or end.	0,1
	pt_code		A unique code for the variable.	1
	pt_link		Used to allow links to be made between 2 different measurements made from the same instrument, or two of the same measurements made with differrent instruments	0,1
	statistic		Indicates if the value is an instantaneous or statistical measurement	0,1
	typing		Indicates the computer typing, real, integer, string.	0,1

Brick: history

This brick stores a computer readable version of past processing. This is essentially a summary brick, summarizing a unit of processing. It is not intended for a narrative style history of the processing.

Version: 8

Element	Atttribute	Constraint	Definition	Occur
	action		The action undertaken by the process.	0,1
	pt_code		A unique code for the variable acted on by the process.	0,1
	pt_link		Used to allow links to be made between 2 different measurements made from the same instrument, or two of the same measurements of variables made with different instruments	0,1
application_date			The date of the processing.	0,1
executor			The agency recording this information.	0,1
process_identifier			Any identifier for the process responsible for creating this history record.	0,1
version			Any version number associated with the process.	0,1

Brick: history_set

A compound brick that contains the history of changed data values

Version: 1

Element	Atttribute	Constraint	Definition	Occur

Brick: instrument

Stores information about the instrument used to acquire the measurements.

Version: 7

Element	Atttribute	Constraint	Definition	Occur
	type		The type of instrument (e.g. XBT, CTD)	1
description			A description of the instrument.	0,1
manufacturer			Who makes the instrument	0,1
model			The instrument model number.	0,1
serial_number			The instrument serial number.	0,1

Brick: latitude

The position latitude in decimal degrees +- 90 where North is positive.

Version: 6

Element	Atttribute	Constraint	Definition	Occur
	kind		Indicates if the variable is used as an independent, I, or dependent, D, variable.	0,1
	property	list	A qualifier to indicate specific events like beginning or end.	0,1
	statistic		Indicates if the value is an instantaneous or statistical measurement	0,1

Brick: ldate

Version: 6

Element	Atttribute	Constraint	Definition	Occur
	property	list	A qualifier to indicate specific events like beginning or end.	0,1
	statistic		Indicates if the value is an instantaneous or statistical measurement	0,1
pdate			A date point in yyyy-mm-ddZ. Note the trailing Z is mandatory.	1,0,1
ptime			A time point in hh:mm:ss.ssZ. Note the trailing Z is mandatory.	0,1,1

Brick: location_set

Compound brick used to encapsulate x,y,z,t information

Version: 1

Element	Atttribute	Constraint	Definition	Occur

Brick: longitude

The position longitude in decimal degrees +- 180 where East is positive.

Version: 6

Element	Atttribute	Constraint	Definition	Occur
	kind		Indicates if the variable is used as an independent, I, or dependent, D, variable.	0,1
	property	list	A qualifier on the position to indicate specific events like beginning or end.	0,1
	statistic		Indicates if the value is an instantaneous or	0,1

statistical measurement

Brick: previous_value

Records the value of a measurement or information before the value was altered by some data quality checking

Version: 5

Element	Atttribute	Constraint	Definition	Occur
	pt_code		A unique code for the variable.	1
	pt_link		Used to allow links to be made between 2 different measurements made from the same instrument, or two of the same measurements of variables made with different instruments	0,1
	typing		Indicates the computer typing, real, integer, string, etc.	0,1

Brick: provenance

Records information about the source of the data

Version: 6

Element	Atttribute	Constraint	Definition	Occur
	platform_type		The type of platform.. (e.g. moored buoy, ship, airplane, etc.)	0,1
agency			The agency from which the data came. This should be as specific as possible.	1
country			The country of origin for the dataset.	0,1
data_grouping			A way to identify groupings of data. This can be used, for example, to identify stations that belong to sections.	0,1
date_created			The date that the record was created.	1
description			An open description of the dataset.	0,1
institute_code			The code representing the institute.	0,1
originator			The name of the person from which the data originated.	0,1
originator_identifier			The identifier used by the originator by which the data are identified.	0,1
platform_name			The name of the platform serving for the data collection.	0,1
project			A general name that the dataset may be associated with. For example, the project name may be related to an international or national program such as WOCE, CLIVAR.	0,1

Brick: quality

Records the results of the data quality tests performed on the data

Version: 5

Element	Atttribute	Constraint	Definition	Occur
	justification_code	see list	The code giving the justification of the reliability code assigned.	0,1
	pt_code		A unique code for the variable.	1
	pt_link		Used to allow links to be made between 2 different measurements made from the same instrument, or two of the same measurements of variables made with different instruments	0,1
	reliability_code	see list	The code giving the reliability of the value.	0,1
	use_code	see list	A code saying what are inappropriate uses of the values.	0,1
qt_date			The date the test was carried out	0,1
tests_failed			A list of the tests that failed	0,1
tests_performed			A list of ids of tests performed	0,1

Brick: quality_testing

Stores information about the tests that are performed on measurements to ensure the results are of high quality.

Version: 4

Element	Atttribute	Constraint	Definition	Occur
test_description			A description of the test.	1
test_id			An identifier used later to describe tests performed and failed	1
test_name			The name of the test employed.	1
test_version			The version of the test.	1

Brick: sampling

Stores information about the sampling technique used to acquire the measurements.

Version: 7

Element	Atttribute	Constraint	Definition	Occur
	pt_code	list (see table)	A unique code for the variable.	1
	pt_link		Used to allow links to be made between 2 different measurements made from the same instrument, or two of the same measurements of variables made with different instruments	0,1

id		none	A unique identifier used for the sampling	0,1
interval		none	Some unit interval between the observations (if regular). This may be a time or space interval depending on the application.	1
method		none	The method used for sampling the variable.	0,1

Brick: sensor

Stores information about the sensor used to make a measurement.

Version: 6

Element	*Atttribute*	*Constraint*	*Definition*	*Occur*
manufacturer			Who made the sensor.	0,1
model			The sensor model number.	0,1
serial_number			The sensor serial number.	0,1
type			The general classification of the sensor (e.g. temperature, conductivity etc.)	1

Brick: units

Stores information about the units used to record measurements.

Version: 6

Element	*Atttribute*	*Constraint*	*Definition*	*Occur*
	pt_code		A unique code for the variable.	1
	pt_link		Used to allow links to be made between 2 different measurements made from the same instrument, or two of the same measurements of variables made with different instruments	0,1
	received_units		The units in which the data were originally received.	0,1
	stored_units		The units in which the data have been archived. Normally this would be in SI.	1
conversion			The formula (using appropriate data dictionary descriptors) by which data were converted from received to stored units.	0,1
reference			A reference that explains the conversion.	0,1
variable_name			Any name the originator wishes to assign to the variable.	0,1

Brick: variable

Stores information about the variable being measured

Version: 8

Element	Atttribute	Constraint	Definition	Occur
	duplicate_indicator		Used to indicate if the variable is duplicated somewhere in the senders archive.	0,1
	kind		Indicates if the variable is used as an independent, I, or dependent, D, variable.	0,1
	pt_code		A unique code for the variable.	1
	pt_link		Used to allow links to be made between 2 different measurements made from the same instrument, or two of the same measurements of variables made with different instruments	0,1
	typing		Indicates the computer typing, real, integer, string, etc.	1
accuracy			The absolute accuracy to which the measurements have been made	0,1
below_detection			The value used to indicate that traces of the variable were found but that they could not be measured	0,1
decimal_places			The number of decimal places being used to represent the variable.	0,1
maximum_value			Maximum value for the variable in this data unit.	0,1
minimum_value			Minimum value for the variable in this data unit.	0,1
null_value			The value assigned to the variable to indicate NO DATA.	0,1
precision			The precision to which the measurements have been made.	0,1
variable_name			Any user specified name for the variable	1

Brick: variable_set

Compound brick used to declare a variable

Version: 1

Annex 2: Controlled Lists

Attribute Constraint List

Attribute action

list_values	*note*	*brick*
see ACT_CODES list	see ACT_CODES list	history

Attribute duplicate_indicator

list_values	*note*	*brick*
D	this is a less desirable copy of data held elsewhere	variable
N	this is not duplicated elsewhere	variable

Attribute indicator

list_values	*note*	*brick*
C	Consult	availability
O	Open	availability
R	restricted to originator	availability

Attribute institute_code

list_values	*note*	*brick*
02PA = Institute 02, Papa data		provenance
9999 = Unknown		provenance

ABS = Arctic Biological Station (Montreal) now closed	provenance
AINA = Arctic Institute of North America	provenance
BIO = Bedford Institute of Oceanography (Halifax)	provenance
BSH = Bundesamt fuer Seeschiffahrt und Hydro.	provenance
CANA = Canadian Navy	provenance
DALU = Dalhousie University	provenance
DOT = Department of Transport	provenance
DREA = Defense Research Establishment, Atlantic	provenance
DREP = Defense Research Establishment, Pacific	provenance
FROR = Orstom, Brest, France	provenance
FWI = Fresh Water Institute (Winnipeg)	provenance
GIRO = Groupe Interuniversitaire de Recherche Oceanographiques de Quebec	provenance
ICES = ICES	provenance
IFMK - Institut fur Meereskunde Kiel	provenance
IIP = International Ice Patrol	provenance
IML = Institut Maurice Lamontagne (Rimouski)	provenance
INCO = INCOIS - Indian research institute	provenance
IOS = Institute of Ocean Sciences (Patricia Bay, B.C.)	provenance
JFA = Japan FIsheries Agency	provenance
MEDS = Marine Environmental Data Service	provenance
MEMU = Memorial University (Nfld)	provenance
MSBO = Marine Sciences Branch, Ottawa	provenance
MSC = Marine Sciences Centre, McGill University	provenance
NAFC = Northwest Atlantic Fisheries Centre (NFLD)	provenance
NIOI = National Institute of Oceanography, India	provenance
NODC = National Oceanographic Data Center (U.S.)	provenance
ODU = Old Dominion Univ, USA	provenance
RUSP = PINRO in Russia	provenance

SAND = St Andrews Biological Station (N.B.)		provenance
SOC = Southampton Oceanographic Centre, UK		provenance
SVP = SVP drifting buoy data		provenance
TIB = Tiberon lab in the US		provenance
UBC = University of British Columbia		provenance
UQAR = University of Quebec at Rimouski		provenance
WHOI = Woods Hole Oceanographic Institute		provenance
WSVP = WOCE SVP data originator (AOML)		provenance
WULT = WOCE Upper Level Thermal		provenance

Attribute isa

list_values	*note*	*brick*
Bottle	Bottle	instrument
CTD	CTD	instrument
XBT	XBT	instrument

Attribute justification_code

list_values	*note*	*brick*
TBD	TBD	quality

Attribute kind

list_values	*note*	*brick*
D	dependent variable	depth_pressure, latitude longitude, variable
I	independent variable	depth_pressure, latitude longitude, variable

Attribute level

list_values	*note*	*brick*
cruise		data_set_id
profile		data_set_id
record		data_set_id
related		data_set_id
station		data_set_id

Attribute order_number

list_values	*note*	*brick*
numeric starting at 1		comment

Attribute platform_type

list_values	*note*	*brick*
drifting buoy	drifting buoy	provenance
moored buoy	Moored buoy	provenance
profiling float	profiling float	provenance
ship	Ship	provenance

Attribute property

list_values	*note*	*brick*
bottom		ldate, latitude, longitude
creation		ldate, latitude, longitude
end		ldate, latitude, longitude

start		ldate, latitude, longitude

Attribute pt_code

list_values	*note*	*brick*
see list of PCODES	see list of PCODES	data_point, depth_pressure, history, previous_value, quality, sampling, units, variable

Attribute pt_link

list_values	*note*	*brick*
numeric starting at 1	numeric starting at 1	data_point, depth_pressure, history, previous_value, quality, sampling, units, variable

Attribute received_units

list_values	*note*	*brick*
SI list	SI list	units

Attribute reliability_code

list_values	*note*	*brick*
0	unchecked	quality
1	good	quality
2	probably good	quality
3	probably bad	quality
4	bad	quality
5	changed	quality

Attribute stored_units

list_values	*note*	*brick*
SI list	SI list	units

Attribute type

list_values	*note*	*brick*
adcp		instrument
bottle		instrument
cm		instrument
CTD		instrument
float		instrument
thermistor chain		instrument
uway		instrument
XBT		instrument

Attribute typing

list_values	*note*	*brick*
C	character string	data_point, depth_pressure, latitude, local_tag, longitude, previous_value, variable
D	date	data_point, depth_pressure, latitude,

		local_tag, longitude, previous_value, variable
DT	date/time	data_point, depth_pressure, latitude, local_tag, longitude, previous_value, variable
I	integer	data_point, depth_pressure, latitude, local_tag, longitude, previous_value, variable
R	real	data_point, depth_pressure, latitude, local_tag, longitude, previous_value, variable
T	time	data_point, depth_pressure, latitude, local_tag, longitude, previous_value, variable

Attribute use_code

list_values	***note***	***brick***
TBD	TBD	quality

Annex 3: Guidelines for Populating Profile Data Structure

Table 4. This table provides a general guide to placing content into the XML profile structure.

BRICK	BRICK	EXAMPLE CONTENT
data_collection	comment	general comment applicable to the collection level
data_collection	data_dictionary	the dictionary to be used in this data collection
data_collection	provenance	provides the owner information to all data in the file
data_collection	location_set	bounding coordinates (time or space) of the collection
data_collection	data_set	
data_set	availability	identifies whether or not the cruise is available for distribution
data_set	comment	general comment applicable to the cruise level
data_set	data_point	integrated value over a cruise. Perhaps a short 3 hour cruise to replace a nearby instrument. The average of a parameter could be applicable. Cruise metadata are applicable.
data_set	data_set_id {level="cruise"}	cruise number
data_set	provenance	where the collection originated from
data_set	quality	identify the result of the test
data_set	quality_testing	describe the actual quality test
data_set	variable_set	variables that are used throughout the cruise
data_set	location_set	bounding coordinates (time or space) of the cruise
data_set	history_set	record the processing history for the data
data_set	data_set	stations within a cruise
data_set	availability	identifies whether the station is available for distribution
data_set	comment	general comment

data_set	data_point	perhaps an integrated value over the station. Eg. Mixed layer depth, e-folding depth of irradiance
data_set	data_set_id {level="station"}	station number
data_set	provenance	an individual station may have originated from a different source
data_set	quality	identify the result of the test
data_set	quality_testing	describe the actual test
data_set	variable_set	variables that are unique to the station
data_set	location_set	station position and time if only one cast
data_set	history_set	record the process the data went through
data_set	data_set	the casts that are made at one station
data_set	availability	identifies whether the cast is available for use
data_set	comment	general comment
data_set	data_point	perhaps an integrated value over the cast or metadata such as the XBT probe type
data_set	data_set_id {level="profile"}	cast number
data_set	provenance	an individual cast may have originated from a different source
data_set	quality	identify the result of the test
data_set	quality_testing	describe the actual test
data_set	variable_set	variables that are unique to the cast
data_set	location_set	cast time
data_set	history_set	record the process the data went through
data_set	data_set	the records that compose one cast
data_set	availability	identifies whether the record is available for use
data_set	comment	general comment
data_set	data_point	individual data value

data_set	data_set_id {level=”record”}	sample ID number. Note: record may be repeated more than once for the same pressure or depth.
data_set	provenance	an individual record may have originated from a different source
data_set	quality	identify the result of the test
data_set	quality_testing	describe the actual test
data_set	variable_set	variables that are unique to the record. Perhaps there is a specific sample being collected at only some depths. May also be useful on combined datasets, where data are merged at the record level.
data_set	location_set	two botle casts (deep, shallow) may have been joined with time identifiers on each bottle trip
data_set	history_set	record the process the data went through
data_set	data_set	the granular nature of data_set allows it to be specified down to a level that is sensible for the particular data type. For profile data, the necessary granularity is "record".
location_set	comment	general comment
location_set	depth_pressure	z coordinate (positive down)
location_set	latitude	position value
location_set	ldate	date and time value
location_set	longitude	position value
location_set	quality	could be a code provided with the GPS fix to determine the quality of the fix
variable_set	analysis_method	an analysis method that was used to determine the variable
variable_set	calibration	any calibration applied to the variable
variable_set	comment	general comment
variable_set	instrument	instrument used to measure the variable or operate the sensor
variable_set	sampling	a description of the method used in the sampling
variable_set	sensor	sensor used to measure the variable
variable_set	units	units for the variable

variable_set	variable	identify the variable
data_set	availability	identifies whether the related data is available for use
data_set	comment	general comment
data_set	data_point	examples are soundings at the beginning, bottom and end of a cast; anemometer reads from the bow
data_set	data_set_id {level=”related”}	perhaps the meteorological people would like some method of grouping met data related to a particular station.
data_set	provenance	a different group may be collecting the related data
data_set	quality	identify the result of the test
data_set	quality_testing	describe the actual test
data_set	variable_set	variables that are unique to the data_set
data_set	location_set	location where the unique variable was measured
data_set	history_set	record the process the data went through
data_set	data_set	

Annex 4: Schema for Profile Data Structure

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <!-- This file is dated: January 20, 2003

      This is the working draft of the schema associated with the
  Canadian XML    efforts to implement 'Keeley Bricks' into an
  XML structure.
  The schema is   divided into five basic sections:
            1) The actual top level structure
            2) All Compound bricks
            3) All Pure bricks
            4) All Attribute Groups
            5) Misc. groups

      Present outstanding issues include:
            a) typing L to indicate lat/long format to be used
            b) the local_tag is not yet defined

      Dec. 13, 2002 - Revised to set attribute occurence.
      Dec. 16, 2002 - Revised to remove typing from latitude and
          longitude, and set the same date format for all
          date elements
      Jan. 7, 2003  - Added 'name' attribute to the coefficient
          element within calibration brick.  Rearranged the
          XML types in the five groups defined above.
      Jan. 20, 2003 - Removed local_tag from schema.  Removed
          order_number attribute    from comment element.
      Feb. 10, 2003 - Corrected error.  instrument was suppose to
          be mandatory in variable_set.
      Feb. 18, 2003 - Removed pt_code from history brick and added
                  set_code.
      Feb. 26, 2003 - Removed mandatory requirement on instrument
                  brick inside variable_set.  Removed set_code
                  in history and replaced it with an optional
                  pt_code.
      Mar. 18, 2003 - Added typing categories for Date, and
Date/Time.


      Anthony W. Isenor  -->

      <xsd:element name="data_collection" type="collection"/>

      <!-- This is the top level of the brick structure for point
data.  -->

      <xsd:complexType name="collection">
            <xsd:sequence>
```

```
                  <xsd:element name="comment" type="comment_brick"
minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="data_dictionary"
type="data_dictionary_brick" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="provenance"
type="provenance_brick" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="location_set"
type="location_set_cbrick" minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="data_set" type="data_set_cbrick"
minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>

      <!-- ***** Compound bricks in this section
*************************************************** -->

      <xsd:complexType name="data_set_base">
            <xsd:sequence>
                  <xsd:element name="availability"
type="availability_brick" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="comment" type="comment_brick"
minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="data_point"
type="data_point_brick" minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="data_set_id"
type="data_set_id_brick" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="provenance"
type="provenance_brick" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="quality" type="quality_brick"
minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="quality_testing"
type="quality_testing_brick" minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="variable_set"
type="variable_set_cbrick" minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="location_set"
type="location_set_cbrick" minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="history_set"
type="history_set_cbrick" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="data_set_cbrick">
            <xsd:complexContent>
                  <xsd:extension base="data_set_base">
                        <xsd:sequence>
                              <xsd:element name="data_set"
type="data_set_cbrick" minOccurs="0" maxOccurs="unbounded"/>
                        </xsd:sequence>
                  </xsd:extension>
            </xsd:complexContent>
      </xsd:complexType>

      <xsd:complexType name="history_set_cbrick">
            <xsd:complexContent>
```

```
                  <xsd:extension base="history_set_cbrick_1">
                        <xsd:sequence>
                              <xsd:element name="history"
type="history_brick" minOccurs="0" maxOccurs="1"/>
                              <xsd:element name="previous_value"
type="previous_value_brick" minOccurs="0" maxOccurs="1"/>
                              <xsd:element name="location_set"
type="location_set_cbrick" minOccurs="0" maxOccurs="1"/>
                        </xsd:sequence>
                  </xsd:extension>
            </xsd:complexContent>
      </xsd:complexType>

      <xsd:complexType name="history_set_cbrick_1">
            <xsd:sequence>
                  <xsd:element name="comment" type="comment_brick"
minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="location_set_cbrick">
            <xsd:sequence>
                  <xsd:element name="comment" type="comment_brick"
minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="depth_pressure"
type="depth_pressure_brick" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="latitude" type="latitude_brick"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="ldate" type="ldate_brick"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="longitude"
type="longitude_brick" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="quality" type="quality_brick"
minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="variable_set_cbrick">
            <xsd:sequence>
                  <xsd:element name="analysis_method"
type="analysis_method_brick" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="calibration"
type="calibration_brick" minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="comment" type="comment_brick"
minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="instrument"
type="instrument_brick" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="sampling" type="sampling_brick"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="sensor" type="sensor_brick"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="units" type="units_brick"
minOccurs="1" maxOccurs="1"/>
```

```
                  <xsd:element name="variable" type="variable_brick"
minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>


      <!-- ***** Pure bricks in this section
**************************************************** -->

      <xsd:complexType name="analysis_method_brick">
            <xsd:sequence>
                  <xsd:element name="analysis_date"
type="date_format" minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="analysis_id" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="analyst_name" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="method" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="availability_brick">
            <xsd:annotation>
                  <xsd:documentation>The availability brick declares
the possible release of the dataset
                  in the community.</xsd:documentation>
            </xsd:annotation>
            <xsd:sequence>
                  <xsd:element name="avail_date" type="date_format"
minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attributeGroup ref="indicator_qualifiers"/>
      </xsd:complexType>

      <xsd:complexType name="calibration_brick">
            <xsd:sequence>
                  <xsd:element name="algorithm_type"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="application_date"
type="date_format" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="calibration_date"
type="date_format" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="coefficients"
type="coefficient_set" minOccurs="0" maxOccurs="unbounded"/>
                  <xsd:element name="number_of_coefficients"
type="xsd:string" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="process" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="comment_brick">
            <xsd:simpleContent>
```

```
                  <xsd:extension base="xsd:string"/>
            </xsd:simpleContent>
      </xsd:complexType>

      <xsd:complexType name="data_dictionary_brick">
            <xsd:sequence>
                  <xsd:element name="dictionary_name"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="data_point_brick">
            <xsd:simpleContent>
                  <xsd:extension base="xsd:string">
                        <xsd:attributeGroup ref="pt_qualifiers"/>
                        <xsd:attributeGroup ref="stat_qualifiers"/>
                        <xsd:attributeGroup ref="typing_qualifiers"/>
                  </xsd:extension>
            </xsd:simpleContent>
      </xsd:complexType>

      <xsd:complexType name="data_set_id_brick">
            <xsd:simpleContent>
                  <xsd:extension base="xsd:string">
                        <xsd:attributeGroup ref="level_qualifiers"/>
                  </xsd:extension>
            </xsd:simpleContent>
      </xsd:complexType>

      <xsd:complexType name="depth_pressure_brick">
            <xsd:simpleContent>
                  <xsd:extension base="xsd:string">
                        <xsd:attributeGroup ref="kind_qualifiers"/>
                        <xsd:attributeGroup ref="pt_qualifiers"/>
                        <xsd:attributeGroup ref="stat_qualifiers"/>
                        <xsd:attributeGroup ref="typing_qualifiers"/>
                  </xsd:extension>
            </xsd:simpleContent>
      </xsd:complexType>

      <xsd:complexType name="history_brick">
            <xsd:sequence>
                  <xsd:element name="application_date"
type="date_format" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="executor" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="process_identifier"
type="xsd:string" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="version" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="action" type="xsd:string"/>
            <xsd:attributeGroup ref="optional_pt_qualifiers"/>
      </xsd:complexType>
      <xsd:complexType name="instrument_brick">
            <xsd:sequence>
                  <xsd:element name="description" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="manufacturer" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="model" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="serial_number" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attributeGroup ref="type_qualifiers"/>
      </xsd:complexType>

      <xsd:complexType name="latitude_brick">
            <xsd:simpleContent>
                  <xsd:extension base="lat_restriction">
                        <xsd:attributeGroup
ref="position_qualifiers"/>
                        <xsd:attributeGroup ref="stat_qualifiers"/>
                  </xsd:extension>
            </xsd:simpleContent>
      </xsd:complexType>

      <xsd:complexType name="ldate_brick">
            <xsd:choice>
                  <xsd:group ref="date_choice"/>
                  <xsd:group ref="time_choice"/>
            </xsd:choice>
            <xsd:attributeGroup ref="position_qualifiers"/>
            <xsd:attributeGroup ref="stat_qualifiers"/>
      </xsd:complexType>

      <xsd:complexType name="longitude_brick">
            <xsd:simpleContent>
                  <xsd:extension base="long_restriction">
                        <xsd:attributeGroup
ref="position_qualifiers"/>
                        <xsd:attributeGroup ref="stat_qualifiers"/>
                  </xsd:extension>
            </xsd:simpleContent>
      </xsd:complexType>

      <xsd:complexType name="previous_value_brick">
            <xsd:simpleContent>
                  <xsd:extension base="xsd:string">
                        <xsd:attributeGroup ref="pt_qualifiers"/>
                        <xsd:attributeGroup ref="typing_qualifiers"/>
                  </xsd:extension>
            </xsd:simpleContent>
      </xsd:complexType>

      <xsd:complexType name="provenance_brick">
```

```
            <xsd:sequence>
                  <xsd:element name="agency" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="country" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="data_grouping" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="date_created" type="date_format"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="description" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="institute_code"
type="xsd:string" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="originator" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="originator_identifier"
type="xsd:string" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="platform_name" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="project" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attributeGroup ref="platform_qualifiers"/>
      </xsd:complexType>

      <xsd:complexType name="quality_brick">
            <xsd:sequence>
                  <xsd:element name="qt_date" type="date_format"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="tests_failed" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="tests_performed"
type="xsd:string" minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attribute name="justification_code"
type="xsd:string"/>
            <xsd:attributeGroup ref="pt_qualifiers"/>
            <xsd:attributeGroup ref="reliability_qualifiers"/>
            <xsd:attributeGroup ref="use_qualifiers"/>
      </xsd:complexType>

      <xsd:complexType name="quality_testing_brick">
            <xsd:sequence>
                  <xsd:element name="test_description"
type="xsd:string" minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="test_id" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="test_name" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="test_version" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>
      <xsd:complexType name="sampling_brick">
            <xsd:sequence>
                  <xsd:element name="id" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="interval" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="method" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attributeGroup ref="pt_qualifiers"/>
      </xsd:complexType>

      <xsd:complexType name="sensor_brick">
            <xsd:sequence>
                  <xsd:element name="manufacturer" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="model" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="serial_number" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="type" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:complexType>

      <xsd:complexType name="units_brick">
            <xsd:sequence>
                  <xsd:element name="conversion" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="reference" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="variable_name" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attributeGroup ref="pt_qualifiers"/>
            <xsd:attribute name="received_units" type="xsd:string"/>
            <xsd:attribute name="stored_units" type="xsd:string"
use="required"/>
      </xsd:complexType>

      <xsd:complexType name="variable_brick">
            <xsd:sequence>
                  <xsd:element name="accuracy" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="below_detection"
type="xsd:string" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="decimal_places"
type="xsd:string" minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="maximum_value" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="minimum_value" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="null_value" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
```

```
                  <xsd:element name="precision" type="xsd:string"
minOccurs="0" maxOccurs="1"/>
                  <xsd:element name="variable_name" type="xsd:string"
minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
            <xsd:attributeGroup ref="duplicate_qualifiers"/>
            <xsd:attributeGroup ref="kind_qualifiers"/>
            <xsd:attributeGroup ref="pt_qualifiers"/>
            <xsd:attributeGroup ref="typing_qualifiers_mandatory"/>
      </xsd:complexType>


      <!-- ***** Attribute Groups in this section
********************************************** -->

      <xsd:attributeGroup name="duplicate_qualifiers">
            <xsd:attribute name="duplicate_indicator">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="N"/>
                              <xsd:enumeration value="D"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="indicator_qualifiers">
            <xsd:attribute name="indicator" use="required">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <!-- Restricted -->
                              <xsd:enumeration value="R"/>
                              <!-- Open -->
                              <xsd:enumeration value="O"/>
                              <!-- Consult -->
                              <xsd:enumeration value="C"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="kind_qualifiers">
            <xsd:attribute name="kind">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="I"/>
                              <!--Independent-->
                              <xsd:enumeration value="D"/>
                              <!--Dependent-->
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>
      <xsd:attributeGroup name="level_qualifiers">
            <xsd:attribute name="level" use="required">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="cruise"/>
                              <xsd:enumeration value="station"/>
                              <xsd:enumeration value="profile"/>
                              <xsd:enumeration value="record"/>
                              <xsd:enumeration value="related"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="platform_qualifiers">
            <xsd:attribute name="platform_type">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="profiling
float"/>
                              <xsd:enumeration value="ship"/>
                              <xsd:enumeration value="moored buoy"/>
                              <xsd:enumeration value="drifting
buoy"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="position_qualifiers">
            <xsd:attributeGroup ref="kind_qualifiers"/>
            <xsd:attribute name="property">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="start"/>
                              <xsd:enumeration value="bottom"/>
                              <xsd:enumeration value="end"/>
                              <xsd:enumeration value="creation"/>
                              <xsd:enumeration value="original"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="pt_qualifiers">
            <xsd:attribute name="pt_code" type="xsd:string"
use="required"/>
            <xsd:attribute name="pt_link" type="xsd:string"/>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="reliability_qualifiers">
            <xsd:attribute name="reliability_code">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:unsignedShort">
```

```
                              <xsd:enumeration value="0"/>
                              <xsd:enumeration value="1"/>
                              <xsd:enumeration value="2"/>
                              <xsd:enumeration value="3"/>
                              <xsd:enumeration value="4"/>
                              <xsd:enumeration value="5"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="optional_pt_qualifiers">
            <xsd:attribute name="pt_code" type="xsd:string"/>
            <xsd:attribute name="pt_link" type="xsd:string"/>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="stat_qualifiers">
            <xsd:attribute name="statistic" type="xsd:string"/>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="type_qualifiers">
            <xsd:attribute name="type" use="required">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="adcp"/>
                              <xsd:enumeration value="bottle"/>
                              <xsd:enumeration value="cm"/>
                              <xsd:enumeration value="CTD"/>
                              <xsd:enumeration value="dbt"/>
                              <xsd:enumeration value="float"/>
                              <xsd:enumeration value="model"/>
                              <xsd:enumeration value="radar"/>
                              <xsd:enumeration value="staff"/>
                              <xsd:enumeration value="staff_gauge"/>
                              <xsd:enumeration value="sounder"/>
                              <xsd:enumeration value="thermistor"/>
                              <xsd:enumeration value="uway"/>
                              <xsd:enumeration value="unknown"/>
                              <xsd:enumeration
value="water_level_gauge"/>
                              <xsd:enumeration value="wave_buoy"/>
                              <xsd:enumeration
value="wave_directional_buoy"/>
                              <xsd:enumeration
value="wave_pressure_gauge"/>
                              <xsd:enumeration
value="wave_recorder"/>
                              <xsd:enumeration value="XBT"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="typing_qualifiers">
```

```
            <xsd:attribute name="typing">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <!--                    D - Double
                              T - Time
                              D - Date
                              DT - Date and time
                              R - Number with a decimal
                              I - Integer
                              C - Character-->
                              <xsd:enumeration value="T"/>
                              <xsd:enumeration value="D"/>
                              <xsd:enumeration value="DT"/>
                              <xsd:enumeration value="R"/>
                              <xsd:enumeration value="I"/>
                              <xsd:enumeration value="C"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="typing_qualifiers_mandatory">
            <xsd:attribute name="typing" use="required">
                  <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                              <!--                    D - Double
                              T - Time
                              D - Date
                              DT - Date and time
                              R - Number with a decimal
                              I - Integer
                              C - Character-->
                              <xsd:enumeration value="T"/>
                              <xsd:enumeration value="D"/>
                              <xsd:enumeration value="DT"/>
                              <xsd:enumeration value="R"/>
                              <xsd:enumeration value="I"/>
                              <xsd:enumeration value="C"/>
                        </xsd:restriction>
                  </xsd:simpleType>
            </xsd:attribute>
      </xsd:attributeGroup>

      <xsd:attributeGroup name="use_qualifiers">
            <xsd:attribute name="use_code" type="xsd:string"/>
      </xsd:attributeGroup>


      <!-- ***** Mics. groups in this section
********************************************** -->

      <xsd:complexType name="coefficient_set">
            <xsd:simpleContent>
                  <xsd:extension base="xsd:string">
```

```
                        <xsd:attribute name="name"
type="xsd:string"/>
                  </xsd:extension>
            </xsd:simpleContent>
      </xsd:complexType>

      <xsd:group name="date_choice">
            <xsd:sequence>
                  <xsd:element name="pdate" type="date_format"
minOccurs="1" maxOccurs="1"/>
                  <xsd:element name="ptime" type="time_restriction"
minOccurs="0" maxOccurs="1"/>
            </xsd:sequence>
      </xsd:group>

      <xsd:complexType name="date_format">
            <xsd:simpleContent>
                  <xsd:extension base="date_restriction"/>
            </xsd:simpleContent>
      </xsd:complexType>

      <xsd:simpleType name="lat_restriction">
            <xsd:restriction base="xsd:decimal">
                  <xsd:minInclusive value="-90.0"/>
                  <xsd:maxInclusive value="90.0"/>
            </xsd:restriction>
      </xsd:simpleType>

      <xsd:simpleType name="long_restriction">
            <xsd:restriction base="xsd:decimal">
                  <xsd:minInclusive value="-180.0"/>
                  <xsd:maxInclusive value="180.0"/>
            </xsd:restriction>
      </xsd:simpleType>

      <xsd:group name="time_choice">
            <xsd:sequence>
                  <xsd:element name="ptime" type="time_restriction"/>
            </xsd:sequence>
      </xsd:group>

      <!--Note:  This restriction is required because I have
  discovered that the validator I am using does not correctly
  implement the date or time xsd datatypes.  The following
  restrictions help ensure the proper checking of the date and
  time datatypes.  Note that the restrictions are in addition
  to the datatype defined by date and time, and so do not
  restrict the exact form of the date or time.  (Example:
  The pattern for hours implies that 88 is a valid value.
  Hovever, the time type properly restricts the values to 23
  or less.

  Note also that the restrictions force Zulu time to be
  specified using the capital Z character.  Also, no time
  zome specification is allowed.

A.W.Isenor (Dec. 2002)-->
      <xsd:simpleType name="date_restriction">
            <xsd:restriction base="xsd:date">
                  <xsd:pattern value="([0-9]{4}-[0-9]{2}-[0-
9]{2}Z)"/>
            </xsd:restriction>
      </xsd:simpleType>

      <xsd:simpleType name="time_restriction">
            <xsd:restriction base="xsd:time">
                  <xsd:pattern value="([0-9]{2}):([0-9]{2}):(([0-
9]{2})|([0-9]{2})\.[0-9]*)Z"/>
            </xsd:restriction>
      </xsd:simpleType>
</xsd:schema>
```

Annex 5: MEDS to XML Data Mapping

Table 5. MEDS to XML data mapping.

MEDS	DATA_SET LEVEL	XPATH	ATTRIBUTES	*
Stn/Fxd/Mkey				10
Stn/Fxd/One_Deg_Sq				10
Stn/Fxd/Cr_Number	Cruise	data_set_id		
Stn/Fxd/Obs_year,month,day,time	Station	location_set/ldate	property=”start”	
Stn/Fxd/Data_Type	Station	provenance/description		1
Stn/Fxd/Iumsgno				10
Stn/Fxd/Stream_Source				10
Stn/Fxd/U_Flag				10
Stn/Fxd/Stn_Number	Profile	data_set_id	level=”profile”	3
Stn/Fxd/Latitude	Station	location_set/latitude		
Stn/Fxd/Longitude	Station	location_set/longitude		2
Stn/Fxd/Q_Pos, Q_Date_Time	Station	location_set/quality	pt_code	6
Stn/Fxd/Q_Record				7
Stn/Fxd/Up_Date				7
Stn/Fxd/Bul_Time	Station	provenance/date_created		8
Stn/Fxd/Bul_Header	Station	provenance/data_grouping		8
Stn/Fxd/Source_Id	Station	provenance/originator or /agency		11
Stn/Fxd/Stream_Ident				10
Stn/Fxd/QC_Version				7
Stn/Fxd/Avail	Station	availability	indicator	
Stn/Fxd/No_Prof				10
Stn/Fxd/Nparms				10

Stn/Fxd/Sparms				10
Stn/Fxd/Num_Hists				10
Stn/Prof(I)/No_Seg				10
Stn/Prof(I)/Prof_Type	Profile			9
Stn/Prof(I)/Dup_Flag	Profile	variable_set/variable	duplicate_indicator	
Stn/Prof(I)/Digit_Code	Profile	variable_set/comment		12
Stn/Prof(I)/Standard	Profile	variable(I)/decimal_places		
Stn/Prof(I)/Deepest_Depth				7
Stn/Surface(i),i=1,nparms/Parm	Station	data_point(i)	pt_code, pt_link, typing='C'	
Stn/Surface(i),i=1,nparms/Q_Parm	Station	data_set/quality(i)	pt_code	
Stn/Surf_Codes(j),i=1,sparms/Cparm	Station	data_point(nparms+j)	pt_code, pt_link, typing='C'	
Stn/Surf_Codes(j),i=1,sparms/Q_Parm	Station	data_set/quality(i) (nparms+j)	pt_code	
Stn/History(i)/Ident_Code	Station	history_set(i)/history/ executor		
Stn/History(i)/PRC_Code	Station	history_set(i)/history/ process_identifier		
Stn/History(i)/Version	Station	history_set(i)/history/ version		
Stn/History(i)/PRC_Date	Station	history_set(i)/history/ application_date		
Stn/History(i)/Act_Code	Station	history_set(i)/history	action	
Stn/History(i)/Act_Parm	Station	history_set(i)/history	pt_code, pt_link	
Stn/History(i)/O_Value	Station	history_set(i)/history/previous_value	pt_code, pt_link, typing	
Profile(i)/Fxd/Prof_Type	Profile	variable_set(i+1)/ variable	pt_code, pt_link, typing='D'	

Profile(i)/Fxd/Profile_Seg				10
Profile(i)/Fxd/No_Depths				10
Profile(i)/Fxd/D_P_Code	Profile	variable_set(1)/variable	pt_code, pt_link, kind="I"	4
Profile(i)/Prof(j)/Depth_Press	Record(j)	location_set/ depth_pressure	pt_code	
Profile(i)/Prof(j)/DP_Flag	Record(j)	quality	pt_code reliability_code	5
Profile(i)/Prof(j)/Parm	Record(j)	data_point(I)	pt_code, pt_link, typing	
Profile(i)/Prof(j)/Q_Parm	Record(j)	quality	pt_code reliability_code	5

1. Put the code into the XML element. Consider putting the looked-up code meaning into a comment. The Data_Type codes on each profile are always the same as the Stn/Fxd/Data_Type.

2. MEDS Longitudes are real numbers, positive for West, range +/- 180 degrees.

3. if there are multiple profiles (eg non-identical depth vectors), then each XML profile will have the same data_set_id.

4. pt_code will be DEPH or PRES depending upon value of D_P_Code.

5. Each record data_set could have a quality element for depth/pressure and for each dependent variable value. If the MEDS quality flags are all the same for the whole profile, for either the depth/pressure or for all the variables in the profile, the quality elements will be moved up to the profile level.

6. The present location_set allows for a single quality flag that applies to the entire brick. In MEDS archive there exists a quality flag for position (latitude and longitude) and one for date (date and time). We choose at this time to take the worst flag of Q_Pos and Q_Date_Time to be written to the quality brick. Other strategies that preserve both flags are possible.

7. This variable is a summary of more detailed information also included in the file. Because it represents a summary, and is derivable from other information in the file, it has not been carried into the xml file.

8. When values for Bul_Time and Bul_Header exist, the data have come from the GTS and this is the way that MEDS records the originator of the data. Because Agency is mandatory in the provenance brick, “GTS” would be used for the agency.

9. This field is used to identify the correct variable to which subsequent information in this structure applies. It matches a pt_code written to variable_set/variable.

10. This information is of internal use to MEDS only and so is not carried into the xml file.

11. This is always placed into the xml brick. When the data come from the GTS (indicated by the Data_Type), the contents of this field is placed in the originator element of the provenance brick. When the data are not from the GTS, the information should be placed in the agency element of the provenance brick.

12. This information can be placed in a comment as indicated or better in the sampling brick. Since the sampling brick has not be implemented in this project, it has been mapped to a comment field.

Annex 6: Example Profile Data in XML Structure

```
<?xml version="1.0"?>
<!--XML created by program MEDS2XML version 1.0 2003-04-22 15:58:42--
>
<!--    using MSXML4.DLL-->
<data_collection xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="c:\Anthony\Projects\xml\odf_conversion
\bricks_v2.xsd">
  <comment>Created from MEDS file: 18ne01026.txt</comment>
  <provenance>
    <agency>Marine Environmental Data Service</agency>
    <country>Canada</country>
    <date_created>2003-04-22Z</date_created>
  </provenance>
  <data_set>
    <!--Cruise 18NE01026-->
    <data_set_id level="cruise">18NE01026</data_set_id>
    <data_set>
      <!--Station 1-->
      <availability indicator="O">
        <avail_date>2002-02-25Z</avail_date>
      </availability>
      <data_point pt_code="ACC$" typing="C">A200204545</data_point>
      <data_point pt_code="CSN$" typing="C">CG2683</data_point>
      <data_point pt_code="PNM1" typing="C">A.Needler</data_point>
      <data_point pt_code="QCP$" typing="C">41E1AFCE</data_point>
      <data_point pt_code="QCF$" typing="C">00000000</data_point>
      <data_set_id level="station"/>
      <provenance>
        <agency>BIO</agency>
        <country>CANADA</country>
        <data_grouping>Bottle</data_grouping>
        <date_created>2002-02-25Z</date_created>
        <description>Bottle</description>
        <originator>D.Swain</originator>
        <originator_identifier>NED2001026 4</originator_identifier>
        <platform_name>18NE</platform_name>
        <project>Pelagic</project>
      </provenance>
      <variable_set>
        <instrument type="CTD"/>
        <units pt_code="ACC$" stored_units="text"/>
        <variable pt_code="ACC$" kind="I" typing="C">
          <variable_name>MEDS accesion number</variable_name>
        </variable>
      </variable_set>
      <variable_set>
        <instrument type="CTD"/>
        <units pt_code="CSN$" stored_units="text"/>
        <variable pt_code="CSN$" kind="I" typing="C">
```

```
          <variable_name>IGOSS call sign</variable_name>
        </variable>
      </variable_set>
      <variable_set>
        <instrument type="CTD"/>
        <units pt_code="PNM1" stored_units="text"/>
        <variable pt_code="PNM1" kind="I" typing="C">
          <variable_name>Platform name</variable_name>
        </variable>
      </variable_set>
      <variable_set>
        <instrument type="CTD"/>
        <units pt_code="QCP$" stored_units="text"/>
        <variable pt_code="QCP$" kind="I" typing="C">
          <variable_name>The indicator encoding which MEDS QC tests
have been executed for T and S data.</variable_name>
        </variable>
      </variable_set>
      <variable_set>
        <instrument type="CTD"/>
        <units pt_code="QCF$" stored_units="text"/>
        <variable pt_code="QCF$" kind="I" typing="C">
          <variable_name>The indicator encoding which MEDS QC tests
have failed for T and S data.</variable_name>
        </variable>
      </variable_set>
      <location_set>
        <latitude>48.1773</latitude>
        <ldate property="start">
          <pdate>2001-06-19Z</pdate>
          <ptime>08:55:00Z</ptime>
        </ldate>
        <longitude>-64.1744</longitude>
        <quality pt_code="GGQF" reliability_code="1"/>
      </location_set>
      <history_set>
        <history pt_code="RCRD" action="CR">
          <application_date>2002-02-21Z</application_date>
          <executor>ME</executor>
          <process_identifier>RFMT</process_identifier>
          <version>1.0</version>
        </history>
      </history_set>
      <history_set>
        <history pt_code="RCRD" action="QC">
          <application_date>2002-02-25Z</application_date>
          <executor>ME</executor>
          <process_identifier>QCAD</process_identifier>
          <version>1.0</version>
        </history>
      </history_set>
      <history_set>
        <history pt_code="RCRD" action="QC">
          <application_date>2002-02-25Z</application_date>
          <executor>ME</executor>
          <process_identifier>IG05</process_identifier>
          <version>2.0</version>
        </history>
      </history_set>
      <history_set>
        <history pt_code="RCRD" action="DC">
          <application_date>2002-02-25Z</application_date>
          <executor>ME</executor>
          <process_identifier>IG03</process_identifier>
          <version>1.4</version>
        </history>
      </history_set>
      <history_set>
        <history pt_code="RCRD" action="UP">
          <application_date>2002-02-25Z</application_date>
          <executor>ME</executor>
          <process_identifier>OCUP</process_identifier>
          <version>1.0</version>
        </history>
      </history_set>
      <data_set>
        <!--Profile-->
        <data_set_id level="profile">1</data_set_id>
        <quality pt_code="PRES" reliability_code="1"/>
        <variable_set>
          <instrument type="CTD"/>
          <units pt_code="PRES" stored_units="dbars"/>
          <variable pt_code="PRES" kind="I" typing="R">
            <variable_name>Sea pressure (sea surface =
0)</variable_name>
          </variable>
        </variable_set>
        <variable_set>
          <instrument type="CTD"/>
          <units pt_code="NTRA" stored_units="mmol/m**3"/>
          <variable pt_code="NTRA" kind="D" typing="R">
            <variable_name>Nitrate (NO3-N) CONTENT</variable_name>
          </variable>
        </variable_set>
        <variable_set>
          <instrument type="CTD"/>
          <units pt_code="OSI$" stored_units="text"/>
          <variable pt_code="OSI$" kind="D" typing="R">
            <variable_name>Originator&apos;s sample identifier
      General purpose   4/22/2002</variable_name>
          </variable>
        </variable_set>
        <variable_set>
          <instrument type="CTD"/>
          <units pt_code="PHOS" stored_units="mmol/m**3"/>
          <variable pt_code="PHOS" kind="D" typing="R">
            <variable_name>Phosphate (PO4-P) content</variable_name>
          </variable>
        </variable_set>
        <variable_set>
          <instrument type="CTD"/>
          <units pt_code="SLCA" stored_units="mmol/m**3"/>
          <variable pt_code="SLCA" kind="D" typing="R">
            <variable_name>Silicate (SiO4-Si) content</variable_name>
          </variable>
        </variable_set>
        <data_set>
          <!--Record 1-->
          <data_point pt_code="NTRA">0.55</data_point>
          <data_point pt_code="OSI$">237906</data_point>
          <data_point pt_code="PHOS">0.33</data_point>
          <data_point pt_code="SLCA">0.77</data_point>
          <data_set_id level="record"/>
          <quality pt_code="NTRA" reliability_code="1"/>
          <quality pt_code="OSI$" reliability_code="0"/>
          <quality pt_code="PHOS" reliability_code="1"/>
          <quality pt_code="SLCA" reliability_code="1"/>
          <location_set>
            <depth_pressure pt_code="PRES">5</depth_pressure>
          </location_set>
        </data_set>
        <data_set>
          <!--Record 2-->
          <data_point pt_code="NTRA">5.23</data_point>
          <data_point pt_code="OSI$">237905</data_point>
          <data_point pt_code="PHOS">1.16</data_point>
          <data_point pt_code="SLCA">5.72</data_point>
          <data_set_id level="record"/>
          <quality pt_code="NTRA" reliability_code="1"/>
          <quality pt_code="OSI$" reliability_code="0"/>
          <quality pt_code="PHOS" reliability_code="1"/>
          <quality pt_code="SLCA" reliability_code="1"/>
          <location_set>
            <depth_pressure pt_code="PRES">43</depth_pressure>
          </location_set>
        </data_set>
      </data_set>
    </data_set>
  </data_set>
</data_collection>
```

Annex 7: IOS to XML Data Mapping

Table 6. IOS to XML data mapping.

SECTION	ITEM OR TABLE/COLUMN	DATASET LEVEL	XPATH	ATTRBUTES	*
File	Start Time	Station	location_set/ldate	property=”start”	1
	End Time	Station	location_set/ldate	property=”end”	1
	Data Description	Station	provenance/description		
	Channel(i)/Name	Profile	variable_set(i)/variable	pt_code, pt_link, kind	2
	Channel(i)/Name	Profile	variable_set(i)/variable/ variable_name		
	Channel(i)/Units	Profile	variable_set(i)/units	stored_units	
	Channel(i)/Minimum	Profile	variable_set(i)/variable/ minimum_value		
	Channel(i)/Maximum	Profile	variable_set(i)/variable/ maximum_value		
	Channel(i)/Pad	Profile	variable_set(i)/variable/ null_value		
	Channel(i)/Type	Profile	variable_set(i)/variable	typing	
	Channel(i)/Decimal_places	Profile	variable_set(i)/variable/ decimal_places		
Admin	Mission	Cruise	data_set_id	level=”cruise”	
	Agency	Cruise	provenance/ institute_code		
	Country	Cruise	provenance/ country		
	Project	Station	provenance/ project		3
	Scientist	Station	provenance/ originator		3
	Platform	Station	provenance/ platform_name		3
Location	Geographic Area	Cruise	Cruise		10
	Station	Station	Station	level ="station"	

	Event Number	Profile	Profile	level="profile"	
	Latitude	Station	Station		
	Longitude	Station	Station		9
	Latitude 2	Station	Station		
	Longitude 2	Station	Station		9
	Water Depth	Station	Station	pt_code	12
	Ice Thickness	Station	Station	pt_code	12
	Magnetic Declination	Station	Station	pt_code	12
Instrument	Type	Station	variable_set(*)/ instrument/ description		4
	Model	Station	variable_set(*)/ instrument/model		4
	Serial Number	Station	variable_set(*)/ serial_number		4
	Sensors(k)/SerialNo	Station	variable_set(*)/ sensor/ serial_number		4
History	Programs(i)/Name	Station	history_set(i)/ history/ process_identifier		
	Programs(i)/Version	Station	history_set(i)/ history/ version		
	Programs(i)/ Date+Time	Station	history_set(i)/ history/ application_date		
	Remarks(i)	Station	history_set/ comment(i)		
Calibration	* Channels(k)/ Formula	Station	variable_set(i)/ calibration/ algorithm_type		6,7
	* Channels(k) Coefficients	Station	variable_set(i)/ calibration/ coefficient_set		
	* Channels(k) (count them!)	Station	variable_set(i)/ calibration/ number_of_coefficients		
	"CALIB"	Station	variable_set(i)/ calibration/process		

	date and time from first CALIB entry in HISTORY	Station	variable_set(i)/ calibration/ application date		8
(other)	source file name	Station	provenance/ originator_identifier		
	header time stamp	Station	provenance/ date_created		
	"Institute of Ocean Sciences"	Collection	provenance/agency		
(Data)	sample_number	Record(j)	data_set_id	level="record"	11
	pressure	Record(j)	location_set/ depth_pressure	pt_code= "Pressure"	
	channel(i)	Record(j)	data_point(i)	pt_code, pt_link	5

1. IOS Full Times will be converted to UTC if necessary before transforming into ldate bricks.

2. The channel name will be copied as the pt_code. pt_link will not be inserted unless there are duplicate pt_codes. If there are duplicate pt_codes, the pt_link of the first one will be set to 1, the second one to 2, etc. The kind attribute will be set to "I" (independent) for pressure, otherwise "D"

3. These elements are probably (but not necessarily) going to be the same for all stations. It will help keep the conversion program simple to store them at the station level, even though duplication could be avoided in many cases by putting them at the cruise level.

4. IOS headers allow for just one instrument. The instrument information in XML is part of the variable_set brick. So the instrument information will be duplicated in each variable_set. Search the Instrument/Sensor table for a Name which is a substring of the variable name. If a match is found, copy the serial number to XML.

5. All channels other than sample_number and depth/pressure. The pt_code (and possibly the pt_link) attribute will link the data to a variable_set element. Kind and Typing information will be stored at the variable_set level.

6. Channel name (k) in tables in the IOS Header Calibration section will have to be matched to the File section Channel name (i) by channel name. Process the three calibration tables RAW, CALCULATED and CORRECTED in that order. If the same channel name re-occurs, create an additional calibration brick for that variable.

7. The <algorithm_type> will be set to "IOS Formula n" where n is the IOS formula number.

8. If there is only one CALIB entry in the HISTORY table, use the run date for the <application_date> element throughout. If there is more than 1 CALIB entry, use that run date for calibration bricks created from the RAW Calibration table, but omit all other application_date elements, because it will be difficult to determine in software which calibration was done in which CALIB run.

9. Longitudes in XML are real numbers, negative for West. In the IOS Header, longitudes are coded in degrees, minutes and hemisphere format.

10. If the Geographic Area information is not the same for all files, the comment could go at the Station level instead.

11. If the file has a "sample_number" channel, that data will go in the text of each record-level data_set_id element, instead of into data_point elements.

12. The IOS item name will be copied as the pt_code attribute value.

Annex 8: ODF to XML Data Mapping

Table 7. BIO ODF to XML data mapping.

ODF ELEMENT	ODF SUBELEMENT	
Cruise_Header	Country_Institute_Code	data_collection\data_set[level=cruise]\provenance\institute_code
Cruise_Header	Cruise_Number	data_collection\data_set[level=cruise]\data_set_id
Cruise_Header	Organization	data_collection\data_set[level=cruise]\provenance\agency
Cruise_Header	Chief_Scientist	data_collection\data_set[level=cruise]\provenance\originator
Cruise_Header	Start_Date	data_collection\data_set[level=cruise]\location_set\ldate{BE}
Cruise_Header	End_date	data_collection\data_set[level=cruise]\location_set\ldate{EN}
Cruise_Header	Platform	data_collection\data_set[level=cruise]\provenance\platform_name
Cruise_Header	Cruise_Name	data_collection\data_set[level=cruise]\provenance\project
Cruise_Header	Cruise_Description	data_collection\data_set[level=cruise]\provenance\description
Event_Header	Data_Type	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\provenance\data_grouping
Event_Header	Event_Number	data_collection\data_set[level=cruise]\data_set[level=station]\data_set_id
Event_Header	Event_Qualifier1	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\data_set_id
Event_Header	Event_Qualifier2	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\provenance\description
Event_Header	Creation_Date	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\location_set\ldate{CR}
Event_Header	Orig_Creation_Date	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\location_setldate{OC}
Event_Header	Start_Date_Time	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\location_set\ldate{BE}
Event_Header	End_Date_Time	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\location_set\ldate{EN}

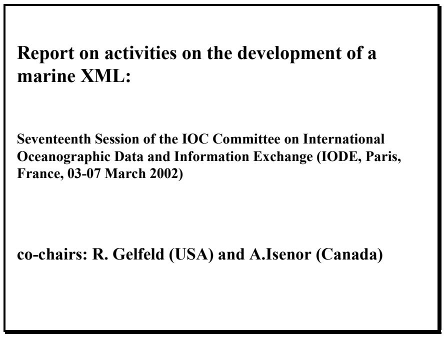
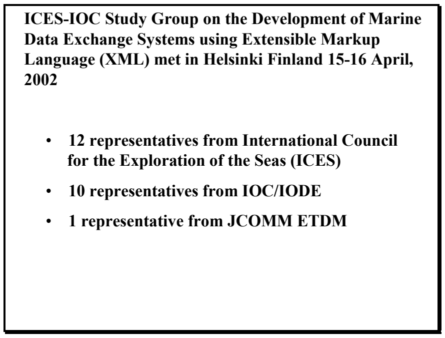
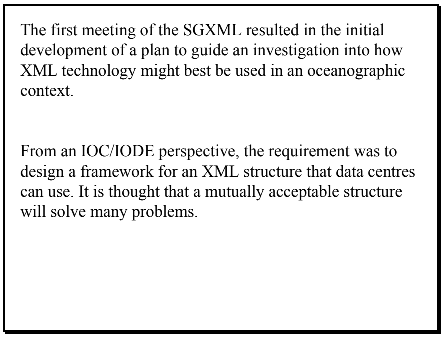
Event_Header	Initial_Latitude	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\location_set\latitude{BE}
Event_Header	Initial_Longitude	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\location_set\longitude{BE}
Event_Header	End_Latitude	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\location_set\latitude{EN}
Event_Header	End_Longitude	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\location_set\longitude{EN}
Event_Header	Min_Depth	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\data_set[level=related]\data_point
Event_Header	Max_Depth	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\data_set[level=related]\data_point
Event_Header	Sampling_Interval	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\variable_set\sampling\sampling_interval
Event_Header	Sounding	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\data_set[level=related]\data_point
Event_Header	Depth_Off_Bottom	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\data_set[level=related]\data_point
Event_Header	Event_Comments	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\comment
Instrument_Header	Inst_Type	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\variable_set\instrument\manufacturer
Instrument_Header	Model	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\variable_set\instrument\model
Instrument_Header	Serial_Number	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\variable_set\instrument\serial_number
Instrument_Header	Description	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\variable_set\instrument\description
Polynominal_Cal_Header	Parameter_Name	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\variable_set\variable\variable_name
Polynominal_Cal_Header	Calibration_date	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\variable_set\calibration\calibration_date
Polynominal_Cal_Header	Application_date	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\variable_set\calibration\application_date
Polynominal_Cal_Header	Number_Coefficients	data_collection\data_set[level=cruise]\data_set[level=station]\data_set[level=profile]\variable_set\calibration\number_of_coefficients

Polynominal_Cal_Header	Coefficients	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\calibration\coefficients
Compass_Cal_Header	Parameter_Name	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\variable\variable_name
Compass_Cal_Header	Calibration_Date	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\calibration\ calibration_date
Compass_Cal_Header	Application_Date	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\calibration\ application_date
Compass_Cal_Header	Directions	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\calibration\coefficients
History_Header	Creation_Date	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\history_set\comment
History_Header	Process	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\history_set\comment
Parameter_Header	Type	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\variable {typing}
Parameter_Header	Name	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\variable\variable_name
Parameter_Header	Units	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\variable\units {stored_units}
Parameter_Header	Code	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\variable {pt_code}
Parameter_Header	Null_Value	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\variable\null_value
Parameter_Header	Print_Field_Width	NO MAPPING
Parameter_Header	Print_Decimal_Places	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\variable\decimal_places
Parameter_Header	Angle_Of_Section	NO MAPPING
Parameter_Header	Magnetic_Variation	NO MAPPING
Parameter_Header	Depth	NO MAPPING
Parameter_Header	Minimum_Value	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\variable\maximum_value
Parameter_Header	Maximum_Value	data_collection\data_set[level=cruise]\data_set[level=station]\ data_set[level=profile]\variable_set\variable\minimum_value
Parameter_Header	Number_Valid	NO MAPPING

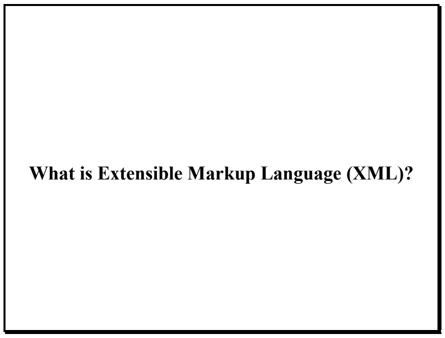
Parameter_Header	Number_Null	NO MAPPING
Record_Header	Num_Calibration	NO MAPPING
Record_Header	Num_Swing	NO MAPPING
Record_Header	Num_History	NO MAPPING
Record_Header	Num_Cycle	NO MAPPING
Record_Header	Num_Param	NO MAPPING

Annex 9: Presentation to IODE

Table 8. Slide presentation made to IODE by Bob Gelfeld, Associate Director, World Data Centre – A Oceanography.

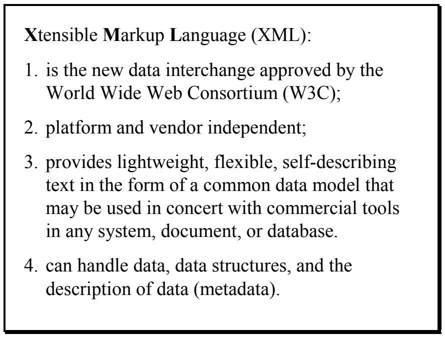
SLIDE NUMBER	SLIDE
1	 **Report on activities on the development of a marine XML:** **Seventeenth Session of the IOC Committee on International Oceanographic Data and Information Exchange (IODE, Paris, France, 03-07 March 2002)** **co-chairs: R. Gelfeld (USA) and A.Isenor (Canada)**
2	 **ICES-IOC Study Group on the Development of Marine Data Exchange Systems using Extensible Markup Language (XML) met in Helsinki Finland 15-16 April, 2002** • **12 representatives from International Council for the Exploration of the Seas (ICES)** • **10 representatives from IOC/IODE** • **1 representative from JCOMM ETDM**
3	 The first meeting of the SGXML resulted in the initial development of a plan to guide an investigation into how XML technology might best be used in an oceanographic context. From an IOC/IODE perspective, the requirement was to design a framework for an XML structure that data centres can use. It is thought that a mutually acceptable structure will solve many problems.

4



What is Extensible Markup Language (XML)?

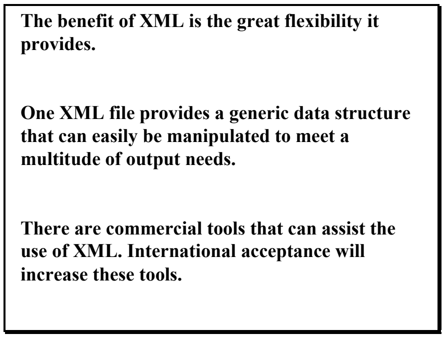
5



Xtensible **M**arkup **L**anguage (XML):

1. is the new data interchange approved by the World Wide Web Consortium (W3C);
2. platform and vendor independent;
3. provides lightweight, flexible, self-describing text in the form of a common data model that may be used in concert with commercial tools in any system, document, or database.
4. can handle data, data structures, and the description of data (metadata).

6

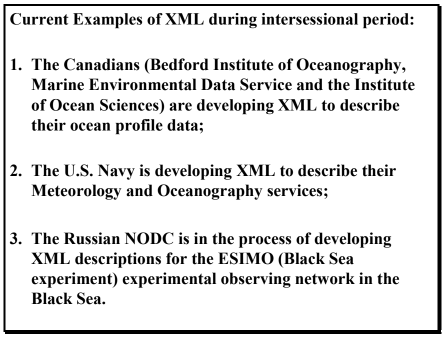


The benefit of XML is the great flexibility it provides.

One XML file provides a generic data structure that can easily be manipulated to meet a multitude of output needs.

There are commercial tools that can assist the use of XML. International acceptance will increase these tools.

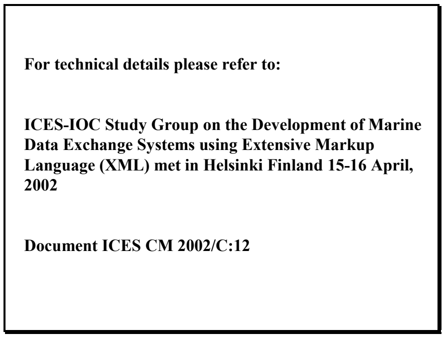
7



Current Examples of XML during intersessional period:

1. **The Canadians (Bedford Institute of Oceanography, Marine Environmental Data Service and the Institute of Ocean Sciences) are developing XML to describe their ocean profile data;**
2. **The U.S. Navy is developing XML to describe their Meteorology and Oceanography services;**
3. **The Russian NODC is in the process of developing XML descriptions for the ESIMO (Black Sea experiment) experimental observing network in the Black Sea.**

8

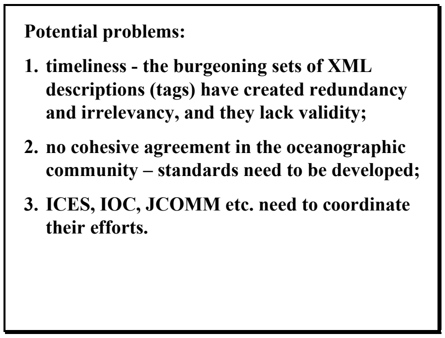


For technical details please refer to:

ICES-IOC Study Group on the Development of Marine Data Exchange Systems using Extensive Markup Language (XML) met in Helsinki Finland 15-16 April, 2002

Document ICES CM 2002/C:12

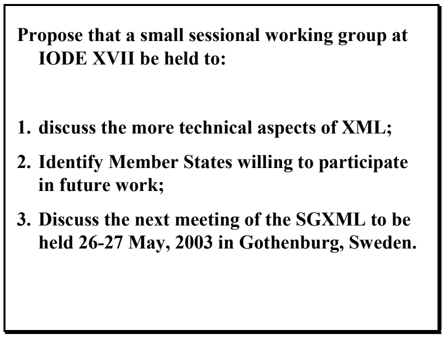
9



Potential problems:

1. **timeliness - the burgeoning sets of XML descriptions (tags) have created redundancy and irrelevancy, and they lack validity;**
2. **no cohesive agreement in the oceanographic community – standards need to be developed;**
3. **ICES, IOC, JCOMM etc. need to coordinate their efforts.**

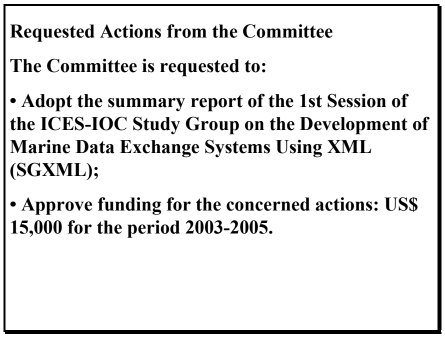
10



Propose that a small sessional working group at IODE XVII be held to:

1. **discuss the more technical aspects of XML;**
2. **Identify Member States willing to participate in future work;**
3. **Discuss the next meeting of the SGXML to be held 26-27 May, 2003 in Gothenburg, Sweden.**

11



Requested Actions from the Committee

The Committee is requested to:

- **Adopt the summary report of the 1st Session of the ICES-IOC Study Group on the Development of Marine Data Exchange Systems Using XML (SGXML);**
- **Approve funding for the concerned actions: US$ 15,000 for the period 2003-2005.**

List of symbols/abbreviations/acronyms/initialisms

API	Application Program(ming) Interface
BIO	Bedford Institute of Oceanography
COM	Common Object Model
CTD	Conductivity-Temperature-Depth Probe
CVF	Compaq Visual Fortran
DFO	Fisheries and Oceans
DND	Department of National Defence
DOM	Document Object Model
DRDC	Defence R&D Canada
DTD	Document Type Definition
EU	European Union
ICES	International Council for the Exploration of the Sea
IOC	Intergovernmental Oceanographic Commission
IODE	Intergovernmental Oceanographic Data and Information Exchange Committee
IOS	Institute of Ocean Sciences
JAXB	Java Architecture for XML Binding
JCOMM	Joint WMO-IOC Technical Commission for Oceanography and Marine Meteorology
MEDS	Marine Environmental Data Service
ODF	Ocean Data Format
ODS	Oceans Data System
OODB	Object Oriented Database

OSD	Ocean Sciences Division
SAX	Simple API for XML
SGXML	ICES-IOC Study Group on the Development of Marine Data Exchange Systems Using XML
SSF	Science Strategic Funds
W3C	World Wide Web Consortium
WMO	World Meteorological Organization
XBT	Expendable Bathythermograph
XML	eXtensible Markup Language
XSD	XML Schema Definition
XSLT	eXtensible Stylesheet Language Transformations

Distribution list

LIST PART 1:

2 DRDC ATLANTIC LIBRARY FILE COPIES
3 DRDC ATLANTIC LIBRARY (SPARES)
2 AUTHOR

7 TOTAL LIST PART 1

LIST PART 2: DISTRIBUTED BY DRDKIM 3

1 NDHQ/ ADMS&T/ DRDKIM 3

1 Doug Bancroft
Dir, Oceanography and Climate Branch
Department of Fisheries and Oceans
Station 12W130
200 Kent Street
Ottawa K1A 0E6

1 Dr. Harry Dooley
International Council for the Exploration of the Sea (ICES)
2-4 Palægade
DK-1261
Copenhagen K
Denmark

1 Steve Foreman,
Met Office
London Road
Bracknell, Berkshire RG12 2SZ
United Kingdom

1 Doug Gregory
Ocean Sciences Division
Bedford Institute of Oceanography
PO Box 1006
Dartmouth, N.S.
B2Y 4A2

2 J. Robert Keeley
Marine Environmental Data Service
Department of Fisheries and Oceans
W12082 - 200 Kent Street
Ottawa, Ontario Canada
K1A 0E6

1 Dr. John Loder
Ocean Sciences Division
Bedford Institute of Oceanography
PO Box 1006
Dartmouth, N.S.
B2Y 4A2

1 Keiran Millard
HR Wallingford Ltd. Howbery Park,
Wallingford, UK
OX10 8BA

2 Joe Linguanti
Institute of Ocean Sciences
PO Box 6000, 9860 West Saanich Road
Sidney, B.C.
V8L 4B2

1 Dr. Roy Lowry
British Oceanographic Data Center,
Proudman Oceanographic Laboratory,
Bidston Observatory, Prenton,
Merseyside, CH43 7RA,
United Kingdom

1 Bernard Pelchat
Head, Data Management
Ocean Sciences Division
Maurice Lamontagne Institute
850, Route de la Mer
Mont-Joli, Québec
G5H 3Z4

1 Dr. Lesley Rickards
Chair, IODE
British Oceanographic Data Center,
Proudman Oceanographic Laboratory,
Bidston Observatory, Prenton,
Merseyside, CH43 7RA,
United Kingdom

1 Charles Sanders
Bureau of Meteorology
PO Box 1289k,
Melbourne, Victoria 3001
Australia

1 Dave Senciall
Physical Oceanography Section
NorthWest Atlantic Fisheries Centre
Dept. Fisheries&Oceans,Gov of Canada
Box 5667 St.John's,NF Canada A1C-5X1

1 Antoine Sioufi
Technology Transfer Advisor
Department of Fisheries and Oceans
200 Kent Street
Ottawa, Ontario,
K1A 0E6

1 Dr. Neville Smith
Bureau of Meteorology.
PO Box 1289k,
Melbourne, Victoria 3001
Australia

1 Dr. Peter Smith
Ocean Sciences Division
Bedford Institute of Oceanography
PO Box 1006
Dartmouth, N.S.
B2Y 4A2

1 Tobias Spears
Informatics
Bedford Institute of Oceanography
PO Box 1006
Dartmouth, N.S.
B2Y 4A2

20 TOTAL LIST PART 2

27 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada – Atlantic
PO Box 1012
Dartmouth, Nova Scotia
B2Y 3Z7

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

Developing an eXtensible Markup Language (XML) Application for DFO Marine Data Exchange via the Web

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Anthony W. Isenor, J. Robert Keeley and Joe Linguanti

5. DATE OF PUBLICATION (month and year of publication of document)

May 2003

6a. NO .OF PAGES (total containing information Include Annexes, Appendices, etc).

103

6b. NO. OF REFS (total cited in document)

12

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

EXTERNAL CLIENT REPORT

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).

Defence R&D Canada – Atlantic
PO Box 1012
Dartmouth, NS, Canada B2Y 3Z7

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant).

9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written).

10a ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)

DRDC Atlantic ECR 2003–025

10b OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(x) Unlimited distribution
(   ) Defence departments and defence contractors; further distribution only as approved
(   ) Defence departments and Canadian defence contractors; further distribution only as approved
(   ) Government departments and agencies; further distribution only as approved
(   ) Defence departments; further distribution only as approved
(   ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Generic data objects, or bricks, have been developed to describe the grouping characteristics of ocean data and metadata. Full definitions for the bricks were developed and the bricks where then implemented in an eXtensible Markup Language (XML) environment. The XML allows added format and content restrictions on XML elements. A full XML schema was developed for the structure representing ocean profile data.

A data exchange application was then developed by the Marine Environmental Data Service (MEDS), Institute of Ocean Sciences (IOS) and the Bedford Institute of Oceanography (BIO) (in cooperation with Defence R&D Canada) to convert profile data from the three ocean labs to the defined XML structure. Applications were developed in Java, Fortran and extensible stylesheet language transformations (XSLT) to port the XML structure to the local formats. Results show that the use of XML has merit in the ocean data community. However, considerable intellectual effort is required in defining the bricks, structure and definitions. Coding requirements are minimal in comparison and should not be considered an impediment to the development of an XML-based exchange language.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

XML
XSL
XSLT
database
ocean
data transfer
Java
msxml
Fortran

Defence R&D Canada

Canada's leader in defence
and national security R&D

R & D pour la défense Canada

Chef de file au Canada en R & D
pour la défense et la sécurité nationale

DEFENCE R&D DÉFENSE

www.drdc-rddc.gc.ca