Specification

for the

IMOS USER CODE LIBRARY

Version 1.0

Prepared by *Laurent Besnard* and *Guillaume Galibert*

IMOS - eMII

07/02/2013

IMOS

Integrated **Marine**
Observing System

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 a NetCDF file
- Outputting the obtained data to a CSV file
- Creating 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 then ordered by areas 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/

…

and where [programing-language-name] can be :

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

So that we could have a directory named MATLAB_R2008 for example.

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 made 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 better than 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 function name 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 call to the function and so 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] , ['varList' , 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 global attributes and variable attributes. 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
'varList' = {'PSAL' , 'TEMP'}
ex. :
netCDFParse('/path/to/netcdfFile.nc' , '*varList'* , {'PSAL' , 'TEMP'})
will only grab data and metadata for both PSAL and TEMP + any dimension PSAL and TEMP are function of

netCDFParse('/path/to/netcdfFile.nc' , '*parserOption'* , 'all', '*varList'* , {'PSAL' , 'TEMP'})
Same as above

netCDFParse('/path/to/netcdfFile.nc' , '*parserOption'* , 'metadata', '*varList'* , { 'TEMP'})
will parse all metadata only for TEMP + any dimension TEMP is function of.

Details on algorithms:
If a variable name does not exist but is written in the list '*varList*', the function will return a warning such as:
'netCDFParse: WARNING variable <variable> does not exist in netcdf file <NETCDF file>'
and will act as if the variable was not in the list.
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 output is language-specific, this section is supposed to give a summary and clear ideas:

dataset.metadata.<attributeName>='attributeValue'
- netcdf_filename (this is the filename of the original NetCDF file)

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
- flag_values
- etc...
-

dataset.dimensions.<dimension_shortname>.

- data
- standard_name
- long_name
- units
- axis
- valid_min
- valid_max
- calendar
- comment
- quality_control_set
- quality_control_conventions
- flags
- flag_meanings
- flag_values
- etc...

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

4. Output to a CSV file

A NetCDF file can be parsed into the working environment of the programing language used by a user with the function *netCDFParse*. Outputting a parsed NetCDF into a CSV file (comma-separated values), with options defined by the user, can be performed with the function ***outputCSV*** described in this section.

The exported CSV file will have the same information as one can find in the original NetCDF file:
- metadata
- variable attributes
- QC flags for data quality

The CSV file once created is characterized by a text delimiter ' **"** ' and and cell delimiter ' **,** ' .

A user will find many interests in this tool, such as importing afterwards the data into the software of his choice (for example Excel or LibreOffice) in order to have a more visual interaction with the data.

Call :
outputCSV (**dataset**, 'varList' , [**varList**] , '**folderOutput**' , [folderOutput])

Inputs:
- **dataset** is the output from the function *netCDFParse* (cf section regarding this function for more details)
- '**varList**' is an optional input. This lets the user the choice to output only a set of variables. If this argument does not exist, all the variables found in 'dataset' (output from *netCDFParse*) will be exported (cf further sections for more details)
- '**folderOutput**' is an optional input so the user can choose the folder of his choice where the csv file(s) will be created. If this argument does not exist, the function will create the files in the home directory of the user (either the environment is Linux or Windows).

Warning messages:
- if the variable chooser by the user does not exist, the function will return a warning message "No Variable to export"
- if the **folderOuput** can't be created, an error will be return "Permission Denied"

The following sections will describe the outputCSV function features.

4.1 Dimensions of the variables to export

Description:

The function highly relies on the dimensions of the variables to export. It is therefor essential to check the size or dimension of each variable to export (chosen or not by the user). The CSV template used has to be slightly different depending of the variable's dimensions.

Details on algorithms:

File-naming
Only variables with the same dimensions dependencies can be exported in a same CSV file.

In the case there are variables with different dimensions dependencies in the original parsed NetCDF file, such as :
TEMP (TIME) → temperature is a function of time
SAL (DEPTH) → salinity is a function of the depth

these two variables cannot be exported in the same CSV file for a matter of order and readability for the user. Two files will be created with a different filename following this naming convention:
<***original NetCDF filename minus extension***>_<***dimensionName***>*_DimensionDependency.csv*

i.e. in our example, this would become:

IMOS_ANMN-NSW_T_20010911T000000Z_WAVESYD_FV01_END-20040518T010000Z_**TIME_DimensionDependency**.csv
IMOS_ANMN-NSW_T_20010911T000000Z_WAVESYD_FV01_END-20040518T010000Z_**DEPTH_DimensionDependency**.csv

Find variables dimensions
It is important to find out the different dimensions of each variable. If one variable is a function of only one dimension, the problem is straightforward. This variable will be exported in a CSV file following a 1D type template (cf annex).

However a variable can be a function of many dimensions and still be exported in a CSV file following a 1D type template. For example :
TEMP (TIME , LATITUDE, LONGITUDE)
But the size of '*TEMP*' is [100 ,1 ,1] → which means the variable *TEMP* is a 3D variable. Only the time dimension varies (100 values) while *LATITUDE* and *LONGITUDE* dimensions have only one value.

It is therefor essential to check the size or length of each of the variable dimensions in order to export the variable in the good CSV template category (1D or 2D template)

4.2 Output for 1D type variable

Description:

A sub-function outputCSV_1D can be created in order to export 1D type variables only.

Details on algorithms:

4.2.1 Global Attributes

All the global attributes found in dataset.metadata (from *netCDFParse*) need to be exported in the CSV file under the section name called "**GLOBAL ATTRIBUTES**".
(cf output format for more details)

4.2.2 Variables' Attributes

All the variables to be exported will have their information written down in the next section called "**VARIABLES**".

A header of variable attributes will also be written down in the next line. This header is the same for all the variables. In the case one variable has different attribute names that another variable to export, both variables' attributes will be merged to create this header. If one variable does not have any information about an attribute, nothing has to be written in the CSV file, except another delimiter.

Some attributes which don't have much of a purpose in a CSV file can be removed from the header, such as :

- flag
- FillValue
- dimensions

4.2.3 Dimensions' attributes

Following the "**VARIABLES**" section is the "**DIMENSION**"'s one. This section will list the main dimension. This part follows the same rules as above regarding the "**VARIABLES**" section.

4.2.4 Optional Dimensions' Attributes

A 1D type variable can be a function of other dimensions with a size equal to one. In this case, those dimensions need to be listed in the section called "**OTHER DIMENSIONS (SIZE == 1)**" .

This part follows the same rules as above regarding the "**VARIABLES**" section.

Another header attribute needs to be added, called "data". This is the unique value of this dimension which needs to be written down on the next line of the CSV. This is extremely helpful to the user (for example to know the location of a mooring).

4.2.5 Data Values

The data is finally written in the next section called "**DATA**".

The dimension variable is written in the first column, followed by all the variables and their respective QC flags if they exist in this order :
"<dimensionName>" , "<varName1>" , "<varName1_flags>" , "<varName2>" , "<varName2_flags>" ...

The time values need to be modified in order to be human readable (either *TIME* is a variable or a dimension). The time format to use in the CSV file is :
"yyyy-mm-ddTHH:MM:SSZ"

i.e. :
for the 9th of September 2012 at 01:40 am and 16seconds, the time is written this way
"2012-08-09T01:40:16Z"

Output format
CF annexes to have a better comprehension of the output format :

4.3 Output for 2D type variable

Not yet written

ANNEXES

Template of a 1D type variable for a CSV file

GLOBAL ATTRIBUTES

title	Great Barrier Reef Ocean Observing System Mooring Array - GBROOS Davies Reef Sensor Float 5 Water Temperature @.6m
author_email	s.bainbridge@aims.gov.au
naming_authority	IMOS
netcdf_filename	IMOS_FAIMMS_T_20120809T014016Z_DAVSF5_FV01_END-20120822T030536Z_C-20121017T035848Z.nc

The header is in the same order for all variables

VARIABLES

shortname	*coordinates*	*flag_meanings*	*flag_quality_control*	*flag_values*	*long_name*	*quality_control_set*	*sensor_depth*	*standard_name*	*units*
TEMP	TIME LATITUDE LONGITUDE	no_qc_performed good_	AIMS standard set	[0;1;2;3;4;5;6;7;8;9]	sea_water_tempera	[]	0.6	sea_water_tempera	degrees celsius

Flag values information

DIMENSIONS

shortname	*long_name*	*standard_name*	*units*	*observation_type*	*CoordinateAxisType*
TIME	time	time	seconds since 197(	measured	Time

OTHER DIMENSIONS (SIZE == 1)

This part is only here when a variable has other dimensions with a size == 1

shortname	*long_name*	*standard_name*	*units*	*CoordinateAxisType*	*data*
LATITUDE	latitude	latitude	degrees_north	Lat	-18.8187198639
LONGITUDE	longitude	longitude	degrees_east	Lon	147.6524353027

Time values are human readable

QC flag values

The dimension data is added here.

DATA

TIME	TEMP		TEMP_flag_QC
2012-08-09T01:40:16Z		21.8332996368	2
2012-08-09T01:42:24Z	NaN		4
2012-08-09T02:05:52Z	NaN		4
2012-08-09T02:20:48Z	NaN		4
2012-08-09T02:35:44Z	NaN		4
2012-08-09T02:50:40Z	NaN		4
2012-08-09T03:05:36Z	NaN		4
2012-08-09T03:20:32Z	NaN		4
2012-08-13T23:00:16Z	NaN		4
2012-08-21T22:45:20Z		20.6480007172	1
2012-08-21T22:49:36Z		20.6340007782	1
2012-08-21T22:56:00Z		20.6469993591	1
2012-08-21T23:00:16Z		20.7019996643	1

GLOBAL ATTRIBUTES

site	M/V SeaFlyte
project	Integrated Marine Observing System
conventions	IMOS version 1.3
title	Sea surface temperature data
institution	Australian Bureau of Meteorology
netcdf_filename	IMOS_SOOP-SST_T_20111110T000000Z_VHW5167_FV01_C-20120529T002258Z.nc
...	...

VARIABLES

shortname	*CoordinateAxisTyp*	*axis*	*flag_meanings*	*flag_quality_control*	*flag_values*	*instrument*	*long_name*	*observation_type*	*quality_control_set*	*reference_datum*	*standard_name*	*units*	*valid_max*	*valid_min*
PL_SPD			Value_out_of_boun	IMOS Reference Ti	B, C, D, E, F, G, H	Garmin GPS16-HV:	platform speed ove	measured	3		platform_speed_wr	meter second-1		
TEMP			Value_out_of_boun	IMOS Reference Ti	B, C, D, E, F, G, H	Seabird SBE38 (s/r	sea temperature	measured	3		sea_surface_temp	celsius		
LATITUDE	Lat	Y	Value_out_of_boun	IMOS Reference Ti	B, C, D, E, F, G, H	Garmin GPS16-HV:	latitude	measured	3	geographical coordi	latitude	degrees_north	90	-90
LONGITUDE	Lon	X	Value_out_of_boun	IMOS Reference Ti	B, C, D, E, F, G, H	Garmin GPS16-HV:	longitude	measured	3	geographical coordi	longitude	degrees_east	180	-180

DIMENSIONS

shortname	*standard_name*	*long_name*	*units*	*axis*	*valid_min*	*valid_max*	*comment*	*ancillary_variables*	*CoordinateAxisType*
TIME	time	time	days since 1950-01T			0	90000 Relative julian days	TIME_quality_contr	Time

DATA

TIME	PL_SPD	PL_SPD_flag_QC	TEMP	TEMP_flag_QC	LATITUDE	LATITUDE_flag_Q	LONGITUDE
2011-11-10T00:00:00Z	3.7039999962	Z	20.0720005035	Z	-31.9959106445	Z	115.5457992554
2011-11-10T00:16:00Z	6.0061392784	Z	21.7842502594	J	-31.995470047	Z	115.5469970703
2011-11-10T00:17:00Z	11.2063140869	Z	20.3061695099	Z	-31.9952297211	Z	115.5520019531
2011-11-10T00:18:00Z	11.6264448166	Z	20.1604995728	Z	-31.9937095642	Z	115.5589981079
2011-11-10T00:19:00Z	11.6264448166	Z	20.2618293762	Z	-31.990020752	Z	115.5650024414
2011-11-10T00:20:00Z	11.5235557556	Z	20.31016922	Z	-31.9860801697	Z	115.570602417
2011-11-10T00:21:00Z	11.4549608231	Z	20.375	Z	-31.9819202423	Z	115.5761032104
2011-11-10T00:22:00Z	11.1548690796	Z	20.3413295746	Z	-31.977930069	Z	115.5815963745
2011-11-10T00:23:00Z	11.4292421341	Z	20.3225002289	Z	-31.9738407135	Z	115.5867004395
2011-11-10T00:24:00Z	11.369222641	Z	20.3659992218	Z	-31.9701309204	Z	115.5924987793
2011-11-10T00:25:00Z	11.2148895264	Z	20.3923301697	Z	-31.966709137	Z	115.598197937
2011-11-10T00:26:00Z	11.2320356369	Z	20.3073291779	Z	-31.9627304077	Z	115.6036987305
2011-11-10T00:27:00Z	11.2920560837	Z	20.2784996033	Z	-31.958770752	Z	115.6091003418
2011-11-10T00:28:00Z	11.26633358	Z	20.2791690826	Z	-31.9551200867	Z	115.6148986816
2011-11-10T00:29:00Z	11.3777990341	Z	20.2800006866	Z	-31.9516391754	Z	115.6207962036
2011-11-10T00:30:00Z	11.472111702	Z	20.2503299713	Z	-31.9480209351	Z	115.626701355
2011-11-10T00:31:00Z	11.5149803162	Z	20.2163295746	Z	-31.9441604614	Z	115.6324005127
2011-11-10T00:32:00Z	11.3777990341	Z	20.2403297424	Z	-31.9401493073	Z	115.6380004883
2011-11-10T00:33:00Z	11.3349256516	Z	20.2676696777	Z	-31.9365196228	Z	115.6437988281
2011-11-10T00:34:00Z	11.4635362625	Z	20.2733306885	Z	-31.9327106476	Z	115.6494979858
2011-11-10T00:35:00Z	11.3092021942	Z	20.2808303833	Z	-31.9289207458	Z	115.6549987793
2011-11-10T00:36:00Z	11.3435001373	Z	20.2771701813	Z	-31.924659729	Z	115.6604003906
2011-11-10T00:37:00Z	11.4635362625	Z	20.2928295135	Z	-31.9202804565	Z	115.665397644
2011-11-10T00:38:00Z	11.4549655914	Z	20.3216705322	Z	-31.9156608582	Z	115.6703033447