hydro Documentation

Andrew Barna

Feb 02, 2023

CONTENTS:

CHAPTER

ONE

GO-SHIP CF/NETCDF DATA FORMAT SPECIFICATION

1.1 Introduction

The traditional way that formats are thought about and described is via "file formats". How the bytes arranged on disk, what the data types are, maybe even just some text with numeric characters in it. Instead of focusing on how the data exist "at rest" or "on disk", netCDF instead describes a data model and an API (application programer interface) for data access. Rather than specify the "on disk" format, we instead will specify a data model, and any format that supports the netCDF enhanced data model and can be accessed via the netCDF API is acceptable. At the time of writing, the two most common at rest formats include HDF5, the traditional netCDF4 file, and zarr, a "cloud native" format designed for non disk/seekable storage mediums.

Requirements

The "on disk" or storage format is anything that supports:

- Access via the netCDF4 software library API
- The data and metadata model presented in this document

1.2 Requirement Levels

When specifying our requirements, we will follow the guidelines set in BCP 14:

> The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Additionally, we will try very hard to make sure all the requirements are in clearly marked admonitions.

Danger: These requirement levels specify our "guarantees" when accessing the data and are specific to each published version. We will try very hard not to change things in a non backwards compatible way, but there may always be mistakes, bugs, community standards changes, etc., that require something breaking.

If something is not specified here, you MUST assume it is undefined behavior that MAY change at any time.

1.3 Conventions

To increase data reusability and ease of access it is very useful to follow one or more community conventions when making decisions about data layout, what metadata to include, etc. The CF Metadata Conventions were chosen as the base for our data model, with influences of Argo and OceanSITES. Specifically, we are using the NetCDF Climate and Forecast (CF) Metadata Conventions version 1.8

Requirements

The data and metadata MUST conform to NetCDF Climate and Forecast (CF) Metadata Conventions version 1.8

Note: Internally, CCHDO is using xarray as the base for almost all the internal software working with netCDF. The internal data model of xarray is very close to but not exactly CF 1.8, and is a subset of what CF 1.8 allows. We have found that accepting the minor limitations of xarray to be worth the access to the large scientific software ecosystem that has developed around it.

1.4 File Naming Conventions

When data are being distributed or shared using files, computer systems often rely on a file extension to identify the file type.

Requirements

As per CF-1.8 Section 2.1, netCDF HDF5 files SHOULD have the extension `.nc`

At CCHDO, out usual data management granularity is cruise/leg, separated by discrete sample (bottle) and continuous sample (CTD) data types. As a convenience, an additional suffix may be added to easily identify data containing only bottle or CTD data.

Requirements

- netCDF files containing exclusively bottle data MAY end with `_bottle.nc`.
- netCDF files containing exclusively ctd data MAY end with `_ctd.nc`.
- netCDF files containing mixed ctd and bottle data MUST NOT end with either `_bottle.nc` or `_ctd.nc`.

1.5 Definitions

The terminology used to describe netCDF data tends to be very technical, with very specific definitions. To confuse things, the netCDF user guide, the CF conventions, and popular software tools such as xarray all have slight variations on their usage of these definitions. Due to this, we feel the need to include some of these definitions here.

dimension
: The netCDf data model is primarily concerned with storing data inside arrays, almost always this is numeric data. A netCDf dimension is the size of one side of these arrays and is given a name to reference it by. For example, a 2-d array of shape NxM has dimensions N and M. netCDf supports arrays with no dimensions, a scalar.

variable
In a netCDF file, a variable is the most basic data object Variables have a name, a data type, a shape, some attributes, and the data itself. Variable names can be almost anything, the only character not allowed in a netCDF variable name is the forward slash "/". Names may start with or contain anything in unicode, they may not be valid variable names in your programing environment of choice.

> **Warning:** It is also important to understand that variable names are simple labels and not data descriptors. If the name does have some human readable meaning, it often meant to help quickly identify which variables might be of interest, not describe the variable with scientific rigor. Do not rely on the inferred meaning of a variable name unless you have no other source of information (attributes, documentation, emails from colleagues, etc.).

ancillary variable
In CF, an ancillary variable is still a normal variable described above, but it contains information about other variables. Perhaps the most common example of an ancillary variable is the quality control flag, but also include information such as uncertainties. Some of the carbon data have strong temperature dependencies and so the temperature of analysis might be reported along side in an ancillary variable.

coordinate
Coordinates are variables that provide the labels for some axis, usually for identifying data in space and time. The typical examples of coordinates are longitude (X-axis), latitude (Y-axis), and time (T-axis). The vertical coordinate is a little more varied, usually oceanographic observation data will use pressure as the Z-axis coordinate.

Xarray calls these "coordinates"

coordinate variables
In many netCDF aware applications there is a special case of variables called "coordinate variables" or "Dimension coordinate". The technical way you will see this defined is as a single dimensional variable that has the same name as its dimension. There tend to be other rules most programs enforce: there must be no missing values, values must be numeric, and values must be monotonic. These are most useful when the data occur on some regular grid.

Perhaps a good way to think of coordinates variables is as the values the ticks would be in a figure plot.

Xarray calls these "Dimension coordinates" and will be shown with a little asterisk * when exploring an xarray Dataset.

auxiliary coordinate
Auxiliary coordinates or "Non-dimension coordinates" are variables that do not share the same names as a dimension. These variables still label axes, but are more flexible for when the data do not occur on a regular grid or when there are multiple sets of coordinates in use. Auxiliary coordinates may be multidimensional. CF requires auxiliary coordinates to appear in the `coordinates` attribute of the variables it labels.

Xarray calls these "Non-dimension coordinates" and will not have an asterisk next to their names when exploring an xarray dataset.

attribute
Attributes are extra pieces of data that are attached to each variable and is where the flexibility of netCDF to describe data is greatly enhanced. Attributes may also be attached at the "global" level Attributes are simple "key" to "value" mappings, the computer science term for these is "associative array". Python and Julia calls these "dictionaries", in matlab these are usually "Structure Arrays".

Most of the focus of the common community data standards, CF, ACDD, OceanSITES etc., are on defining attribute keys, values, and how to interpret them. CF defines and controls attributes important to CF, but then allows any number of extra attributes.

1.6 Dataset Structure

Todo: write overview

- which CF DSG
- Dims * Strings vs Char arrays
- Global attributes
- Required variables
- Technical variables and attrs (the geometry ones)
- Notes on strings and chars * Encoding, line endings * where are actual strings allowed, netCDF4 python forces string types if non ascii

1.6.1 Dimensions

There are two basic dimensions in the data file, how many profiles there are, and how many vertical levels there are. The two dimension names match the dimenion names found in argo profile files: N_PROF and N_LEVELS.

While netCDF4 supports an actual string data type, for compatibility and compression reasons, character arrays will be used to represent text data. Character arrays have the string length as their last dimension, the number and values of these string dimensions is currently uncontrolled (xarray sets these automatically). All char arrays or strings will be UTF-8 encoded.

Requirements

- There MUST be a dimension named `N_PROF` that describes the first axis of variables with a "profile" dimension.
- There MUST be a dimension named `N_LEVELS` that describes the first axis of variables with no "profile" dimension, or the second axis of variables with a "profile" dimension
- There MAY be zero or more string length dimensions.
- Extra dimensions MAY exist if needed by data variables, these extra names are not standardized.
- Any char array or strings, both in variable and attributes, MUST be UTF-8 encoded and MUST NOT have a byte order mark.

Note: There is currently a single variable which requires an additional dimension to describe the radiation wavelength of its measurement. This dimension is currently called `CDOM_WAVELENGTHS` and is stored as the only coordinate variable in use. The actual relationship between the parent variable and this coordinate is contained in attributes defined by the CF conventions.

1.6.2 Global Attributes

Global attributes contain information that applies to the entire dataset. Some of these are defined by community standards, other by this document for internal use. The following, case sensitive, global attributes are REQUIRED to be present:

Conventions
: Conventions is a char array listing what community standards and their versions are being followed. It MUST have the value "CF-1.8 CCHDO-1.0" and will change as new conventions are adopted

featureType
: The feature type char array attribute comes from the CF conventions section about discrete sampling geometries. It MUST have the value "profile"

cchdo_software_version
: The cchdo software version is a char array containing the version of the cchdo.hydro library used to create or manipulate the dataset. It currently takes the form of "hydro w.x.y.z" where w.x is the data conventions version, and y.z is the actual software library version.

cchdo_parameters_version
: The cchdo parameters version char array contains the version for the internal parameters database the software was using at the time of dataset creation or manipulation. It currently takes the form of "params x.y.z".

The following, case sensitive, global attributes are OPTIONAL:

comments
: Comments human readable string containing information not captured in any other attributes or variables.

Requirements

- There MUST be a `Conventions` global attribute char array with space separate convention version strings defined by those conventions.
- There MUST be a `featureType` global attribute char array with the value "profile".
- There MUST be a `cchdo_software_version` global attribute char array with the version string of the cchdo.hydro software.
- There MUST be a `cchdo_parameters_version` global attribute char array with the version string of the cchdo.params database.
- There MAY be a `comments` attribute with more information. This attribute MAY be a string rather than a char array if there are non ASCII code points present.

1.6.3 Variable Attributes

Todo: Attrs to talk about:

- whp_name
- whp_unit
- geometry
- _Encoding
- coordinates
- ancillary_variables

- standard_name
- flag_values
- flag_meanings
- conventions
- resolution (time)
- axis
- units
- calendar
- C_format
- C_format_source
- positive
- reference_scale
- geometry_type
- node_coordinates

1.6.4 Required Variables

The following variables are required in all files:

- `geometry_container`
- `profile_type`
- `expocode`
- `station`
- `cast`
- `sample`
- `longitude`
- `latitude`
- `pressure`
- `time`

CHAPTER

TWO

ATTRIBUTES

2.1 Variable Level Attributes

2.1.1 _FillValue

dtype
same as the variable

usage
variables

required
only if there are missing data

reference
CF-1.8, NUG

CF Definiton

A value used to represent missing or undefined data. Allowed for auxiliary coordinate variables but not allowed for coordinate variables.

CCHDO Usage

For floating point type data (float, double), the IEEE NaN value will be used. Woce flag variables will be initialized with the value *9b*. Some special coordinate variables are not allowed to have any `_FillValue` values in them

The `_FillValue` attribute has special meaning to the netCDF4 libraries (C and Java). When the size of the variable is known (i.e. the variable does not have an "unlimited" dimmension) at the time the netCDF file is written, all of the space in the variable will be initalized with the value in `_FillValue`. This is usually almost entirely transparent to you the user, some software will change the data type when a variable still contains `_FillValue` values. Matlab for example, will change byte (integers between 0 and 255) data into IEEE floating point values while replacing the fill value with NaNs.

2.1.2 whp_name

dtype
char or array of strings

usage
variables

required
conditionally (see CCHDO Usage)

reference
CCHDO

CF Definiton

Not used or defined in the CF conventions.

CCHDO Usage

This attribute contains the name this variable would have inside a WHP Exchange or WOCE sea/ctd file. Forms a pair with whp_unit This attribute will only be on variables which are data, and not on flag variables or certain specal variables meant to be interpreted by CF compliant readers (e.g. `geometry_container`).

Some variables cannot be represented by a single column in the WHP Exchange format, when this occurs, the attribute will be an array of strings containing all the names needed to represent this variable in WHP Exchange format. The most frequent example will be the `time` variable, in WHP Exchange files, this may either be a pair of columns (DATE, TIME) or a single column (DATE) when time of day is not reported. This will very likly be used to represet ex and em wavelengths for optical sensors with multiple channels.

There is no requiremnt that all variables in a netCDF file contain unique `whp_name` and `whp_unit` pairs.

2.1.3 whp_unit

dtype
char or array of strings

usage
variables

required
conditionally (see CCHDO Usage)

reference
CCHDO

CF Definiton

Not used or defined in the CF conventions.

CCHDO Usage

For this variable, the value which would appear in the units line of the WHP Exchange or WOCE sea/ctd file. Forms a pair with :ref:`whp_name` Usage is the same as `whp_name`

2.1.4 `standard_name`

dtype
: char

usage
: variables

required
: conditionally (see CF Usage)

reference
: CF 1.8

CF Definiton

Todo: get cf definiton

CCHDO Usage

The CF usage will be followed, if a CF standard name exists for physical quantity represeted by a variable, the most specific name MUST be used and appear in the `standard_name` attribute. The CF standard names table is updated frequently, as names are added they will be evaluated for including in the CCHDO netCDF files to both be more specific or to add a standard name to a variable that did not have one previously. Always check the param version attribute to see which version of the standard name table is in use for a particular file.

It is important to understand that standard names represet the physical quantity of the variable and not how that physical quantity was made. Standard names cannot distinguish between salinity measured in situ with a CTD, salinity measured with an autosal, or even salinity from a model output. The names are meant to help with intercomparison of the values themselves, not methods of determing that value.

2.1.5 `units`

dtype
: char

usage
: variables

required
: conditionally

reference
CF 1.8

CF Definiton

Todo: get cf definiton

CCHDO Usage

The units attribute will follow CF. The value must be physically comparible with the canonical units of the `standard_name`. The value will be the `whp_unit` translated into SI.

Unitless parameters will have the symbol "1" as their units.

Todo: get ref to SI paper

Some examples:

- discintigrations per minute (DPM) will be translated to their equivalent Bq, which will be scaled (1DPM = 0.0166 Bq)
- Practical salinity will have the units of "1", not variabtions on "PSU" or even "0.001" implying g/kg of actual salinity.
- Tritium Units are really parts per 1e18, so the equivalent SI units are the recriprical: 1e-18

2.1.6 `reference_scale`

dtype
char

usage
variables

required
conditionally

reference
OceanSITES 1.4

CF Definiton

This attribute is not defined in CF.

CCHDO Usage

Todo: get OceanSITES definition.

Some variables (e.g. temperature) are not described well enough by their units and standard name alone. For example, depending on when it was measured, the temperature sensors may have been calibrated on the ITS-90, IPTS-68, or WHAT_WAS_BEFORE_t68 calibration scales. While all the temperatures are degree C, users doing precice work need to know the difference.

Todo: this is a controlled list internally, list which variables have a scale and what their value can be.

2.1.7 `C_format`

dtype
: char

usage
: variables

required
: no

reference
: NUG

CF Definiton

`C_format` is not mentioned at all in the CF-1.8 docs.

CCHDO Usage

The `C_format` attribute will contain the format string from either the internal database of parameters or calcualted when converting from a text input. The presence or lack of presence of this attribute will not change the underlyying values in the variable (e.g. you cannot round the values to the nearst integer using C_format). This attribute is sometimes used when _displaying_ data values to a user. When performing calculations in most software, the underlying data values are almost always used directly. The only software we have seen respect the `C_format` attribute is ncdump when dumping to CDL.

If the data soure for this variable came from a text source, the `C_format` will contain the format string which represents the largest string seen. For example, if a data source had text values of "0.001" and "0.0010", the `C_format` attribute would be set to `"%.4f"`. This can be tricky for data managers: if for example, the data source was an excel file, it is important to use the underlying value as the actual data and not a copy/paste or text based export.

Warning: Use `C_format` as implied uncertanty if you have *no other* source of uncertanty (including statistical methods across the dataset).

Historically, storing numeric values in text and the cost of storage meant there was a tradeoff between cost and precision. When looking though our database of format strings, the text print precision was almost always set at one decimal place more than the actual measuremnt uncertanty. Having these values published in the WOCE manual also lead to values being reported a certain way to conform to the WOCE format, which disconnected "print

precision" from uncertanty. Additionally, the WOCE format was designed when IEEE floating point numbers were quite new.

More recent measuremnets have started to include explicit uncertanties which will be reported along side the data values. Often, the cruise report will contain some charicterizaion of the uncertanty of a given measumrnet.

2.1.8 C_format_source

dtype
char

usage
variables

required
yes if C_format is present

reference
CCHDO

CF Definiton

This attribute is not used in CF.

CCHDO Usage

This attribute describes where the value in `C_format` came from. This attribute will only have the values of either `"database"` to indicate the `C_format` was taken from the internal parameters table, or `"source_file"` if the values was calcualted from input text.

Todo: Attrs:

Variable Level:

- ancillary_variables
- coordinates
- flag_values
- flag_meanings
- conventions
- _Encoding
- geometry_type
- node_coordinates
- axis
- geometry

Global Level:

- Conventions
- cchdo_software_version

- cchdo_parameters_version
- comments
- featureType

ACDD Things we want at variable level:

- creator_name
- creator_email
- processing_level
- instrument
- instrument_vocabulrary
- comments (more of them)
- contributor_name
- contributor_email
- contributor_role

Non ACDD thing var level:

- program_group

Non ACDD global level?:

- platform (ICES ship code)
- start/end ports
- actual start/end dates

Huge TODO… history at the var and global level, including seperation between metadata and data history.

CHAPTER

THREE

VARIABLES IN ERDDAP

Table 1: Variables In ERDDAP

netcdf varname	Units	In ERDDAP
expocode	None	yes
section_id	None	yes
line_id	None	yes
station	None	yes
cast	None	yes
bios_castid	None	no
sample	None	yes
geotraces_event	None	no
geotraces_sample	None	no
bionbr	None	no
event_number	None	no
bottle_number	None	yes
date	None	yes
time	None	yes
latitude	None	yes
longitude	None	yes
btm_depth	meters	yes
pressure	dbar	yes
ctd_pressure_raw	dbar	no
ctd_temperature_unk	degC	yes
ctd_temperature_68	degC	yes
ctd_temperature	degC	yes
ctd_salinity	1	yes
ctd_absolute_salinity	g/kg	no
ctd_conservative_temperature	degC	no
bottle_salinity	1	yes
density_salinity	g/kg	no
density_salinity2	g/kg	no
refractive_index_anomaly	1	no
density_salinity_practical_salinity	1	no
density_salinity_practical_salinity2	1	no
ctd_sound_velocity_salinity	g/kg	no
ctd_oxygen_ml_l	ml/l	yes
ctd_oxygen	umol/kg	yes
ctd_oxygen_umol_l	umol l-1	yes
ctd_optode_oxygen	umol/kg	no

continues on next page

Table 1 – continued from previous page

netcdf varname	Units	In ERDDAP
ctd_optode_oxygen_raw	volts	no
oxygen_ml_l	ml/l	yes
oxygen	umol/kg	yes
silicate	umol/kg	yes
silicate_l	umol l-1	yes
ammonium	umol/kg	yes
nitrate	umol/kg	yes
ctd_nitrate	umol/kg	no
nitrite	umol/kg	yes
phosphate	umol/kg	yes
phosphate_l	umol l-1	yes
nitrite_nitrate	umol/kg	yes
nitrite_nitrate_l	umol l-1	yes
cfc_11	pmol/kg	yes
cfc_11_l	pmol/l	yes
cfc_12	pmol/kg	yes
cfc_12_l	pmol/l	yes
cfc_113	pmol/kg	yes
cfc_113_	pmol/l	yes
dichlorofluoroethane	pmol/kg	no
chlorodifluoroethane	pmol/kg	no
chlorodifluoromethane	pmol/kg	no
sulfur_hexifluoride	fmol/kg	yes
sulfur_hexifluoride_l	fmol/l	yes
total_carbon	umol/kg	yes
total_alkalinity	umol/kg	yes
fco2	uatm	yes
fco2_in_situ	uatm	no
fco2_temperature	degC	yes
partial_pressure_of_co2	uatm	yes
co2_mole_fraction	1e-6	no
partial_co2_temperature	degC	yes
ph_total_h_scale	None	yes
ph_unknown_scale	None	yes
ph_nbs	None	no
ph_sws	None	yes
ph_temperature	degC	yes
dissolved_organic_carbon	umol/kg	yes
fdom	1	no
dissolved_organic_carbon_nasa	umol l-1	no
tritium_activity	kBq m-3	yes
tritium	1e-18	yes
helium	nmol/kg	yes
helium_l	nmol/l	yes
delta_helium_3	percent	yes
ref_temperature_c	degC	yes
ref_temperature	degC	yes
rev_pressure	dbar	yes
rev_temperature_c	degC	yes

continues on next page

Table 1 – continued from previous page

netcdf varname	Units	In ERDDAP
rev_temperature	degC	yes
rev_temperature_90	degC	yes
del_carbon_13_dic	1e-3	yes
del_carbon_14_dic	1e-3	yes
dissolved_organic_nitrogen	umol/kg	yes
total_organic_carbon	umol/kg	yes
total_organic_carbon_l	umol l-1	yes
particulate_organic_carbon	ug/kg	yes
particulate_organic_carbon_l	ug/l	yes
d13c_poc	1e-3	no
particulate_organic_nitrogen	ug/kg	yes
particulate_organic_nitrogen_l	ug/l	yes
particulate_organic_nitrogen_mol	umol l-1	yes
particulate_organic_phosphorus_l	ug/l	no
particulate_organic_phosphorus	umol l-1	no
particulate_chemical_oxygen_demand_l	ug/l	no
dissolved_organic_phosphorus	umol/kg	no
total_dissolved_phosphorus	umol/kg	no
total_dissolved_phosphorus_l	umol l-1	no
dissolved_atp	pmol/l	no
particulate_atp	pmol/l	no
total_dissolved_nitrogen	umol/kg	yes
total_organic_nitrogen	umol/kg	no
total_organic_nitrogen_l	umol l-1	no
neon	nmol/kg	no
neon_l	nmol/l	no
del_oxygen_18	1e-3	yes
del_oxygen_17	1e-3	no
del_deuterium	1e-3	no
delsi30	1e-3	no
del_nitrogen_15	1e-3	no
carbon_tetrachloride	pmol/kg	yes
carbon_tetrachloride_l	pmol/l	yes
nickel	umol l-1	no
dissolved_aluminum	nmol/l	no
barium	nmol/kg	yes
barium_l	nmol/l	yes
copper	umol l-1	no
iron	nmol/l	no
manganese	nmol/l	no
ctd_fluor	mg/m^3	yes
ctd_fluor_arbitrary	None	yes
ctd_fluor_raw	volts	yes
par	umol m-2 s-1	yes
par_raw	volts	yes
cdom300	m^-1	yes
cdom325	m^-1	yes
cdom340	m^-1	yes
cdom380	m^-1	yes

continues on next page

Table 1 – continued from previous page

netcdf varname	Units	In ERDDAP
cdom412	m^-1	yes
cdom443	m^-1	yes
cdom490	m^-1	yes
cdom555	m^-1	yes
spar_raw	volts	no
cdom2c	None	no
cdom3c	None	no
cdomsl	1/nm	yes
cdomsn	1/nm	yes
iodine_129	Bq m-3	no
plutonium	mBq m-3	no
radium_226	0.000166 Bq/kg	yes
radium_228	0.000166 Bq/kg	yes
ctd_transmissometer	1e-2	yes
ctd_transmissometer_raw	volts	yes
ctd_beamcp	m^-1	yes
beamap	m^-1	no
ctd_beta700	m-1 sr-1	yes
ctd_beta700_raw	volts	yes
ctd_bbp700	m^-1	yes
ctd_turbidity_ftu	1	yes
ctd_turbidity_ntu	1	yes
ctd_cdom	1	yes
ctd_cdom_raw	volts	yes
argon_39	1e-2	no
cesium_137_bq	Bq m-3	no
cesium_137	0.000166 Bq/kg	no
cesium_137_bq_kg	mBq kg-1	no
cesium_134_bq	Bq m-3	no
cesium_134_bq_kg	mBq kg-1	no
krypton_85	0.0000166 Bq/kg	no
strontium_90	0.000166 Bq/kg	no
nitrous_oxide	nmol/kg	yes
radium_228_226	None	no
quality_word_one	None	no
quality_word_two	None	no
methyl_chloroform	pmol/kg	no
iodate	nmol/kg	no
iodide	nmol/kg	no
chlorophyll_a_ug_kg	ug/kg	yes
chlorophyll_a	ug/l	yes
phaeophytin	ug/kg	yes
phaeophytin_ug_l	ug/l	yes
methyl_chloride	pmol/kg	no
methane	nmol/kg	no
methane_l	nmol/l	no
dimethyl_sulfide	nmol/l	no
nitrogen	umol/kg	no
calcium	mmol kg-1	no

continues on next page

Table 1 – continued from previous page

netcdf varname	Units	In ERDDAP
argon	umol/kg	yes
argon_l	umol l-1	yes
dissolved_organic_carbon_14	1e-3	no
dissolved_organic_carbon_13	1e-3	no
d15n_no3	1e-3	yes
d15n_no2	1e-3	no
d15n_nh4	1e-3	no
d15n_n2o	1e-3	no
d15n_nitrite_nitrate	1e-3	yes
d15n_pon	1e-3	no
d18o_nitrite_nitrate	1e-3	yes
d18o_nitrate	1e-3	yes
d18o_nitrite	1e-3	no
d18o_nitrust_oxide	1e-3	no
urea	umol/kg	no
hplc_tot_chl_a	mg/m^3	no
hplc_tot_chl_b	mg/m^3	no
hplc_tot_chl_c	mg/m^3	no
hplc_alpha_beta_carotenes	mg/m^3	no
hplc_19butanoyloxyfucoxanthin	mg/m^3	no
hplc_19_hexanoyloxyfucoxanthin	mg/m^3	no
hplc_alloxanthin	mg/m^3	no
hpld_antheraxanthin	mg/m^3	no
hplc_diadinoxanthin	mg/m^3	no
hplc_diatoxanthin	mg/m^3	no
hplc_fucoxanthin	mg/m^3	no
hplc_peridinin	mg/m^3	no
hplc_zeaxanthin	mg/m^3	no
hplc_monovinyl_chlorophyll_a	mg/m^3	no
hplc_divinyl_chlorophyll_a	mg/m^3	no
hplc_chlorophyllide_a	mg/m^3	no
hplc_monovinyl_chlorophyll_b	mg/m^3	no
hplc_divinyl_chlorophyll_b	mg/m^3	no
hplc_chlorophyll_c1_c2	mg/m^3	no
hplc_chlorophyll_c2	mg/m^3	no
hplc_chlorophyll_c3	mg/m^3	no
hplc_lutein	mg/m^3	no
hplc_neoxanthin	mg/m^3	no
hplc_violaxanthin	mg/m^3	no
hplc_pheophytin_a	mg/m^3	no
hplc_pheophorbide_a	mg/m^3	no
hplc_prasinoxanthin	mg/m^3	no
hplc_gyroxanthin_diester	mg/m^3	no
bottle_date	None	no
bottle_time	None	no
package_depth	meters	no
odf_pressure	dbar	no
bottle_latitude	None	no
bottle_longitude	None	no

continues on next page

Table 1 – continued from previous page

netcdf varname	Units	In ERDDAP
ctd_number_of_observations	None	no
ctd_elapsed_time	seconds	no
instrument_id	None	no
ctd_sampling_rate	1/s	no
potential_temperature_c	degC	no
potential_temperature_68	degC	no
potential_temperature	degC	no
apparent_oxygen_utilization	umol/kg	no
arabinose	nmol/kg	no
bacterial_cell_count	1e8 l-1	no
cellcount	l-1	no
synechococcus_cell_count	1e6 l-1	no
picoeukaryote_cell_counts	1e6 l-1	no
prochlorophyte_cell_count	1e7 l-1	no
black_carbon	umol l-1	no
brdu_uptake	pmol l-1 h-1	no
methyl_bromide	pmol/kg	no
methyl_iodide	pmol/kg	no
dcns	nmol/kg	no
fucose	nmol/kg	no
galactose	nmol/kg	no
glucose	nmol/kg	no
mannose	nmol/kg	no
rhamnose	nmol/kg	no
density	kg m-3	no
krypton	nmol/kg	yes
krypton_l	nmol/l	yes
xenon	nmol/kg	yes
xenon_l	nmol/l	yes
pigments	None	no
reference_salinity	g/kg	no
trifluoromethylsulfur_pentafluoride	fmol/kg	no
trifluoromethylsulfur_pentafluoride_l	fmol/l	no
downcast_pressure	dbar	no
downcast_oxygen	umol/kg	no
sigma0	kg m-3	no
somma_salinity	1	no
hplc_placeholder	None	no
dna_placeholder	None	no
update_placeholder	None	no
flow_cytometry_placeholder	None	no
abundance_placeholder	None	no
stable_isotope_probing_placeholder	None	no
quota_placeholder	None	no
image_placeholder	None	no
viral_abundance_placeholder	None	no
cdom_nasa_placeholder	None	no
cdom_ucsb_placeholder	None	no
microgel_abundance	1e6 l-1	no

continues on next page

Table 1 – continued from previous page

netcdf varname	Units	In ERDDAP
n2_argon_ratio	None	no
n2_argon_ratio_unstripped	None	no
d15n_n2	1e-3	no
o2_ar	None	no
sm_depth	meters	no
fm_depth	meters	no
cyanobacteria_cell_count	ml-1	no
phytoplankton_cell_count	ml-1	no
he3_he4_ratio	None	no
nd_143_d_epsilon_bottle	1e4	no
la_d_conc_bottle	pmol/l	no
ce_d_conc_bottle	pmol/l	no
pr_d_conc_bottle	pmol/l	no
sm_d_conc_bottle	pmol/l	no
eu_d_conc_bottle	pmol/l	no
gd_d_conc_bottle	pmol/l	no
tb_d_conc_bottle	pmol/l	no
dy_d_conc_bottle	pmol/l	no
ho_d_conc_bottle	pmol/l	no
er_d_conc_bottle	pmol/l	no
tm_d_conc_bottle	pmol/l	no
yb_d_conc_bottle	pmol/l	no
lu_d_conc_bottle	pmol/l	no
user_station_number	None	no
user_sample_number	None	no
user_bottle_number	None	no
ldeo_sample_number	None	no
bnlid	None	no

CHAPTER

FOUR

BASIC CF/NETCDF OPERATORS

This is a planning/ideas document.

4.1 Overview

Manipulation of the CCHDO/ODF CF/netCDF format is needed to support data operations at sea and on shore. On shore, CCHDO performs "data merges" where data submitted by program participants is integrated into single data files. At sea, ODF is creating the initial data files and integrating onboard data similar to CCHDO. Perhaps the largest difference is that ODF must create the basic profile records while CCHDO is often doing updates of an existing record. To support both. a set of "low level" operations needs to be defined.

Here is a broad overview of what is needed:

- Initialize an "empty" dataset
- Add/Remove Profiles (N_PROF dim)
- Add/Remove vertical levels (N_LEVELS dim)
- Add/Remove Per Profile Vertical Levels (Z axis)
- Add/Remove non required variables
- Add/Remove ancillary variables
- Add/Remove variable data
- Add/Remove ancillary variable data

4.2 Initialize Empty Dataset

A function initializing an empty dataset should return an xr.Dataset with the following properties:

- Contain 2 dimensions N_PROF and N_LEVELS with their values set to 0 (this might create a netCDF4 dataset with unlimited dims)
- Include all the required variables with the correct attrs, dims, and variable dtypes.
- Sets correct global attrs (TBD)

4.3 Add/Remove Profiles

Adding a profiles requires that certain attributes about it are known before it can be created. These include:

- Expocode
- station
- cast
- longitude (X)
- latitude (Y)
- time (T)

The actual vertical level (Z), in our case pressure, is not needed at profile initialization time. A function adding a profile should require the above coordinates and append the profile information to the end. Optionally, it might sort the profiles by time. All expanded arrays should have the new "slots" filled with an appropriate fill values, nan for numeric (even flags internally), and empty string for char arrays.

Removal of a profile should remove whatever it needs such that the profile is gone. Optionally guard against deletion of non coordinate data

4.4 Add/Remove Vertical Levels

Due to the use of the incomplete multidimensional array representation of profiles (CF 9.3.2), it is valid for the Z coordinate to contain missing values as long as every other variable is missing the same data. A function that adds vertical levels therefore is one that just expands the N_LEVELS dimension and adds the appropriate fill values in the new slots. Example, it would make sense for a cruise is using a 36 place rosette to expand the N_LEVELS from 0 to 36 and not need to add any additional vertical levels for the remainder of the cruise, only adding profiles.

Removal of one or more vertical levels should ideally only be needed at the "end" of the array/profile. Optionally guard against deletion of non coordinate data.

4.5 Add/Remove Per Profile Vertical Levels

The above Add/Remove Vertical Levels only creates the space in the data structures to hold the actual vertical axis data. The use of an incomplete multidimensional array means not every profile will have the same number of vertical levels. In the CF/netCDf format, a vertical level for a profile is considered available if and only if it has a value for "sample", this is true for CTD files as well as bottle. Additionally, the vertical coordinate, pressure, must not have any fill values where there is also a "sample".

This means both "sample" and "pressure" are needed to create a valid vertical level "slot" in a profile. The block of data needs to be contiguous, i.e. it starts from the 0 position in the array and ends at the n-1 index, where n is the actual number of vertical levels of the specific profile. The Z values also need to be sorted from shallow to deep

Removal of a vertical level should probably be done by "sample". If the last vertical level is not the one being removed, the resulting array needs to be rearranged so the data are contiguous. The array shape would not change. The removal of a vertical level would need to occur in all variables that are referenced. Optionally can guard against deletion of non coordinate data

4.5.1 Possible Idea:

Since two bits of information are needed, and their data types are known, perhaps the API might be one that accepts a python dict:

```
levels = {"1": 5000, "2", 4600.3}
add_profile_level(level)
```

The add profile level function could also be the place the data are sorted.

4.6 Add/Remove non required variables

The non required variables are what most people would consider to be the actual data in the file. Things like temperature, salinity, oxygen, etc… Adding a variable is one of the most basic operations in netCDF (there is a createVariable function) and for our purposes, involves setting the correct dtype, referencing the correct dims, and getting the proper attributes set. The correct attributes depend on what the specific variable is. These should reference the cchdo.params database until we have a well defined way of dealing with "non canonical" variables.

Removing a variable need to cleanup any ancillary variables that exclusively reference the removed variable. Some optical parameters require cleanup of additional coordinate dimensions.

4.7 Add/Remove ancillary variables

Ancillary variables include flags, uncertainties, and in the case of many carbon parameters, the analytical temperature. They are created/removed the same way as the variables above, however, the "parent variable" must already exist and be updated to reference the newly created ancillary variable.

Removal of an ancillary variable must cleanup any references to that ancillary variable. There is not a one to one relation between variable and ancillary variables, e.g. a single flag variable might be referenced by multiple other variables.

4.8 Add/Remove variable data

Adding and removing data is done using the (expocode, station, cast, sample) composite keys to reference specific cells and change their values. Some variables need more coordinate information (e.g. wavelength) to get the specific cell.

Removal of variable data is done by setting the cell value to the appropriate fill values (nan or empty string) depending on variable dtype.

Optionally (perhaps by default), data changes should only be allowed where the flag ancillary variable suggests there should be values.

Variable data updates are closely tied with ancillary data updates, especially flags. We probably want this function and the next one to actually be the same function.

4.9 Add/Remove ancillary variable data

Ancillary variable data is indexed similarly to the variable data. It is listed separately here because one of the earliest data operations that occurs is setting the flags where data are expected in the future. ODF calls this "sample log entry". The flag value indicates what variables collected water for analysis and is updated when the data actually arrive. Flag updates also happen when QC is performed.

There is a situation where a problem was identified with the sampling device itself (niskin) and all water samples that came from that bottle should at least be flagged as "not good". This has not been without disagreement, since the flags for variables are supposed to be about the specific measurement and not if that measurement was done on water that makes sense. However, checking the "bottle flag" is a nuance missed on many users of the data.

CHAPTER

FIVE

CHANGELOG

5.1 v1.0.2.3 (unreleased)

- (bug) Remove the *C_format* and *C_format_source* attributes for non floating point variables. Integer and string values are exact so do not need any sort of format hint. Including a format string for non floating point values is undefined behavior in the netCDF-C Library and can result in crashing.
- (new) Add *to_coards()* and *to_woce()* accessors to maintain legacy formats at CCHDO.
- (new) All the *to_** accessors now support a path argument that will accept a writeable binary mode file like object or a filesystem path to write to.
- (new) Add a *compact_profile()* accessor that drops the trailing fill values from a profile
- (new) Add the a *file_seperator* and *keep_seperator* to *cchdo.hydro.exchange.read_exchange()*. The *keep_seperator* argument defualts to True. This is specifically to allow the reading of CTD exchange files that have been concatenated together (rather than zipped). Assuming there is nothing after "END_DATA" and you cat a bunch of _ct1.csv files together, they should be readable if "END_DATA" is passed into the *file_seperator* argument.
- (new) Add *–dump-data-counts* option to the exchange status generator which will dump a json document containing a object with nc_var name strings to count integers of how many variables with this name actually contain any data (i.e. are not just entirely fill value).
- Add a *–version* option to the cli interface
- (changed) Export *read_exchange* from the top level *cchdo.hydro* namespace.
- (changed) Bump min *cchdo.params* version to 0.1.20
- (changed) Dropped netCDF4 as requried for installation, if netCDF4 isn't installed already you can install with the *cchdo.hydro[netcdf4]* optional.
  - While this might seem like an odd choice for a library that started as one to convert WHP Exchange files to netCDf, netCDF itself is not called untill the very end of the conversion process. Internally, everything is an *xarray.Dataset*. This means you can install this library to read exchange files in tricky enviornments like pyodide or jupyterlite which already tend to have pandas and numpy in them.
- (bug) fix *pressure* variable not having a *_FillValue* attribute

5.2 v1.0.2.2 (2022-08-18)

- Support for time values that are equal to 2400, when this is encountered, the date will be set to midnight of the next day.
- *read_exchange()* will now accept bytes and bytearray objects as input, wrapping data in an *io.BytesIO* is not needed anymore.

5.3 v1.0.2.1 (2022-07-08)

- (breaking) fix misspelling of `convert_exchange` subcommand
- Will not rely on the python universal newlines for reading exchange data
- Will now combine CDOM parameters into a single variable with a new wavelength dimmension in the last axis.
- Update the WHP error name lookup to be compatable with cchdo.params v0.1.18, this is now the minimum version
- Add an `error_data` attribute to `ExchangeParameterUndefError` that will contain a list of all the unknown `(param, unit)` pairs in an exchange file when attempting to read one.
- Add an `error_data` attribute to `ExchangeDataFlagPairError` that will contain a list of all the found flag errors as an xarray.Dataset
- Automatically attempt to use BTLNBR as a fallback if SAMPNO is not present in a bottle file.
- Automatically reconstruct the date of a missing BTL_DATE param if only BTL_TIME is present.
- Add `--dump-unknown-params` option to the status_exchange subcommand which will dump an unknown param list into a json format into the `out_dir`.
- Performing a flag check is now behind a feature switch (defaults to true, for the status-exchange it is set to false)
- If a TIME column contains entirely the string "0" (not 0000) it will be ignored

5.4 v1.0.2.0 (2022-04-12)

This release includes an almost complete rewrite of how the exchange to netCDF conversion works. It now more directly uses numpy and has significant memory reduction and speed improvements when converting CTD (bottle is about the same).

- (breaking) The CLI was changed to support multiple actions which caused the exchange to netCDF functions to be moved to a sub-command "convert-exchnage" with the same interface as before.
- (breaking) The "source_C_format" attribute has been removed in favor of only having one "C_format" attribute, the "source" of the value in the C_format attribute will be listed in a new attribute "C_format_source" with the value of either "input_file" if the C_format was calculated from a text based input, or "database" if the C_format was taken from the internal database.
- (temporary) the netCDF to exchange function is not quite ready yet to work as an xarray accessor.
- (provisional) the order which netCDF variables appear is now in "exchange preferred" order.

5.4.1 Bug Fixes

- Fixed an issue where the WOCE sumfile accessor would misalign latitude columns near the equator since they lacked a digit in the tens place.
- Fixed an issue where the WOCE sumfile accessor would use "pressure levels" of CTD source netCDF files as the number of bottles.
- Fixed an issue where stations might occur in an unexpected order.

5.5 v1.0.1.3 (2021-08-25)

This release fixes many of the issues identified after the initial "1.0.0.0" release. Highlights include:

- Explicitly set the `_FillValue` attribute for the bottle closure time variable.
- The dtype for real number variables has been changed from `float` to `double`
- If the source data is an "exchange csv", a `source_C_format` attribute will (with some exceptions) be present on the real number data variables.

5.6 v1.0.1.2 (2021-03-11)

This release fixes a typo in the pyproject.toml file which would cause the _version.py file to be invalid.

5.7 v1.0.1.0 (2021-03-11)

Hopefully this fixes the errors which prevented the project from being published automatically to pypi.

5.8 v1.0.0.0 (2021-03-11)

After a whole bunch of testing, meetings, more testing, arguments, and a lot of work. We have declared the current status of the project as "good enough" for a 1.0.0 release.

There is much work to be done, especially since not all our files convert currently, but we think the ones that do convert are ready for public consumption. Unless something crazy goes wrong or is discovered, format changes should only be additive in nature (e.g. new attributes on variables).

The version will hopefully use the following (close to semver):

x.y.z

Where:

- x is incremented when a real breaking change to the netCDF output format is made.
- y is incremented when things are added to the netCDF format that should not break code which relies on previously existing attributes
- z is incremented for normal software releases that don't change the netCDF output.

Note: The version number was since updated to be w.x.y.z where w.x is the CCHDO netCDF format version and y.z is the software versions

CHAPTER

SIX

API REFERENCE

This page contains auto-generated API reference documentation[1].

6.1 hydro

6.1.1 Subpackages

hydro.exchange

Submodules

hydro.exchange.exceptions

Module Contents

exception hydro.exchange.exceptions.**ExchangeError**

Bases: ValueError

This is the base exception which all the other exceptions derive from. It is a subclass of ValueError.

exception hydro.exchange.exceptions.**ExchangeEncodingError**

Bases: *ExchangeError*

Error raised when the bytes for some exchange file cannot be decoded as UTF-8.

exception hydro.exchange.exceptions.**ExchangeBOMError**

Bases: *ExchangeError*

Error raised when the exchange file has a byte order mark.

exception hydro.exchange.exceptions.**ExchangeLEError**

Bases: *ExchangeError*

Error raised when the exchange file does not have the correct line endings.

exception hydro.exchange.exceptions.**ExchangeMagicNumberError**

Bases: *ExchangeError*

Error raised when the exchange file does not start with BOTTLE or CTD.

[1] Created with sphinx-autoapi

exception hydro.exchange.exceptions.**ExchangeEndDataError**

Bases: *ExchangeError*

Erorr raised when END_DATA cannot be found in the exchange file.

exception hydro.exchange.exceptions.**ExchangeParameterError**

Bases: *ExchangeError*

Base exception for errors related to parameters and units.

exception hydro.exchange.exceptions.**ExchangeParameterUndefError**(*error_data*)

Bases: *ExchangeParameterError*

Error raised when the library does not have a definition for a parameter/unit pair in the exchange file.

Parameters
error_data (*List[Tuple[str, Optional[str]]]*) –

exception hydro.exchange.exceptions.**ExchangeParameterUnitAlignmentError**

Bases: *ExchangeParameterError*

Error raised when there is a mismatch between the number of parameters and number of units in the exchange file.

exception hydro.exchange.exceptions.**ExchangeDuplicateParameterError**

Bases: *ExchangeParameterError*

Error raised when the same parameter/unit pair occurs more than once in the excahnge file.

exception hydro.exchange.exceptions.**ExchangeOrphanFlagError**

Bases: *ExchangeParameterError*

Error raised when there exists a flag column with no corresponding parameter column.

exception hydro.exchange.exceptions.**ExchangeOrphanErrorError**

Bases: *ExchangeParameterError*

Error raised when there exists an error column with no corresponding parameter column.

exception hydro.exchange.exceptions.**ExchangeFlaglessParameterError**

Bases: *ExchangeParameterError*

Error raised when a parameter has a flag column when it is not supposed to.

exception hydro.exchange.exceptions.**ExchangeFlagUnitError**

Bases: *ExchangeParameterError*

Error raised if a flag column has a non empty units.

exception hydro.exchange.exceptions.**ExchangeDataError**

Bases: *ExchangeError*

Base exception for errors which occur when parsing the data porition of an exchange file.

exception hydro.exchange.exceptions.**ExchangeDataColumnAlignmentError**

Bases: *ExchangeDataError*

Error raised when the number of columns in a data line does not match the expected number of columns based on the parameter/unit lines.

exception hydro.exchange.exceptions.**ExchangeDataFlagPairError**(*error_data*)

Bases: *ExchangeDataError*

There is a mismatch between what the flag value expects, and the fill/data value.

Examples:

- something with a flag of 9 has a non fill value
- something with a flag of 2 as a fill value instead of data

Parameters
error_data (*xarray.Dataset*) –

exception hydro.exchange.exceptions.**ExchangeDataPartialKeyError**

Bases: *ExchangeDataError*

Error raised when there is no value for one (or more) of the following parameters.

- EXPOCODE
- STNNBR
- CASTNO
- SAMPNO (only for bottle files)
- CTDPRS (only for CTD files)

These form the "composite key" which uniquely identify the "row" of exchange data.

exception hydro.exchange.exceptions.**ExchangeDuplicateKeyError**

Bases: *ExchangeDataError*

Error raised when there is a duplicate composite key in the exchange file.

This would occur if the exact values for the following parameters occur in more than one data row:

- EXPOCODE
- STNNBR
- CASTNO
- SAMPNO (only for bottle files)
- CTDPRS (only for CTD files)

exception hydro.exchange.exceptions.**ExchangeDataPartialCoordinateError**

Bases: *ExchangeDataError*

Error raised if values for latitude, longitude, or pressure are missing.

It is OK by the standard to omit the time of day.

exception hydro.exchange.exceptions.**ExchangeDataInconsistentCoordinateError**

Bases: *ExchangeDataError*

Error raised if the reported latitude, longitude, and date (and time) vary for a single profile.

A "profile" in an exchange file is a grouping of data rows which all have the same EXPOCODE, STNNBR, and CASTNO. The SAMPNO/CTDPRS is allowed/requried to vary for a single profile and is what identifies samples within one profile.

exception hydro.exchange.exceptions.**ExchangeInconsistentMergeType**

Bases: *ExchangeError*

Error raised when the merge_ex method is called on mixed ctd and bottle exchange types

exception hydro.exchange.exceptions.**ExchangeRecursiveZip**

Bases: *ExchangeError*

Error raised if there are zip files inside the zip file that read exchange is trying to read

hydro.exchange.flags

A Collection of Flag Schemes

Module Contents

Classes

ExchangeBottleFlag	Enum representing a WHP Bottle flag
ExchangeSampleFlag	Enum where members are also (and must be) ints
ExchangeCTDFlag	Enum where members are also (and must be) ints

class hydro.exchange.flags.**ExchangeBottleFlag**(*flag*)

Bases: enum.IntEnum

Enum representing a WHP Bottle flag

This flag represents information about the sampling device itself (i.e. the niskin bottle). It should only be used for "BTLNBR_FLAG_W" values and should never be used with CTD files.

property definition

Prints the human readable flag definition

property cf_def

property has_value

Should the data this is a flag for contain a value

NOFLAG = 0

NO_INFO = 1

GOOD = 2

LEAKING = 3

BAD_TRIP = 4

NOT_REPORTED = 5

DISCREPANCY = 6

UNKNOWN = 7

PAIR = 8

NOT_SAMPLED = 9

class hydro.exchange.flags.**ExchangeSampleFlag**(*flag*)

Bases: enum.IntEnum

Enum where members are also (and must be) ints

property definition

property cf_def

property has_value

NOFLAG = 0

MISSING = 1

GOOD = 2

QUESTIONABLE = 3

BAD = 4

NOT_REPORTED = 5

MEAN = 6

CHROMA_MANUAL = 7

CHROMA_IRREGULAR = 8

NOT_SAMPLED = 9

class hydro.exchange.flags.**ExchangeCTDFlag**(*flag*)

Bases: enum.IntEnum

Enum where members are also (and must be) ints

property definition

property cf_def

property has_value

NOFLAG = 0

UNCALIBRATED = 1

GOOD = 2

QUESTIONABLE = 3

BAD = 4

NOT_REPORTED = 5

INTERPOLATED = 6

DESPIKED = 7

NOT_SAMPLED = 9

hydro.exchange.helpers

Module Contents

Functions

simple_bottle_exchange([params, units, data, comments])	
gen_complete_bottle([ctd_params_only, param_counts, ...])	

hydro.exchange.helpers.**simple_bottle_exchange**(*params=None*, *units=None*, *data=None*, *comments=None*)

Parameters
comments (*str*) –

hydro.exchange.helpers.**gen_complete_bottle**(*ctd_params_only=False*, *param_counts=None*, *min_count=5*, *filter_erddap=False*)

Parameters
param_counts (*Optional[Dict[str, int]]*) –

Package Contents

Classes

ExchangeBottleFlag	Enum representing a WHP Bottle flag
ExchangeCTDFlag	Enum where members are also (and must be) ints
ExchangeSampleFlag	Enum where members are also (and must be) ints
FileType	Generic enumeration.
_ExchangeData	Dataclass containing exchange data which has been parsed into ndarrays
_ExchangeInfo	Low level dataclass containing the parts of an exchange file
CheckOptions	A simple typed namespace. At runtime it is equivalent to a plain dict.

Functions

_bottle_get_params(params_units)	Given an ordered iterable of param, unit pairs, return the index of the column in the datafile for known WHP params.
_bottle_get_flags(params_units, whp_params)	Given an ordered iterable of param unit pairs and WHPNames known to be in the file, return the index of the column indicies of the flags for the WHPNames.
_bottle_get_errors(params_units, whp_params)	Given an ordered iterable of param unit pairs and WHPNames known to be in the file, return the index of the column indicies of the errors/uncertanties for the WHPNames.
_ctd_get_header(line[, dtype])	
_is_all_dataarray(val)	
flatten_cdom_coordinate(dataset)	Takes the a dataset with a CDOM wavelength and explocdes it back into individual variables
add_cdom_coordinate(dataset)	Find all the paraters in the cdom group and add their wavelength in a new coordinate
add_geometry_var(dataset)	Adds a CF-1.8 Geometry container variable to the dataset
add_profile_type(dataset, ftype)	Adds a *profile_type* string variable to the dataset.
finalize_ancillary_variables(dataset)	Turn the ancillary variable attr into a space seperated string
combine_bottle_time(dataset)	Combine the bottle dates and times if present
check_is_subset_shape(a1, a2[, strict])	Ensure that the shape of the data in a2 is a subset (or strict subset) of the data shape of a1
check_flags(dataset[, raises])	Check WOCE flag values agaisnt their param and ensure that the param either has a value or is "nan"
_get_fill_locs(arr[, fill_values])	
_extract_numeric_precisions(data)	Get the numeric precision of a printed decimal number
_is_valid_exchange_numeric(data)	
_combine_dt_ndarray(date_arr[, time_arr, time_pad])	
sort_ds(dataset)	Sorts the data values in the dataset
check_sorted(dataset)	Check that the dataset is sorted by the rules in sort_ds()
combine_dt(dataset[, is_coord, date_name, time_name, ...])	Combine the exchange style string variables of date and optinally time into a single
set_axis_attrs(dataset)	Set the CF axis attribute on our axis variables (XYZT)
set_coordinate_encoding_fill(dataset)	Sets the _FillValue encoidng to None for 1D coordinate vars
_load_raw_exchange(filename_or_obj, *[, ...])	
all_same(ndarr)	Test if all the values of an ndarray are the same value
read_exchange(filename_or_obj, *[, fill_values, ...])	Loads the data from filename_or_obj and returns a xr.Dataset with the CCHDO

Attributes

CCHDO_VERSION
log
DIMS
EXPOCODE
STNNBR
CASTNO
SAMPNO
DATE
TIME
LATITUDE
LONGITUDE
CTDPRS
BTLNBR
COORDS
FLAG_SCHEME
GEOMETRY_VARS
FILLS_MAP
WHPNameIndex
WHPParamUnit
ExchangeIO
WHPNameAttr

exception hydro.exchange.**ExchangeDataFlagPairError**(*error_data*)

Bases: ExchangeDataError

There is a mismatch between what the flag value expects, and the fill/data value.

Examples:

- something with a flag of 9 has a non fill value

- something with a flag of 2 as a fill value instead of data

Parameters
error_data (*xarray.Dataset*) –

exception hydro.exchange.**ExchangeDataInconsistentCoordinateError**

Bases: ExchangeDataError

Error raised if the reported latitude, longitude, and date (and time) vary for a single profile.

A "profile" in an exchange file is a grouping of data rows which all have the same EXPOCODE, STNNBR, and CASTNO. The SAMPNO/CTDPRS is allowed/requried to vary for a single profile and is what identifies samples within one profile.

exception hydro.exchange.**ExchangeDataPartialCoordinateError**

Bases: ExchangeDataError

Error raised if values for latitude, longitude, or pressure are missing.

It is OK by the standard to omit the time of day.

exception hydro.exchange.**ExchangeDataPartialKeyError**

Bases: ExchangeDataError

Error raised when there is no value for one (or more) of the following parameters.

- EXPOCODE
- STNNBR
- CASTNO
- SAMPNO (only for bottle files)
- CTDPRS (only for CTD files)

These form the "composite key" which uniquely identify the "row" of exchange data.

exception hydro.exchange.**ExchangeDuplicateKeyError**

Bases: ExchangeDataError

Error raised when there is a duplicate composite key in the exchange file.

This would occur if the exact values for the following parameters occur in more than one data row:

- EXPOCODE
- STNNBR
- CASTNO
- SAMPNO (only for bottle files)
- CTDPRS (only for CTD files)

exception hydro.exchange.**ExchangeEncodingError**

Bases: *ExchangeError*

Error raised when the bytes for some exchange file cannot be decoded as UTF-8.

exception hydro.exchange.**ExchangeBOMError**

Bases: *ExchangeError*

Error raised when the exchange file has a byte order mark.

exception hydro.exchange.**ExchangeError**

Bases: ValueError

This is the base exception which all the other exceptions derive from. It is a subclass of ValueError.

exception hydro.exchange.**ExchangeInconsistentMergeType**

Bases: *ExchangeError*

Error raised when the merge_ex method is called on mixed ctd and bottle exchange types

exception hydro.exchange.**ExchangeMagicNumberError**

Bases: *ExchangeError*

Error raised when the exchange file does not start with BOTTLE or CTD.

exception hydro.exchange.**ExchangeDuplicateParameterError**

Bases: ExchangeParameterError

Error raised when the same parameter/unit pair occurs more than once in the excahnge file.

exception hydro.exchange.**ExchangeParameterUnitAlignmentError**

Bases: ExchangeParameterError

Error raised when there is a mismatch between the number of parameters and number of units in the exchange file.

exception hydro.exchange.**ExchangeOrphanFlagError**

Bases: ExchangeParameterError

Error raised when there exists a flag column with no corresponding parameter column.

exception hydro.exchange.**ExchangeOrphanErrorError**

Bases: ExchangeParameterError

Error raised when there exists an error column with no corresponding parameter column.

exception hydro.exchange.**ExchangeParameterUndefError**(*error_data*)

Bases: ExchangeParameterError

Error raised when the library does not have a definition for a parameter/unit pair in the exchange file.

Parameters
error_data (*List[Tuple[str, Optional[str]]]*) –

exception hydro.exchange.**ExchangeFlaglessParameterError**

Bases: ExchangeParameterError

Error raised when a parameter has a flag column when it is not supposed to.

exception hydro.exchange.**ExchangeFlagUnitError**

Bases: ExchangeParameterError

Error raised if a flag column has a non empty units.

class hydro.exchange.**ExchangeBottleFlag**(*flag*)

Bases: enum.IntEnum

Enum representing a WHP Bottle flag

This flag represents information about the sampling device itself (i.e. the niskin bottle). It should only be used for "BTLNBR_FLAG_W" values and should never be used with CTD files.

property definition

Prints the human readable flag definition

property cf_def

property has_value

Should the data this is a flag for contain a value

NOFLAG = 0

NO_INFO = 1

GOOD = 2

LEAKING = 3

BAD_TRIP = 4

NOT_REPORTED = 5

DISCREPANCY = 6

UNKNOWN = 7

PAIR = 8

NOT_SAMPLED = 9

class hydro.exchange.**ExchangeCTDFlag**(*flag*)

Bases: enum.IntEnum

Enum where members are also (and must be) ints

property definition

property cf_def

property has_value

NOFLAG = 0

UNCALIBRATED = 1

GOOD = 2

QUESTIONABLE = 3

BAD = 4

NOT_REPORTED = 5

INTERPOLATED = 6

DESPIKED = 7

NOT_SAMPLED = 9

class hydro.exchange.**ExchangeSampleFlag**(*flag*)

Bases: enum.IntEnum

Enum where members are also (and must be) ints

property definition

property cf_def

property has_value

NOFLAG = 0

MISSING = 1

GOOD = 2

QUESTIONABLE = 3

BAD = 4

NOT_REPORTED = 5

MEAN = 6

CHROMA_MANUAL = 7

CHROMA_IRREGULAR = 8

NOT_SAMPLED = 9

hydro.exchange.CCHDO_VERSION

hydro.exchange.log

hydro.exchange.DIMS = ('N_PROF', 'N_LEVELS')

hydro.exchange.EXPOCODE

hydro.exchange.STNNBR

hydro.exchange.CASTNO

hydro.exchange.SAMPNO

hydro.exchange.DATE

hydro.exchange.TIME

hydro.exchange.LATITUDE

hydro.exchange.LONGITUDE

hydro.exchange.CTDPRS

hydro.exchange.BTLNBR

hydro.exchange.COORDS

hydro.exchange.FLAG_SCHEME

hydro.exchange.GEOMETRY_VARS = ('expocode', 'station', 'cast', 'section_id', 'time')

hydro.exchange.FILLS_MAP

class hydro.exchange.**FileType**

Bases: enum.Enum

Generic enumeration.

Derive from this class to define new enumerations.

CTD = 'C'

BOTTLE = 'B'

hydro.exchange.**WHPNameIndex**

hydro.exchange.**WHPParamUnit**

hydro.exchange.**_bottle_get_params**(*params_units*)

Given an ordered iterable of param, unit pairs, return the index of the column in the datafile for known WHP params.

Exchange files have comma separated parameter names on one line, and the corresponding units on the next. This function will search for this name+unit pair in the builtin database of known WHP parameter names and return a mapping of `WHPName` to column indicies.

It is currently an error for the parameter in a file to not be in the built in database.

This function will ignore uncertainty (error) columns and flag columns, those are parsed by other functions.

Warning: Convert semantically empty units (e.g. empty string, all whitespace) to None before passing into this function

Note: The parameter name database will convert unambiguous aliases to their canonical exchange parameter and unit pair.

Parameters

params_units (*tuple in the form (str, str) or (str, None)*) – Paired (e.g. zip) parameter names and units

Returns

Mapping of `WHPName` to column indicies

Return type

dict with keys of `WHPName` and values of int

Raises

ExchangeParameterUndefError – if the parameter unit pair cannot be found in the built in database

hydro.exchange.**_bottle_get_flags**(*params_units*, *whp_params*)

Given an ordered iterable of param unit pairs and WHPNames known to be in the file, return the index of the column indicies of the flags for the WHPNames.

Exchange files can have status flags for some of the parameters. Flag columns must have no units. Some parameters must not have status flags, these include the spatiotemporal parameters (e.g. lat, lon, but also pressure) and the sample identifying parameters (expocode, station, cast, sample, but *not* bottle id).

Parameters

- **params_units** (*tuple in the form (str, str) or (str, None)*) – Paired (e.g. zip) parameter names and units
- **whp_params** (Mapping of WHPName to int) – Mapping of parameters known to be in the file, this is the output of *_bottle_get_params()*

Returns
Mapping of WHPName to column indicies for the status flag column

Return type
dict with keys of WHPName and values of int

Raises

- ***ExchangeFlagUnitError*** – if the flag column has units other than None
- ***ExchangeFlaglessParameterError*** – if the flag column is for a parameter not allowed to have status flags
- ***ExchangeOrphanFlagError*** – if the flag column is for a parameter not in the passed in mapping of whp_params

hydro.exchange.**_bottle_get_errors**(*params_units*, *whp_params*)

Given an ordered iterable of param unit pairs and WHPNames known to be in the file, return the index of the column indicies of the errors/uncertanties for the WHPNames.

Some parameters may have uncertanties associated with them, this function finds those columns and pairs them with the correct parameter.

Note: There is no programable way to find the error columns for a given unit (e.g. no common suffix like the flags). This must be done via lookup in the built in database of params.

Parameters

- **params_units** (*tuple in the form (str, str) or (str, None)*) – Paired (e.g. zip()) parameter names and units
- **whp_params** (Mapping of WHPName to int) – Mapping of parameters known to be in the file, this is the output of *_bottle_get_params()*

Returns
Mapping of WHPName to column indicies for the error column

Return type
dict with keys of WHPName and values of int

hydro.exchange.**_ctd_get_header**(*line*, *dtype=str*)

hydro.exchange.**_is_all_dataarray**(*val*)

Parameters
val (*List[Any]*) –

Return type
typing_extensions.TypeGuard[List[xarray.DataArray]]

hydro.exchange.**flatten_cdom_coordinate**(*dataset*)

Takes the a dataset with a CDOM wavelength and explocdes it back into individual variables

Parameters
dataset (*xarray.Dataset*) –

Return type
xarray.Dataset

hydro.exchange.**add_cdom_coordinate**(*dataset*)

Find all the paraters in the cdom group and add their wavelength in a new coordinate

Parameters
dataset (*xarray.Dataset*) –

Return type
xarray.Dataset

hydro.exchange.**add_geometry_var**(*dataset*)

Adds a CF-1.8 Geometry container variable to the dataset

This allows for compatabiltiy with tools like gdal

Parameters
dataset (*xarray.Dataset*) –

Return type
xarray.Dataset

hydro.exchange.**add_profile_type**(*dataset*, *ftype*)

Adds a *profile_type* string variable to the dataset.

This is for ODV compatability

> **Warning:** Currently mixed profile types are not supported

Parameters

- **dataset** (*xarray.Dataset*) –
- **ftype** (FileType) –

Return type
xarray.Dataset

hydro.exchange.**finalize_ancillary_variables**(*dataset*)

Turn the ancillary variable attr into a space seperated string

It is nice to have the ancillary variable be a list while things are being read into it

Parameters
dataset (*xarray.Dataset*) –

hydro.exchange.**combine_bottle_time**(*dataset*)

Combine the bottle dates and times if present

Raises if only one is present

Parameters
dataset (*xarray.Dataset*) –

hydro.exchange.**check_is_subset_shape**(*a1*, *a2*, *strict='disallowed'*)

Ensure that the shape of the data in a2 is a subset (or strict subset) of the data shape of a1

For a given set of param, flag, and error arrays you would want to ensure that: * errors are a subset of params (strict is allowed) * params are a subset of flags (strict is allowed)

For string vars, the empty string is considered the "nothing" value. For woce flags, flag 9s should be converted to nans (depending on scheme flag 5 and 1 may not have param values)

Return a boolean array of invalid locations

Parameters

- **a1** (*numpy.typing.NDArray*) –
- **a2** (*numpy.typing.NDArray*) –

Return type
numpy.typing.NDArray[numpy.bool_]

hydro.exchange.**check_flags**(*dataset*, *raises=True*)

Check WOCE flag values agaisnt their param and ensure that the param either has a value or is "nan" depedning on the flag definition.

Return a boolean array of invalid locations?

Parameters
dataset (*xarray.Dataset*) –

class hydro.exchange.**_ExchangeData**

Dataclass containing exchange data which has been parsed into ndarrays

single_profile: bool

param_cols: Dict[cchdo.params.WHPName, numpy.ndarray]

flag_cols: Dict[cchdo.params.WHPName, numpy.ndarray]

error_cols: Dict[cchdo.params.WHPName, numpy.ndarray]

param_precisions: Dict[cchdo.params.WHPName, numpy.typing.NDArray[numpy.int_]]

error_precisions: Dict[cchdo.params.WHPName, numpy.typing.NDArray[numpy.int_]]

comments: str

__post_init__()

set_expected(*params*, *flags*, *errors*)

Puts fill columns for expected params which are missing

This can occur when there are disjoint columns in CTD files

Parameters

- **params** (*Set[cchdo.params.WHPName]*) –
- **flags** (*Set[cchdo.params.WHPName]*) –
- **errors** (*Set[cchdo.params.WHPName]*) –

split_profiles()

Split into single profile containing _ExchangeData instances

Done by looking at the expocode+station+cast composate keys

str_lens()

Figure out the length of all the string params

The char size can vary by platform.

Return type
Dict[cchdo.params.WHPName, int]

hydro.exchange.**_get_fill_locs**(*arr*, *fill_values=('-999',)*)

Parameters
fill_values (*Tuple[str, Ellipsis]*) –

class hydro.exchange.**_ExchangeInfo**

Low level dataclass containing the parts of an exchange file

property stamp

Returns the filestamp of the exchange file

e.g. "BOTTLE,20210301CCHSIOAMB"

property comments

Returns the comments of the exchange file with leading # stripped

property ctd_headers

Returns a dict of the CTD headers and their value

property data

Returns the data block of an exchange file as a tuple of strs. One line per entry.

property post_data

Returns any post data content as a tuple of strs

property _np_data_block

stamp_slice: slice

comments_slice: slice

ctd_headers_slice: slice

params_idx: int

units_idx: int

data_slice: slice

post_data_slice: slice

_raw_lines: Tuple[str, Ellipsis]

params()

Returns a list of all parameters in the file (including CTD "headers")

units()

Returns a list of all the units in the file (including CTD "headers")

Will have the same shape as params

whp_params()

Parses the params and units for base parameters

Returns a dict with a WHPName to column index mapping

whp_flags()

Parses the params and units for flag values

returns a dict with a WHPName to column index of flags mapping

whp_errors()

Parses the params and units for uncertanty values

returns a dict with a WHPName to column index of errors mapping

finalize(*fill_values=('-999',)*, *precision_source='file'*)

Parse all the data into ndarrays of the correct dtype and shape

Returns an ExchangeData dataclass

Return type
_ExchangeData

classmethod from_lines(*lines*, *ftype*)

Figure out the line numbers/indicies of the parts of the exchange file

Parameters

- **lines** (*Tuple[str, Ellipsis]*) –
- **ftype** (FileType) –

hydro.exchange.**_extract_numeric_precisions**(*data*)

Get the numeric precision of a printed decimal number

Parameters
data (*numpy.typing.NDArray[numpy.str_]*) –

Return type
numpy.typing.NDArray[numpy.int_]

hydro.exchange.**_is_valid_exchange_numeric**(*data*)

Parameters
data (*numpy.typing.NDArray[numpy.str_]*) –

Return type
numpy.bool_

hydro.exchange.**ExchangeIO**

hydro.exchange.**_combine_dt_ndarray**(*date_arr*, *time_arr=None*, *time_pad=False*)

Parameters

- **date_arr** (*numpy.typing.NDArray[numpy.str_]*) –
- **time_arr** (*Optional[numpy.typing.NDArray[numpy.str_]]*) –

Return type
numpy.ndarray

hydro.exchange.**sort_ds**(*dataset*)

Sorts the data values in the dataset

Ensures that profiles are in the following order: * Earlier before later (time will increase) * Southerly before northerly (latitude will increase) * Westerly before easterly (longitude will increase)

The two xy sorts are esentially tie breakers for when we are missing "time"

Inside profiles: * Shallower before Deeper (pressure will increase)

Parameters
dataset (*xarray.Dataset*) –

Return type
xarray.Dataset

hydro.exchange.**check_sorted**(*dataset*)

Check that the dataset is sorted by the rules in *sort_ds()*

Parameters
dataset (*xarray.Dataset*) –

Return type
bool

hydro.exchange.**WHPNameAttr**

hydro.exchange.**combine_dt**(*dataset*, *is_coord=True*, *date_name=DATE*, *time_name=TIME*, *time_pad=False*)

Combine the exchange style string variables of date and optinally time into a single variable containing real datetime objects

This will remove the time variable if present, and replace then rename the date variable. Date is replaced/renamed to maintain variable order in the xr.DataSet

Parameters

- **dataset** (*xarray.Dataset*) –
- **is_coord** (*bool*) –
- **date_name** (*cchdo.params.WHPName*) –
- **time_name** (*cchdo.params.WHPName*) –

Return type
xarray.Dataset

hydro.exchange.**set_axis_attrs**(*dataset*)

Set the CF axis attribute on our axis variables (XYZT)

- longitude = "X"
- latitude = "Y"
- pressure = "Z", addtionally, positive is down
- time = "T"

Parameters
dataset (*xarray.Dataset*) –

Return type
xarray.Dataset

hydro.exchange.**set_coordinate_encoding_fill**(*dataset*)

Sets the _FillValue encoidng to None for 1D coordinate vars

Parameters
dataset (*xarray.Dataset*) –

Return type
xarray.Dataset

hydro.exchange.**_load_raw_exchange**(*filename_or_obj*, *, *file_seperator=None*, *keep_seperator=True*)

Parameters

- **filename_or_obj** (*ExchangeIO*) –
- **file_seperator** (*Optional[str]*) –

Return type
List[str]

hydro.exchange.**all_same**(*ndarr*)

Test if all the values of an ndarray are the same value

Parameters
ndarr (*numpy.ndarray*) –

Return type
numpy.bool_

class hydro.exchange.**CheckOptions**

Bases: TypedDict

A simple typed namespace. At runtime it is equivalent to a plain dict.

TypedDict creates a dictionary type that expects all of its instances to have a certain set of keys, where each key is associated with a value of a consistent type. This expectation is not checked at runtime but is only enforced by type checkers. Usage:

```
class Point2D(TypedDict):
    x: int
    y: int
    label: str

a: Point2D = {'x': 1, 'y': 2, 'label': 'good'}  # OK
b: Point2D = {'z': 3, 'label': 'bad'}           # Fails type check

assert Point2D(x=1, y=2, label='first') == dict(x=1, y=2, label='first')
```

The type info can be accessed via Point2D.__annotations__. TypedDict supports two additional equivalent forms:

```
Point2D = TypedDict('Point2D', x=int, y=int, label=str)
Point2D = TypedDict('Point2D', {'x': int, 'y': int, 'label': str})
```

By default, all keys must be present in a TypedDict. It is possible to override this by specifying totality. Usage:

```
class point2D(TypedDict, total=False):
    x: int
    y: int
```

This means that a point2D TypedDict can have any of the keys omitted.A type checker is only expected to support a literal False or True as the value of the total argument. True is the default, and makes all items defined in the class body be required.

The class syntax is only supported in Python 3.6+, while two other syntax forms work for Python 2.7 and 3.2+

flags: bool

hydro.exchange.**read_exchange**(*filename_or_obj*, ***, *fill_values=('-999',)*, *checks=None*, *precision_source='file'*, *file_seperator=None*, *keep_seperator=True*)

Loads the data from filename_or_obj and returns a xr.Dataset with the CCHDO CF/netCDF structure

Parameters

- **filename_or_obj** (*ExchangeIO*) –
- **checks** (*Optional[CheckOptions]*) –

Return type
xarray.Dataset

hydro.legacy

Subpackages

hydro.legacy.coards

Legacy COARDS netcdf make from libcchdo ported to take a CCHDO CF/netCDF xarray.Dataset object as input

The goal is, as much as possible, to use the old code with minimal changes such that the following outputs are identical:

- Exchange -> CF/netCDF -> COARDS netCDF (this library)
- Exchange -> COARDS netCDF (using libcchdo)

The entrypoint function is *to_coards()*

Package Contents

Functions

strftime_woce_date_time(dt)	Take an xr.DataArray with time values in it and convert to strings
_ascii(x)	Force all codepoints into valid ascii range
simplest_str(s)	Give the simplest string representation.
_pad_station_cast(x)	Pad a station or cast identifier out to 5 characters.
get_filename(expocode, station, cast, extension)	Generate the filename for COARDS netCDF files
minutes_since_epoch(dt, epoch[, error])	Make the time value for netCDF files
define_dimensions(nc_file, length)	Create NetCDF file dimensions.
define_attributes(nc_file, expocode, sect_id, ...)	Sets the global attributes of the input nc_file as a side effect
set_original_header(nc_file, ds)	Sets the ORIGINAL_HEADER global attribute to whatever is in ds.attrs["comments"]
create_common_variables(nc_file, latitude, longitude, ...)	Add variables to the netcdf file object such as date, time etc.
create_and_fill_data_variables(nc_file, ds)	Add variables to the netcdf file object that correspond to data.
_create_common_variables(nc_file, ds)	Extracts the latitude, longitude, station, and cast from ds and passes them to create_common_variables
write_ctd(ds)	How to write a CTD NetCDF file.
write_bottle(ds)	How to write a Bottle NetCDF file.
to_coards(ds)	Convert an xr.Dataset to a zipfile with COARDS netCDF files inside

Attributes

log	logger object for message logging
PARAMS	mapping of whp names to nc names
CTD_ZIP_FILE_EXTENSION	Filename extention for a zipped collection ctd coards netcdf files
BOTTLE_ZIP_FILE_EXTENSION	Filename extention for a zipped collection bottle coards netcdf files
FILL_VALUE	Const from old libcchdo, -999.0
QC_SUFFIX	Variable name suffix for flag variables
FILE_EXTENSION	filenmae extention for all netcdf files
EPOCH	dateime referenced in the units of time variables in netCDF files: 1980-01-01
STATIC_PARAMETERS_PER_CAST	List of WHP names that are ignored when calling *create_and_fill_data_variables()*
NON_FLOAT_PARAMETERS	params not in *STATIC_PARAMETERS_PER_CAST* that are also ignored by *create_and_fill_data_variables()*
UNKNOWN	Value used when some string value isn't found
UNSPECIFIED_UNITS	Value used when there are no units
STRLEN	length of char array variables, hardcoded to 40

hydro.legacy.coards.**log**
logger object for message logging

hydro.legacy.coards.**PARAMS**
mapping of whp names to nc names

This is loaded at module import time from a dump from the old internal params sqlite database

hydro.legacy.coards.**CTD_ZIP_FILE_EXTENSION = 'nc_ctd.zip'**
Filename extention for a zipped collection ctd coards netcdf files

hydro.legacy.coards.**BOTTLE_ZIP_FILE_EXTENSION = 'nc_hyd.zip'**
Filename extention for a zipped collection bottle coards netcdf files

hydro.legacy.coards.**FILL_VALUE**
Const from old libcchdo, -999.0

hydro.legacy.coards.**QC_SUFFIX = '_QC'**
Variable name suffix for flag variables

hydro.legacy.coards.**FILE_EXTENSION = 'nc'**
filenmae extention for all netcdf files

hydro.legacy.coards.**EPOCH**
dateime referenced in the units of time variables in netCDF files: 1980-01-01

hydro.legacy.coards.**STATIC_PARAMETERS_PER_CAST = ('EXPOCODE', 'SECT_ID', 'STNNBR', 'CASTNO', '_DATETIME', 'LATITUDE', 'LONGITUDE', 'DEPTH',...**
List of WHP names that are ignored when calling *create_and_fill_data_variables()*

hydro.legacy.coards.**NON_FLOAT_PARAMETERS = ('CTDNOBS',)**
params not in *STATIC_PARAMETERS_PER_CAST* that are also ignored by *create_and_fill_data_variables()*

hydro.legacy.coards.**UNKNOWN = 'UNKNOWN'**
Value used when some string value isn't found

This is mmostly mitigated by the guarantees of the new CF format, but e.g. section id might be missing

hydro.legacy.coards.**UNSPECIFIED_UNITS = 'unspecified'**
Value used when there are no units

hydro.legacy.coards.**STRLEN = 40**
length of char array variables, hardcoded to 40

hydro.legacy.coards.**strftime_woce_date_time**(*dt*)
Take an xr.DataArray with time values in it and convert to strings

Parameters
dt (*xarray.DataArray*) –

hydro.legacy.coards.**_ascii**(*x*)
Force all codepoints into valid ascii range

Works by encoding the str into ascii bytes with the replace err param, then decoding the bytes to str again

Parameters
x (*str*) – string with any unicode codepoint in it

Returns
string with all non ascii codepoints replaced with whatever "replace" does in `str.encode()`

Return type
str

hydro.legacy.coards.**simplest_str**(*s*)

Give the simplest string representation.

If a float is almost equivalent to an integer, swap out for the integer.

Return type
str

hydro.legacy.coards.**_pad_station_cast**(*x*)

Pad a station or cast identifier out to 5 characters.

This is usually for use in a file name.

Parameters
x (*str*) – a string to be padded

Return type
str

hydro.legacy.coards.**get_filename**(*expocode*, *station*, *cast*, *extension*)

Generate the filename for COARDS netCDF files

Was ported directly from libcchdo and should have the same formatting behavior

hydro.legacy.coards.**minutes_since_epoch**(*dt*, *epoch*, *error=-9*)

Make the time value for netCDF files

The custom implimentation in libcchdo was discarded in favor of the date2num function from cftime. Not sure if cftime exsited in the netCDF4 python library at the time.

Parameters
dt (*xarray.DataArray*) –

hydro.legacy.coards.**define_dimensions**(*nc_file*, *length*)

Create NetCDF file dimensions.

This creates all the COARDS dimensions in the input nc_file as a side effect (does not return) Dimensions created are:

- time
- pressure
- latitude
- longitude
- string_dimension

Parameters

- **nc_file** (*netCDF4.Dataset*) –
- **length** (*int*) –

hydro.legacy.coards.**define_attributes**(*nc_file*, *expocode*, *sect_id*, *data_type*, *stnnbr*, *castno*, *bottom_depth*)

Sets the global attributes of the input nc_file as a side effect

Parameters
nc_file (*netCDF4.Dataset*) –

hydro.legacy.coards.**set_original_header**(*nc_file*, *ds*)

Sets the ORIGINAL_HEADER global attribute to whatever is in ds.attrs["comments"]

Parameters

- **nc_file** (*netCDF4.Dataset*) –
- **ds** (*xarray.Dataset*) –

hydro.legacy.coards.**create_common_variables**(*nc_file*, *latitude*, *longitude*, *woce_datetime*, *stnnbr*, *castno*)

Add variables to the netcdf file object such as date, time etc.

Parameters

- **nc_file** (*netCDF4.Dataset*) –
- **latitude** (*float*) –
- **longitude** (*float*) –

hydro.legacy.coards.**create_and_fill_data_variables**(*nc_file*, *ds*)

Add variables to the netcdf file object that correspond to data.

Parameters
ds (*xarray.Dataset*) –

hydro.legacy.coards.**_create_common_variables**(*nc_file*, *ds*)

Extracts the latitude, longitude, station, and cast from ds and passes them to create_common_variables

This logic could eventually just move to create_common_variables as it was previously rather complicated

Parameters

- **nc_file** (*netCDF4.Dataset*) –
- **ds** (*xarray.Dataset*) –

hydro.legacy.coards.**write_ctd**(*ds*)

How to write a CTD NetCDF file.

Parameters
ds (*xarray.Dataset*) –

Return type
bytes

hydro.legacy.coards.**write_bottle**(*ds*)

How to write a Bottle NetCDF file.

Parameters
ds (*xarray.Dataset*) – CCHDO CF/netCDF xarray dataset containing only a single bottle profile

Returns
the bytes of a netCDF3 COARDS file

Return type
bytes

hydro.legacy.coards.**to_coards**(*ds*)

Convert an xr.Dataset to a zipfile with COARDS netCDF files inside

This function does support mixed CTD and Bottle datasets and will convert using profile_type var on a per profile basis.

Parameters
ds (*xarray.Dataset*) – A dataset conforming to CCHDO CF/netCDF

Returns
a zipfile with one or more COARDS netCDF files as members.

Return type
bytes

hydro.legacy.woce

Package Contents

Functions

flag_description(flag_map)	
simplest_str(s)	Give the simplest string representation.
_pad_station_cast(x)	Pad a station or cast identifier out to 5 characters. This is usually
get_filename(expocode, station, cast, file_ext)	
convert_fortran_format_to_c(ffmt)	Simplistic conversion from Fortran format string to C format string.
get_exwoce_params()	Return a dictionary of WOCE parameters allowed for Exchange conversion.
writeable_columns(ds[, is_ctd])	Return the columns that belong in a WOCE data file.
columns_and_base_format(dfile[, is_ctd])	Return columns and base format for WOCE fixed column data.
truncate_row(lll)	Return a new row where all items are less than or equal to column width.
write_data(ds, columns, base_format)	Write WOCE data in fixed width columns.
write_bottle(ds)	How to write a Bottle WOCE file.
write_ctd(ds)	How to write a CTD WOCE file.
to_woce(ds)	

Attributes

CTD_ZIP_FILE_EXTENSION
CTD_FILE_EXTENSION
BOTTLE_FILE_EXTENSION
FILL_VALUE
ASTERISK_FLAG
CHARACTER_PARAMETERS
COLUMN_WIDTH
SAFE_COLUMN_WIDTH
UNKNONW_TIME_FILL
BOTTLE_FLAGS
CTD_FLAGS
WATER_SAMPLE_FLAGS
BOTTLE_FLAG_DESCRIPTION
CTD_FLAG_DESCRIPTION
WATER_SAMPLE_FLAG_DESCRIPTION
_UNWRITTEN_COLUMNS
_EXWOCE_PARAMS

hydro.legacy.woce.**CTD_ZIP_FILE_EXTENSION = 'ct.zip'**

hydro.legacy.woce.**CTD_FILE_EXTENSION = 'ct.txt'**

hydro.legacy.woce.**BOTTLE_FILE_EXTENSION = 'hy.txt'**

hydro.legacy.woce.**FILL_VALUE**

hydro.legacy.woce.**ASTERISK_FLAG**

hydro.legacy.woce.**CHARACTER_PARAMETERS = ['STNNBR', 'SAMPNO', 'BTLNBR']**

hydro.legacy.woce.**COLUMN_WIDTH = 8**

hydro.legacy.woce.**SAFE_COLUMN_WIDTH**

hydro.legacy.woce.**UNKNONW_TIME_FILL = '0000'**

hydro.legacy.woce.**BOTTLE_FLAGS**

hydro.legacy.woce.**CTD_FLAGS**

hydro.legacy.woce.**WATER_SAMPLE_FLAGS**

hydro.legacy.woce.**flag_description**(*flag_map*)

hydro.legacy.woce.**BOTTLE_FLAG_DESCRIPTION**

hydro.legacy.woce.**CTD_FLAG_DESCRIPTION**

hydro.legacy.woce.**WATER_SAMPLE_FLAG_DESCRIPTION**

hydro.legacy.woce.**_UNWRITTEN_COLUMNS = ['EXPOCODE', 'SECT_ID', 'LATITUDE', 'LONGITUDE', 'DEPTH', '_DATETIME']**

hydro.legacy.woce.**simplest_str**(*s*)

Give the simplest string representation.

If a float is almost equivalent to an integer, swap out for the integer.

Return type
str

hydro.legacy.woce.**_pad_station_cast**(*x*)

Pad a station or cast identifier out to 5 characters. This is usually for use in a file name.

Parameters
x (*str*) – a string to be padded

Return type
str

hydro.legacy.woce.**get_filename**(*expocode*, *station*, *cast*, *file_ext*)

hydro.legacy.woce.**convert_fortran_format_to_c**(*ffmt*)

Simplistic conversion from Fortran format string to C format string.

This only operates on F formats.

Parameters
ffmt (*str*) –

hydro.legacy.woce.**get_exwoce_params**()

Return a dictionary of WOCE parameters allowed for Exchange conversion.

Returns
{'PMNEMON': {'unit_mnemonic': 'WOCE', 'range': [0.0, 10.0], 'format': '%8.3f'}}

hydro.legacy.woce.**_EXWOCE_PARAMS**

hydro.legacy.woce.**writeable_columns**(*ds*, *is_ctd=False*)

Return the columns that belong in a WOCE data file.

Parameters
ds (*xarray.Dataset*) –

hydro.legacy.woce.**columns_and_base_format**(*dfile*, *is_ctd=False*)

Return columns and base format for WOCE fixed column data.

hydro.legacy.woce.**truncate_row**(*lll*)

Return a new row where all items are less than or equal to column width.

Warnings will be given for any truncations.

hydro.legacy.woce.**write_data**(*ds*, *columns*, *base_format*)

Write WOCE data in fixed width columns.

columns and base_format should be obtained from columns_and_base_format()

hydro.legacy.woce.**write_bottle**(*ds*)

How to write a Bottle WOCE file.

Parameters
ds (*xarray.Dataset*) –

hydro.legacy.woce.**write_ctd**(*ds*)

How to write a CTD WOCE file.

Parameters
ds (*xarray.Dataset*) –

hydro.legacy.woce.**to_woce**(*ds*)

Parameters
ds (*xarray.Dataset*) –

hydro.tests

Subpackages

hydro.tests.data

Submodules

hydro.tests.conftest

Module Contents

Functions

nc_empty()
nc_placeholder()

hydro.tests.conftest.**nc_empty**()

hydro.tests.conftest.**nc_placeholder**()

hydro.tests.test_accessors

Module Contents

Functions

test_gen_fname_machinery(expocode, station, cast, ...)	
test_exchange_bottle_round_trip()	
test_exchange_ctd_round_trip()	

Attributes

exp_stn_cast	

hydro.tests.test_accessors.**exp_stn_cast**

hydro.tests.test_accessors.**test_gen_fname_machinery**(*expocode*, *station*, *cast*, *profile_type*, *profile_count*, *ftype*)

hydro.tests.test_accessors.**test_exchange_bottle_round_trip**()

hydro.tests.test_accessors.**test_exchange_ctd_round_trip**()

hydro.tests.test_core_ops

Module Contents

Functions

test_create_new()	
test_add_profile()	

hydro.tests.test_core_ops.**test_create_new**()

hydro.tests.test_core_ops.**test_add_profile**()

hydro.tests.test_exchange

Module Contents

Functions

test_btl_date_time()	
test_btl_date_time_missing_warn()	
test_ctd_nan()	
test_file_seperator()	
test_reject_bad_examples(data, error)	
test_http_loads(uri, requests_mock)	
test_pressure_flags(flag)	
test_pressure_flags_bad(flag)	
test_duplicate_name_different_units()	
test_duplicate_name_same_units()	
test_multiple_unknown_params()	
test_fix_bottle_time_span()	

hydro.tests.test_exchange.**test_btl_date_time**()

hydro.tests.test_exchange.**test_btl_date_time_missing_warn**()

hydro.tests.test_exchange.**test_ctd_nan**()

hydro.tests.test_exchange.**test_file_seperator**()

hydro.tests.test_exchange.**test_reject_bad_examples**(*data*, *error*)

hydro.tests.test_exchange.**test_http_loads**(*uri*, *requests_mock*)

hydro.tests.test_exchange.**test_pressure_flags**(*flag*)

hydro.tests.test_exchange.**test_pressure_flags_bad**(*flag*)

hydro.tests.test_exchange.**test_duplicate_name_different_units**()

hydro.tests.test_exchange.**test_duplicate_name_same_units**()

hydro.tests.test_exchange.**test_multiple_unknown_params**()

hydro.tests.test_exchange.**test_fix_bottle_time_span**()

hydro.tests.test_merge

Module Contents

Functions

test_fq_merge(nc_placeholder)	
test_fq_merge_with_error(nc_placeholder)	

hydro.tests.test_merge.**test_fq_merge**(*nc_placeholder*)

hydro.tests.test_merge.**test_fq_merge_with_error**(*nc_placeholder*)

hydro.tests.test_rename

Module Contents

Functions

is_not_none(a)	
test_rename()	
test_to_argo_variable_names()	

hydro.tests.test_rename.**is_not_none**(*a*)

hydro.tests.test_rename.**test_rename**()

hydro.tests.test_rename.**test_to_argo_variable_names**()

6.1.2 Submodules

hydro.__main__

Module Contents

Functions

setup_logging(level)		
convert()		
convert_exchange(exchange_path, check_flag, ...)	out_path,	
status()		
cchdo_loader(dtype)		
cached_file_loader(file)		
vars_with_value(ds)		
status_exchange(dtype, dump_unknown_params, ...)	out_dir,	Generate a bottle conversion status for all ex files of type type in the CCHDO Dataset

Attributes

log
cli

hydro.__main__.**log**

hydro.__main__.**setup_logging**(*level*)

hydro.__main__.**convert**()

hydro.__main__.**convert_exchange**(*exchange_path*, *out_path*, *check_flag*, *precision_source*)

hydro.__main__.**status**()

hydro.__main__.**cchdo_loader**(*dtype*)

hydro.__main__.**cached_file_loader**(*file*)

hydro.__main__.**vars_with_value**(*ds*)

Parameters
ds (*xarray.Dataset*) –

Return type
List[str]

hydro.__main__.**status_exchange**(*dtype*, *out_dir*, *dump_unknown_params*, *verbose*, *dump_data_counts*)

Generate a bottle conversion status for all ex files of type type in the CCHDO Dataset

hydro.__main__.**cli**

hydro.__main_helpers

Module Contents

Functions

p_file(file_m)	

hydro.__main_helpers.**p_file**(*file_m*)

hydro._version

Module Contents

hydro._version.**version = '0.1.dev50+g5fc18a4'**

hydro.accessors

Module Contents

Classes

CCHDOAccessorBase	Class base for CCHDO accessors
MatlabAccessor	Accessor containing the experimental matlab machinery
LegacyFormatAccessor	Accessor containing legacy format outputs (coards, woce)
GeoAccessor	Accessor providing geo_interface machinery
MiscAccessor	Accessor with misc functions that don't fit in some other category
ExchangeAccessor	Class containing the to_exchange functionn
WHPIndxer	
MergeFQAccessor	Class base for CCHDO accessors
CCHDOAccessor	Collect all the accessors into a single class

Functions

write_or_return(data[, path_or_fobj])	

Attributes

FLAG_NAME
ERROR_NAME
PathType

hydro.accessors.**FLAG_NAME** = 'cchdo.hydro._qc'

hydro.accessors.**ERROR_NAME** = 'cchdo.hydro._error'

hydro.accessors.**PathType**

hydro.accessors.**write_or_return**(*data*, *path_or_fobj=None*)

Parameters

- **data** (*bytes*) –
- **path_or_fobj** (*Optional[Union[PathType, io.BufferedWriter]]*) –

Return type
Optional[bytes]

class hydro.accessors.**CCHDOAccessorBase**(*xarray_obj*)

Class base for CCHDO accessors

saves the xarray object to self._obj for all the subclasses

Parameters
xarray_obj (*Union[xarray.DataArray, xarray.Dataset]*) –

class hydro.accessors.**MatlabAccessor**(*xarray_obj*)

Bases: *CCHDOAccessorBase*

Accessor containing the experimental matlab machinery

Parameters
xarray_obj (*Union[xarray.DataArray, xarray.Dataset]*) –

to_mat(*fname*)

Experimental Matlab .mat data file generator

The support for netCDF files in Matlab is really bad. Matlab also has no built in support for the standards we are trying to follow (CF, ACDD), the most egregious lack of support is how to deal with times in netCDF files. This was an attempt to make a mat file which takes care of some of the things matlab won't do for you. It requires scipy to function.

The file it produces is in no way stable.

class hydro.accessors.**LegacyFormatAccessor**(*xarray_obj*)

Bases: *CCHDOAccessorBase*

Accessor containing legacy format outputs (coards, woce)

Parameters
xarray_obj (*Union[xarray.DataArray, xarray.Dataset]*) –

to_coards(*path=None*)

to_woce(*path=None*)

to_sum(*path=None*)

netCDF to WOCE sumfile maker

This is missing some information that is not included anymore (wire out, height above bottom). It is especially lacking in including woce parameter IDs

class hydro.accessors.**GeoAccessor**(*xarray_obj*)

Bases: *CCHDOAccessorBase*

Accessor providing geo_interface machinery

Parameters
xarray_obj (*Union[xarray.DataArray, xarray.Dataset]*) –

property __geo_interface__

The station positions as a MultiPoint geo interface

See https://gist.github.com/sgillies/2217756

property track

A dict which can be dumped to json which conforms to the expected structure for the CCHDO website

class hydro.accessors.**MiscAccessor**(*xarray_obj*)

Bases: *CCHDOAccessorBase*

Accessor with misc functions that don't fit in some other category

Parameters
xarray_obj (*Union[xarray.DataArray, xarray.Dataset]*) –

static _gen_fname(*expocode*, *station*, *cast*, *profile_type*, *profile_count=1*, *ftype='cf'*)

Parameters

- **expocode** (*str*) –
- **station** (*str*) –
- **cast** (*int*) –
- **profile_type** (hydro.exchange.FileType) –
- **profile_count** (*int*) –

Return type
str

gen_fname(*ftype='cf'*)

Generate a human friendly netCDF filename for this object

Return type
str

compact_profile()

Drop the trailing empty data from a profile.

Because we use the incomplete multidimensional array representation of profiles there is often "wasted space" at the end of any profile that is not the longest one. This accessor drops that wasted space for xr.Dataset objects containing a single profile

class hydro.accessors.**ExchangeAccessor**(*xarray_obj*)

Bases: *CCHDOAccessorBase*

Class containing the to_exchange functionn

Parameters
xarray_obj (*Union[xarray.DataArray, xarray.Dataset]*) –

property file_type

date_names

time_names

static cchdo_c_format_precision(*c_format*)

Parameters
c_format (*str*) –

Return type
Optional[int]

_make_params_units_line(*params*)

Parameters
params (*Dict[cchdo.params.WHPName, xarray.DataArray]*) –

static _whpname_from_attrs(*attrs*)

Return type
List[cchdo.params.WHPName]

_make_ctd_headers(*params*)

Return type
List[str]

_make_data_block(*params*)

Return type
List[str]

_get_comments()

to_whp_columns(*compact=False*)

to_exchange(*path=None*)

Convert a CCHDO CF netCDF dataset to exchange

class hydro.accessors.**WHPIndxer**(*obj*)

Parameters
obj (*xarray.Dataset*) –

__getitem__(*key*)

class hydro.accessors.**MergeFQAccessor**(*xarray_obj*)

Bases: *CCHDOAccessorBase*

Class base for CCHDO accessors

saves the xarray object to self._obj for all the subclasses

Parameters
xarray_obj (*Union[xarray.DataArray, xarray.Dataset]*) –

merge_fq(*fq*, *, *check_flags=True*)

class hydro.accessors.**CCHDOAccessor**(*xarray_obj*)

Bases: *ExchangeAccessor*, *GeoAccessor*, *LegacyFormatAccessor*, *MatlabAccessor*, *MiscAccessor*, *MergeFQAccessor*

Collect all the accessors into a single class

Parameters
xarray_obj (*Union[xarray.DataArray, xarray.Dataset]*) –

hydro.conformance

Module Contents

Classes

CheckResult

CCHDOnetCDF10

Attributes

FORMAT

log

CCHDOnetCDF

hydro.conformance.**FORMAT** = '%(message)s'

hydro.conformance.**log**

class hydro.conformance.**CheckResult**

property ok: bool

Return type
bool

error: Optional[str]

warning: Optional[str]

class hydro.conformance.**CCHDOnetCDF10**

__cchdo_version__ = '1.0'

check_cf_version(*ds*)

iter_errors(*ds*)

Parameters
ds (*xarray.Dataset*) –

validate(*ds*)

Parameters
ds (*xarray.Dataset*) –

hydro.conformance.**CCHDOnetCDF**

hydro.convert

Functions for converting objects from one to another

For example, exchange flags to argo flags

hydro.core

Core operations on a CCHDO CF/netCDF file

Module Contents

Functions

_dataarray_factory(param[, ctype, N_PROF, N_LEVELS])	
add_param(ds, param[, with_flag])	
add_profile_level(ds, idx, levels)	
add_level(ds[, n_levels])	
add_profile(ds, expocode, station, cast, time, ...)	
create_new()	Create an empty CF Dataset with the minimum required contents

Attributes

DIMS
FILLS_MAP
dtype_map
EXPOCODE
STNNBR
CASTNO
SAMPNO
DATE
TIME
LATITUDE
LONGITUDE
CTDPRS
BTLNBR
COORDS
FLAG_SCHEME

hydro.core.**DIMS = ('N_PROF', 'N_LEVELS')**

hydro.core.**FILLS_MAP**

hydro.core.**dtype_map**

hydro.core.**EXPOCODE**

hydro.core.**STNNBR**

hydro.core.**CASTNO**

hydro.core.**SAMPNO**

hydro.core.**DATE**

hydro.core.**TIME**

hydro.core.**LATITUDE**

hydro.core.**LONGITUDE**

hydro.core.**CTDPRS**

hydro.core.**BTLNBR**

hydro.core.**COORDS**

hydro.core.**FLAG_SCHEME**

hydro.core.**_dataarray_factory**(*param*, *ctype='data'*, *N_PROF=0*, *N_LEVELS=0*)

Parameters
param (*cchdo.params.WHPName*) –

Return type
xarray.DataArray

hydro.core.**add_param**(*ds*, *param*, *with_flag=False*)

Parameters

- **ds** (*xarray.Dataset*) –
- **param** (*cchdo.params.WHPName*) –

Return type
xarray.Dataset

hydro.core.**add_profile_level**(*ds*, *idx*, *levels*)

Parameters
ds (*xarray.Dataset*) –

Return type
xarray.Dataset

hydro.core.**add_level**(*ds*, *n_levels=1*)

Parameters
ds (*xarray.Dataset*) –

Return type
xarray.Dataset

hydro.core.**add_profile**(*ds*, *expocode*, *station*, *cast*, *time*, *latitude*, *longitude*, *profile_type*)

Parameters

- **ds** (*xarray.Dataset*) –
- **expocode** (*numpy.typing.ArrayLike*) –
- **station** (*numpy.typing.ArrayLike*) –
- **cast** (*numpy.typing.ArrayLike*) –
- **time** (*numpy.typing.ArrayLike*) –
- **latitude** (*numpy.typing.ArrayLike*) –
- **longitude** (*numpy.typing.ArrayLike*) –
- **profile_type** (*numpy.typing.ArrayLike*) –

Return type
xarray.Dataset

hydro.core.**create_new**()

Create an empty CF Dataset with the minimum required contents

Return type
xarray.Dataset

hydro.rename

Module Contents

Functions

is_not_none(obj)	
rename_with_bookkeeping(xarray_obj[, name_dict, attrs])	Find and update all instances of a given variable to a new name.
to_argo_variable_names(xarray_obj)	

hydro.rename.**is_not_none**(*obj*)

hydro.rename.**rename_with_bookkeeping**(*xarray_obj*, *name_dict=None*, *attrs=None*)

Find and update all instances of a given variable to a new name.

Parameters can be referenced in the attributes of separate parameter (e.g. `ancillary_variables`) and need to be updated appropriately when renaming variables.

Parameters

- **xarray_obj** (*xarray.Dataset*) – A Dataset containing variables, flags, etc.
- **name_dict** (*Mapping*) – Mapping of old variable names to new.
- **attrs** (*List[str]*) – Names of variable attributes to search through.

Return type
xarray.Dataset

hydro.rename.**to_argo_variable_names**(*xarray_obj*)

Parameters
xarray_obj (*xarray.Dataset*) –

Return type
xarray.Dataset

hydro.tutorial

Module Contents

Classes

CCHDOBottleData	

Functions

_cache_dir()	
load_cchdo_bottle_data()	Downloads some CCHDO data for playing with...

Attributes

bottle_uri	
bottle_fname	

hydro.tutorial.**bottle_uri** = **'https://cchdo.ucsd.edu/search?q=a&download=exchange%2cbottle'**

hydro.tutorial.**bottle_fname** = **'bottle_data.zip'**

hydro.tutorial.**_cache_dir**()

hydro.tutorial.**load_cchdo_bottle_data**()

Downloads some CCHDO data for playing with…

class hydro.tutorial.**CCHDOBottleData**

Bases: collections.abc.Mapping

__len__()

__iter__()

__getitem__(*key*)

6.1.3 Package Contents

Functions

read_exchange(filename_or_obj, *[, fill_values, ...])	Loads the data from filename_or_obj and returns a xr.Dataset with the CCHDO

hydro.**read_exchange**(*filename_or_obj*, *, *fill_values=('-999',)*, *checks=None*, *precision_source='file'*, *file_seperator=None*, *keep_seperator=True*)

Loads the data from filename_or_obj and returns a xr.Dataset with the CCHDO CF/netCDF structure

Parameters

- **filename_or_obj** (*ExchangeIO*) –
- **checks** (*Optional[CheckOptions]*) –

Return type

xarray.Dataset

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W