Specification

for the

IMOS USER CODE LIBRARY

Version 1.0

Prepared by *Laurent Besnard* and *Guillaume Galibert*

IMOS - eMII

15/01/2013

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Context

IMOS (Integrated Marine Observing System) is designed to be a fully integrated national array of observing equipments monitoring the open oceans and coastal marine environment around Australia, covering physical, chemical and biological variables. All IMOS data is freely and openly available through the IMOS Ocean Portal for the benefit of Australian marine and climate science as a whole.

Most of the IMOS data is stored as NetCDF files (cf http://www.unidata.ucar.edu for more information regarding NetCDF).

Many of IMOS data users aren't computer literate enough to be able to use the NetCDF file format from scratch. A wide range of tools is already available online to help NetCDF users, but there is also a need for users who don't know where to start to include NetCDF files in their workflow.

The purpose of the "IMOS user code library" is to provide a ready to go code solution to incorporate data from NetCDF files in their working environment, starting with a NetCDF parser.

In its first release, users should be able to:

- access an IMOS NetCDF file either locally, or from an OPeNDAP URL from the same function and retrieve all the data and metadata contained in it into a variable structure.

1.2 Purpose of document

This document aims to provide specifications for the "IMOS user code library" and its first component the NetCDF parser. It will also provide whoever would like to contribute in writing or improving the "IMOS user code library" with pre-requisites information and coding best practices . This document can be applied to any scientific programming language.

1.3 Future directions

In future releases, more features could be added, such as :

- Sub-setting NetCDF file
- Output the obtained data to a CSV file
- Create some basic plots (time-series, profile, TS)
- RAMADDA search implementation of a specific data set
- iPython Notebook : a python console directly available as a web interface. For example IMOS staff could run directly some examples from this application during Data User Workshops http://ipython.org/ipython-doc/dev/interactive/htmlnotebook.html

1.4 Resources

A number of documents written by IMOS are useful resources for anyone who would like to contribute to this project:

- IMOS NetCDF Manual:
  http://imos.org.au/fileadmin/user_upload/shared/IMOS%20General/documents/Facility_manuals/IMOS_netCDF_usermanual_v1.3.pdf
- IMOS Filename Convention:
  http://imos.org.au/fileadmin/user_upload/shared/IMOS%20General/documents/Facility_manuals/IMOS_netCDF_filenaming_convention_v1.4.pdf

The followings links are access to different IMOS products:

- The IMOS toolbox used to convert raw files from instruments to IMOS compliant NetCDF files:
  http://code.google.com/p/imos-toolbox/
- IMOS Portal:
  http://imos.aodn.org.au/webportal/
- IMOS THREDDS catalog
  http://thredds.aodn.org.au/thredds/catalog/IMOS/

2. Library Requirements and Pre-requisites

2.1 GitHub Repository Structure

A public git repository is available at:
https://github.com/aodn/imos_user_code_library/

Having a GitHub repository will help developers to update the codes and keep track of the changes (such as with svn). It is also possible to contribute to this project by forking the repository. Users can download a full repository easily as a zip file without being logged in to Github.

For more information about how to interact with Github, please take a look at this help page
https://help.github.com/articles/fork-a-repo

Any person who would like to contribute to the main project will have to contact IMOS / eMII in order to become a developer of this project. Please send an email to info@emii.org.au and mention in your email you would like to contribute to the "IMOS user code library" project.

The developer will be granted with relevant permissions to work within this repository using a Github client (under Linux, we recommend RabbitVCS).
The most common tools/codes which would be the same for all the different data-types will be placed in:
https://github.com/aodn/imos_user_code_library/[programing-language-name]/commons

and ordered by type. Such as :
https://github.com/aodn/imos_user_code_library/[programing-language-name]/commons/NetCDF/
https://github.com/aodn/imos_user_code_library/[programing-language-name]/commons/plotting/
https://github.com/aodn/imos_user_code_library/[programing-language-name]/commons/ramadda/

…

where [programing-language-name] can be :

- MATLAB_[VERSION]
- R_[VERSION]
- PYTHON_[VERSION]

So that we could have a directory named MATLAB_R2008A.

A 'Where_to_Start.txt' text file, which has general information and hints for new users, and suggests which functions to start with could be added in the top directory (before the [programing-language-name] folder). This would be equivalent for all languages.

For more specific tools, a developer might have to create some codes only applicable to a unique dataset / facility / sub-facility. In this case, this set of new scripts has to be put in a facility/sub-facility folder. It is essential to keep the same folder structure as the one which already exists on the THREDDS catalog and on the portal menu, for consistency.

For example, a user wants to plot a NetCDF file from a Bio-Optical dataset which needs specific plotting tools (The Bio-Optical database is a sub-facility of SRS). The script folder structure on Github should be:
https://github.com/aodn/imos_user_code_library/[programing-language-name]/SRS/BioOptical/
in order to host the specific scripts.

Once the code's location is set up, it is important to follow the same directory structure and filenames across programing languages.

2.2 Folder Documentation

The content of each of the Github sub-directories requires a **FUNCTION_SUMMARY.txt** file which lists all the available functions and gives a quick abstract about them as well as a tutorial. Follows is a FUNCTION_SUMMARY.txt file example:

> FUNCTION_SUMMARY.txt
>
> THIS FILE LISTS ALL THE FUNCTIONS AVAILABLE IN THE CURRENT FOLDER, GIVES A SHORT DESCRIPTION AND A HOWTO
>
> Contents
>
> 1 Function 1
> 1.1 Installation
> 1.2 Dependencies
> 1.3 description
> 1.4 Use example
> 1.5 Known Issues
> 1.6 Contact
> etc...

2.3 Files Header

As well as a FUNCTION_SUMMARY file, each single function, or file needs to be properly documented. The header of each new script file should follow the following templates:

MATLAB file header :

```
%FUNCTION_NAME - One line description of what the function or script performs (H1
%line)
%Optional file header info (to give more details about the function than in the H1 line)
%Optional file header info (to give more details about the function than in the H1 line)
%Optional file header info (to give more details about the function than in the H1 line)
%
% Syntax:  [output1,output2] = function_name(input1,input2,input3)
%
% Inputs:
%    input1 - Description
%    input2 - Description
%    input3 - Description
%
```

```
% Outputs:
%    output1 - Description
%    output2 - Description
%
% Example:
%    Line 1 of example
%    Line 2 of example
%    Line 3 of example
%
% Other m-files/library required: none
% Subfunctions: none
% MAT-files required: none
%
% See also: OTHER_FUNCTION_NAME1,  OTHER_FUNCTION_NAME2
%
% Author:
% Institute:
% email address:
% Website:
% December 2012; Last revision: 30-Nov-2012
%
% Copyright 2013 IMOS
% The script is distributed under the terms of the GNU General Public License
```

PYTHON File header:

```
#!/bin/env python
# -*- coding: utf-8 -*-
#
#FUNCTION_NAME - One line description of what the function or script performs (H1
#line)
#Optional file header info (to give more details about the function than in the H1 line)
#Optional file header info (to give more details about the function than in the H1 line)
#Optional file header info (to give more details about the function than in the H1 line)
#
# Syntax:  [output1,output2] = function_name(input1,input2,input3)
#
# Inputs:
#    input1 - Description
#    input2 - Description
#    input3 - Description
#
# Outputs:
#    output1 - Description
#    output2 - Description
#
# Example:
#    Line 1 of example
#    Line 2 of example
#    Line 3 of example
#
# Other python-files/library required: none
# Subfunctions: none
```

```
#
# See also: OTHER_FUNCTION_NAME1,  OTHER_FUNCTION_NAME2
#
# Author:
# Institute:
# email address:
# Website:
# December 2012; Last revision: 30-Nov-2012
#
# Copyright 2013 IMOS
# The script is distributed under the terms of the GNU General Public License
```

2.4 Coding best practices

Among many important properties, this "IMOS user code library" should become an example of what can be a good code to interact with IMOS files and tools. This statement is even more critical when one can consider that the users who might be mostly interested in this library are the less computer literate.

As a general rule, the code must be well documented, clear, concise and robust. It is wiser to start with simple short functions before thinking about more complex features.

Some of the coding highlights are:

- Choice of external APIs / libraries
  As a general rule, it is recommended, when available, to use the most commonly used, active and up to date external library or API if there is a need to include any in the "IMOS user code library".

  For example, if there is no native NetCDF support in your targeted programming language, it is recommended when possible to use UNIDATA's API when dealing with NetCDF file format.
  One of the recommended toolbox in order to have a non-native integration of OPeNDAP capabilities within MATALB is the ***NCTOOLBOX*** (http://code.google.com/p/nctoolbox/). The installation is easy and doesn't need any compilation. With Python the ***NetCDF4*** module can be used.

  In some cases, eMII might have to consider support for the use of distinct popular libraries. These external libraries/APIs don't have to be included in the Github repository. But It is necessary to write all the required steps to install an external API onto a new machine. Such information will be make available in the main README.txt file.

- Include comments to describe your code and some major sections of it. Choice of function and variable names can act as well as comments when clearly chosen
- Closing an opened file is important!
- Use try catch error handling
- Use a 'config' text file when relevant instead of letting the user modifying critical variables inside the code. (example access to FTP server when login pwd … need to be provided)
- Check the inputs of the function are of a good type before running the script such as
  ```
  if ~iscellstr(filename), error('filename must be a cell array of strings');
  ```
- Check the number of input arguments is correct

- If a developer wants to write an already existing function into a new programing language, the same variable names should be used.

The following are some links to best programing practices for PYTHON and MATLAB developers.
http://www.python.org/dev/peps/pep-0008/
http://www.mathworks.com.au/matlabcentral/fileexchange/2529-matlab-programming-style-guidelines

3. NetCDF parser

The NetCDF parser function, named **netCDFParse**, is the core of the "IMOS user code library". This function parses a NetCDF file, either from a local address or an OPeNDAP URL, and harvests its entire content into the workspace of the programing language used.
It is important to consider both scenarios depending of the external library used to handle OPeNDAP and/or local files. For example, if coding with MATLAB, the ***nctoolbox*** can be used for both remote and local NetCDF files. With Python the ***NetCDF4*** module can be used. In addition, because of the nature of these two libraries, the code will be exactly the same whether a file is local or on an OPeNDAP server.
Both the metadata and data can be extracted. The output of the NetCDF parser can be afterwards easily used as an input of more specific functions.

Call : netCDFParse (inputFileName, '*parserOption'* , [parserOption] , 'variable' , [varList])

The output will be a data structure that is easy to read and manipulate in the relevant programming environment. Cf the ouput format section.

The following sections will describe the NetCDF parser features.

3.1 Parse a whole file

Description:
The **netCDFParse** function is used to retrieve all metadata (global attributes and variables attributes) and data from the file.

Input:
- A string of the NetCDF file 'path/address' as the first argument
- and/or the optional argument '*parserOption'* = 'all'

ex. :
netCDFParse('/path/to/netcdfFile.nc')
netCDFParse('/path/to/netcdfFile.nc' , '*parserOption'* , 'all')
Both calls are equivalent

Output:
CF. Chapter 3.4, all the dataset structure will be documented

Details on algorithms:

Open/Close a NetCDF file
If the file is not accessible, the function stops and returns an error message "netCDFParse: File <NetCDF file> not found". The function needs to open the NetCDF file, and to close it afterwards even if the function has an error statement while running (try catch exception should be used).

Retrieve global attributes of a NetCDF file
Each global attribute and its respective value is harvested. This can be done with a sub-function. We leave the choice to the developer.

MATLAB:
The output is stored as a structure called 'dataset.metadata' and each global attribute becomes a field of this structure. For example if we have a global attribute called 'abstract' with the value 'this is the abstract' , the information is therefor stored as dataset.metadata.abstract = 'this is the abstract'

PYTHON:
Similar to MATLAB but with the use of a dictionary

Retrieve variable values, variable quality control flags and variable attributes of a NetCDF file .

It is essential to only look for the variables which don't have the string '_quality_control' in their name. We only want to list the non quality control variables. For example, if there is a variable called 'TEMP' in the NetCDF file, one of the attribute of this variable should be 'ancillary_variables'. This attribute value is the name of the quality control variable which 'TEMP' is linked to. In this case it should be 'TEMP_quality_control'. If the the attribute 'ancillary_variables' is missing or empty, the developer needs to check if the variable 'TEMP_quality_control' at least exists. The QC values will be stored as shown below with the main 'TEMP' variable.

The dimensions of each variables are also retrieved in a separate branch of the structure. The dependencies are written in :
dataset.variables.<variable_shortname>.dimensions
which give the shortnames of the dimensions.
The output is stored as a structure called
'dataset.variables.<variable_shortname>' for the variables
and *'dataset.dimensions.<dimension_shortname>'* for the variable dimensions.

For example, assuming there is a variable called TEMP in a NetCDF file (The following data structure is specific to MATLAB but can be applicable to any other programming language) :

dataset.variables.TEMP=
dimensions: {'TIME'}
data: [16699x1 double]
data_CORR: [16699x1 double]
standard_name: 'sea_water_temperature'
long_name: 'sea_water_temperature'
units: 'Celsius'
FillValue: 999999
valid_min: 0
valid_max: 50
ancillary_variables: 'TEMP_quality_control'
quality_control_set: 1
flags: [16699x1 int8]
flag_meanings: [10x1 int8]
flags_values: [10x1 int8]
flag_quality_control_conventions

dataset.dimensions.TIME=
data: [16699x1 double]
standard_name: 'time'
long_name: 'time'
units: 'days since 1950-01-01 00:00:00 UTC'
axis: 'T'

valid_min: 0
valid_max: 90000
FillValue_: 999999
calendar: 'gregorian'
comment: 'timeOffsetPP: TIME dimension and time_coverage_start/end global attributes have been applied the following offset : +0 hours.'
quality_control_set: 1
quality_control_conventions
flags: [16699x1 int8]
flag_meanings: [10x1 int8]
flags_values: [10x1 int8]

Like in ARGO dataset, the data can eventually include a '_CORR' version of the variables. This means that the user will be provided with the original values, and the values that are thought to be best inferred from the QC flags. Any value with a flag strictly higher than 2 ('probably good') would be replaced by a NaN value or relevant equivalent in the current programming language.

Ex:

dataset.variables.TEMP.data = [19; 19; 20; 21; 19; 0; 20]
dataset.variables.TEMP.flags= [1; 1; 1; 1; 1; 4; 1]
dataset.variables.TEMP.data_CORR = [19; 19; 20; 21; 19; NaN; 20]

As well as harvesting the values, it is important to convert some of them so they can be directly used:

- 1-For each variable, replace the empty values of each variable with the current programming language NaN (Not a Number) value by reading the '_FillValue' attribute of each variable

- 2-Add the offset and scale factor by reading the 'add_offset' and 'scale_factor' variable attributes according to this formula:
  `varDataModified = varDataUnmodified*scake_factor+varAtt.add_offset;`

- For a TIME variable, it is necessary to convert the TIME values stored in the NetCDF file into the current programming language time reference and unit. This is done by reading the 'units' attribute from the TIME variable. The 'units' value can vary amongst datasets. It is possible to have 'days since …' or 'seconds since …' . The rest of the string is always written in the same way 'DD-MM-YYYY' (where DD=int(Day); MM=int(Month); YYYY=int(Year))

3.2 Parse Metadata only

Description:
The **netCDFParse** function can be used to retrieve only metadata (global attributes and variables attributes) from a NetCDF file and no data

Input:

- A string of the NetCDF file path/address as the first argument,
- and the optional argument '*parserOption'* = 'metadata'
  in order to retrieve only the metadata. If the optional value is missing or equal to 'all', the file will be parsed completely (cf 'parse a whole file' section')

ex. : netCDFParse('/path/to/netcdfFile.nc','*parserOption'* = 'metadata')

Output:
CF. Chapter 3.4, only the sub-structure metadata will be documented

Details on algorithms:
For more details please read **Retrieve global attributes of a NetCDF file** section above

3.3 Parse only a specified set of variables

Description:
The **netCDFParse** function can be used to retrieve only some variables chosen by the user. This is an optional feature. The NetCDF parser will take a list (cell array in Matlab) of string called '*variables'*.

Input:
A string of the NetCDF file 'path/address' and/or the optional argument
'variables' = {'PSAL' , 'TEMP'}
ex. :
netCDFParse('/path/to/netcdfFile.nc' , '*variables'* , {'PSAL' , 'TEMP'})
will only grab data and metadata for both PSAL and TEMP

netCDFParse('/path/to/netcdfFile.nc' , '*parserOption'* , 'all', '*variables'* , {'PSAL' , 'TEMP'})
will parse absolutely everything, because '*parserOption*' has the value 'all'

netCDFParse('/path/to/netcdfFile.nc' , '*parserOption'* , 'metadata', '*variables'* , { 'TEMP'})
will parse all metadata plus data only for TEMP

Details on algorithms:
If a variable name does not exist but is written in the list '*variables*', the function will return a warning such as:
'netCDFParse: WARNING variable <variable> does not exist in netcdf file <NETCDF file>'
and will return the metadata only.
For more details, please read **Parse a whole file** section above.

3.4 Output format

This is a summary of how the output of the NetCDF parser is suppose to look like. Although the parser ouput is language-specific, this section is supposed to give a summary and clear ideas:

dataset.metadata.<attributeName>='attributeValue'

dataset.variables.<variable_shortname>.

- dimensions = [<dimension_shortname1> <dimension_shortname2> ...]
- data
- standard_name
- long_name
- units
- FillValue

- valid_min
- valid_max
- ancillary_variables
- quality_control_set
- flags
- flag_meanings
- flags_values

dataset.dimensions.<dimension_shortname>.

- data
- standard_name
- long_name
- units
- axis
- valid_min
- valid_max
- FillValue
- calendar
- comment
- quality_control_set
- quality_control_conventions
- flags
- flag_meanings
- flags_values

This variable structure might be different depending on the programing language (ex Matlab vs Python <=> structures vs dictionaries) .