adi_py

Release 4.0.0

Carina Lansing

May 04, 2022

CONTENTS

To start developing a VAP with the new API, please use the create_adi_project tool to stub out the new file format:

```
[lansing@emerald ~]$ create_adi_project
usage: create_adi_project [-h] [-t TEMPLATE] [-p PROCESS] [-o OUTPUT]
                          [-a AUTHOR] [-n PHONE] [-e EMAIL] [-i INPUT]
                          [-d DUMP_JSON] [-v DOD_VERSION]
```

In particular, use one of the two new templates:

```
----------------------------------------
New VAP TEMPLATES available in Py
vap-py-retriever  --> uses the new xarray API
vap-py-transform  --> uses the new xarray API
----------------------------------------
```

The main difference between old ADI Py processes and ones created with the new API is the `__main__.py` file. Instead of implementing dsproc hook functions individually, new processes will extend the Process base class and then the main function will simply call Process.run() to invoke ADI.

Note

Much of the functionality that was previously in the __main__.py template is now built into the Process base class (or XArrray). Therefore, you will see that the new templates contain significantly less boilerplate code.

Each Process can implement any of the ADI hooks by overriding the default (empty) implementation in the base class. Please see the tutorial_py project for an example. Also review the Process API Reference for a list of functions that are available to your class.

Prerequisites

For the best experience working with the new ADI Python API, we recommend a solid foundation in the XArray and Numpy libraries. Below are a list of tutorials to get you started:

1. https://xarray-contrib.github.io/xarray-tutorial/scipy-tutorial/00_overview.html
2. https://www.youtube.com/watch?v=mecN-Ph_-78
3. https://www.youtube.com/watch?v=xdrcMi_FB8Q
4. http://gallery.pangeo.io/repos/pangeo-data/pangeo-tutorial-gallery/xarray.html

---

CHAPTER

ONE

DEVELOPING WITH AN IDE

We **strongly** recommend that you use an IDE such as VSCode or PyCharm to develop your code remotely. This will enable you to get autocomplete support for all Process functions which include built-in Python docstrings. This will make it significantly easier to work with the new API. In addition, you will be able to set break points and step through your code which will make it much easier to debug.

The new create_adi_project templates will provide a **.vscode folder** that contains a README file with instructions and all the config files needed to get started debugging remotely with VSCode. Additional instructions for PyCharm will be provided at a later time. Click here to view the VSCode README file.

CHAPTER

TWO

DEVELOPER SUPPORT

This page provides useful resources for developers working with the new Python API

2.1 Examples

The following processes have been converted over to use the new Python API and provide different examples of working with XArray:

1. tutorial_py
2. adi_example1_py_newapi
3. lassolwp (new_pyapi branch)

2.2 code.arm.gov Code Search

To find additional projects using the new API, please utilize the code.arm.gov Code Search tool and search for this query.

2.3 Stack Overflow

We have created a private Stack Overflow team for ADI developers that contains answers to common questions encountered during the development process. Feel free to post XArray questions to this site so we can build our knowledge base. To gain access, please send an email to Maxwell Levin or Krista Gaustad at PNNL. You will also need a Stack Overflow account, which is free to join.

2.4 ARM DI Slack Workspace

Click here to join the ARM DI workspace

2.5 XArray Tutorials

1. https://xarray-contrib.github.io/xarray-tutorial/scipy-tutorial/00_overview.html
2. https://www.youtube.com/watch?v=mecN-Ph_-78
3. https://www.youtube.com/watch?v=xdrcMi_FB8Q
4. http://gallery.pangeo.io/repos/pangeo-data/pangeo-tutorial-gallery/xarray.html

CHAPTER

THREE

API REFERENCE

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

3.1 adi_py

This module provides the new ADI Python bindings which incorporate full XArray compatibility.

3.1.1 Submodules

3.1.1.1 adi_py.constants

This module provides a variety of symbolic constants that can be used to find/select allowable values without hardcoding strings.

3.1.1.1.1 Classes

ADIAtts	
ADIDatasetType	Used to easily reference different types of ADI datasets.
BitAssessment	Used to easily reference bit assessment values used in ADI QC
SpecialXrAttributes	Enumerates the special XArray variable attributes that are assigned
SplitMode	Enumerates the split mode which is used to define the output file size
TransformAttributes	Used to easily reference transformation metadata attrs used in ADI QC

class adi_py.constants.**ADIAtts**

ANCILLARY_VARIABLES = ancillary_variables

DESCRIPTION = description

FILL_VALUE = ['_FillValue']

[1] Created with sphinx-autoapi

LONG_NAME = long_name

MISSING_VALUE = missing_value

STANDARD_NAME = standard_name

UNITS = units

VALID_MAX = valid_max

VALID_MIN = valid_min

---

class adi_py.constants.ADIDatasetType
Bases: enum.Enum

Used to easily reference different types of ADI datasets.

OUTPUT = 3

RETRIEVED = 1

TRANSFORMED = 2

---

class adi_py.constants.BitAssessment
Bases: enum.Enum

Used to easily reference bit assessment values used in ADI QC

BAD = Bad

INDETERMINATE = Indeterminate

---

class adi_py.constants.SpecialXrAttributes
Enumerates the special XArray variable attributes that are assigned temporarily to help sync data between xarray and adi.

COORDINATE_SYSTEM = __coordsys_name

DATASET_TYPE = __dataset_type

DATASTREAM_DSID = __datastream_dsid

OBS_INDEX = __obs_index

OUTPUT_TARGETS = __output_targets

SOURCE_DS_NAME = __source_ds_name

SOURCE_VAR_NAME = __source_var_name

---

class adi_py.constants.SplitMode
Bases: enum.Enum

Enumerates the split mode which is used to define the output file size used when storing values. See dsproc.set_datastream_split_mode().

SPLIT_ON_DAYS

SPLIT_ON_HOURS

SPLIT_ON_MONTHS

SPLIT_ON_STORE

---

class adi_py.constants.**TransformAttributes**

Used to easily reference transformation metadata attrs used in ADI QC

CELL_TRANSFORM = cell_transform

3.1.1.2 adi_py.exception

This module defines ADI exceptions which may be thrown from process hooks.

exception adi_py.exception.**SkipProcessingIntervalException**

Bases: Exception

Processes should throw this exception if the current processing interval should be skipped. All other exceptions will be considered to fail the process.

Initialize self. See help(type(self)) for accurate signature.

3.1.1.3 adi_py.logger

This module contains the **ADILogger** class which provides a python-like logging API facade around the dsproc logging methods.

3.1.1.3.1 Classes

ADILogger	This class provides python-like logging API facade around the dsproc

class adi_py.logger.**ADILogger**

This class provides python-like logging API facade around the dsproc logging methods.

Class Methods

debug	
error	
exception	Use this method to log the stack trace of any raised exception to the process's
info	
warning	

Method Descriptions

static debug(*message*, *debug_level=1*)

static error(*message*)

static exception(*message*)
Use this method to log the stack trace of any raised exception to the process's ADI log file.

Parameters **message** (-) – str An optional additional message to log, in addition to the stack trace.

static info(*message*)

static warning(*message*)

3.1.1.4 adi_py.process

This module contains the **Process** class which should be used for implementing ADI processes in Python.

Important: All new ADI Python processes MUST extend this base class. This class implements empty hook functions for all possible ADI hooks. You only need to override the hook methods that are required for your process.

3.1.1.4.1 Classes

Process	The base class for running an ADI process in Python. All Python processes

class adi_py.process.**Process**
The base class for running an ADI process in Python. All Python processes should extend this class.

Class Methods

add_qc_variable	Add a companion qc variable for the given variable
add_variable	Create a new variable in the given xarray dataset with the specified dimensions,
assign_coordinate_system_to_variable	Assign the given variable to the designated ADI coordinate system.
assign_output_datastream_to_variable	Assign the given variable to the designated output datastream.
convert_units	For the specified variables, convert the units from old_units to new_units.
debug_level	Get the debug level passed on the command line when running the process.
drop_transform_metadata	This method removes all associated companion variables that are generated
drop_variables	This method removes the given variables plus all associated companion
facility	Get the facility where this invocation of the process is running
find_retrieved_variable	Find the input datastream where the given retrieved variable came

continues on next page

Table 12 – continued from previous page

finish_process_hook	This hook will be called once just after the main data processing loop finishes. This function should be used
get_bad_qc_mask	Get a mask of same shape as the variable's data which contains True values
get_companion_transform_variable_names	For the given variable, get a list of the companion/ancillary variables
get_datastream_files	See `utils.get_datastream_files()`
get_dsid	Gets the corresponding dataset id for the given datastream (input or output)
get_missing_value_mask	Get True/False mask of same shape as passed variable(s) which is used to
get_non_missing_value_mask	Get a True/False mask of same shape as passed variable(s) that is used to
get_nsamples	Get the ADI sample count for the given variable (i.e., the length
get_output_dataset	Get an ADI output dataset converted to an xarray.Dataset.
get_output_dataset_by_dsid	
get_qc_variable	Return the companion qc variable for the given data variable.
get_quicklooks_file_name	Create a properly formatted file name where a quicklooks plot should be
get_retrieved_dataset	Get an ADI retrieved dataset converted to an xarray.Dataset.
get_retrieved_dataset_by_dsid	
get_source_ds_name	For the given variable, get name of the input datastream
get_source_var_name	For the given variable, get the name of the variable
get_transformed_dataset	Get an ADI transformed dataset converted to an xarray.Dataset.
get_transformed_dataset_by_dsid	
include_debug_dumps	Setting controlling whether this process should provide debug dumps of the
init_process_hook	This hook will will be called once just before the main data processing loop begins and before the initial
location	Get the location where this invocation of the process is running.
post_retrieval_hook	This hook will will be called once per processing interval just after data retrieval,
post_transform_hook	This hook will be called once per processing interval just after data
pre_retrieval_hook	This hook will will be called once per processing interval just prior to data retrieval.
pre_transform_hook	This hook will be called once per processing interval just prior to data
process_data_hook	This hook will be called once per processing interval just after the output

continues on next page

Table 12 – continued from previous page

process_model	The processing model to use. It can be one of:
process_name	The name of the process that is currently being run.
process_names	The name(s) of the process(es) that could run this code. Subclasses must
process_version	The version of this process's code. Subclasses must define the
quicklook_hook	This hook will be called once per processing interval just after all data
record_qc_results	For the given variable, add bitwise test results to the companion qc
rollup_qc	ADI setting controlling whether all the qc bits are rolled up into a
run	Run the process.
set_datastream_flags	Apply a set of ADI control flags to a datastream as identified by the
set_datastream_split_mode	This method should be called in your init_process_hook if you need to
set_retriever_time_offsets	This method should be called in your init_process_hook if you need to override
shift_output_interval	This method should be called in your init_process_hook (i.e., before the
shift_processing_interval	This method should be called in your init_process_hook (i.e., before the
site	Get the site where this invocation of the process is running
sync_datasets	Sync the contents of one or more XArray.Datasets with the corresponding ADI
variables_exist	Check if the given variables exist in the given dataset.

Method Descriptions

static add_qc_variable(*dataset: xarray.Dataset*, *variable_name: str*)

Add a companion qc variable for the given variable

Parameters

- **dataset** (*xr.Dataset*) –
- **variable_name** (*str*) –

Returns The newly created DataArray

static add_variable(*dataset: xarray.Dataset*, *variable_name: str*, *dim_names: List[str]*, *data: numpy.ndarray*, *long_name: str = None*, *standard_name: str = None*, *units: str = None*, *valid_min: Any = None*, *valid_max: Any = None*, *missing_value: numpy.ndarray = None*, *fill_value: Any = None*)

Create a new variable in the given xarray dataset with the specified dimensions, data, and attributes.

Important: If you want to add the created variable to a given coordinate system, then you follow this with a call to assign_coordinate_system_to_variable. Similarly, if you want to add the created variable to a given output datastream, then you should follow this with a call to assign_output_datastream_to_variable

---

See also:

- assign_coordinate_system_to_variable
- assign_output_datastream_to_variable

Parameters

- **dataset** (*xr.Dataset*) – The xarray dataset to add the new variable to
- **variable_name** (*str*) – The name of the variable
- **dim_names** (*List[str]*) – A list of dimension names for the variable
- **data** (*np.ndarray*) – A multidimensional array of the variable's data Must have the same shape as the dimensions.
- **long_name** (*str*) – The long_name attribute for the variable
- **standard_name** (*str*) – The standard_name attribute for the variable
- **units** (*str*) – The units attribute for the variable
- **valid_min** (*Any*) – The valid_min attribute for the variable. Must be the same data type as the variable.
- **valid_max** (*Any*) – The valid_max attribute for the variable Must be the same data type as the variable.
- **missing_value** (*np.ndarray*) – An array of possible missing_value attributes for the variable. Must be the same data type as the variable.
- **()** (*fill_value*) – The fill_value attribute for the variable. Must be the same data type as the variable.

Returns The newly created variable (i.e., xr.DataArray object)

static assign_coordinate_system_to_variable(*variable: xarray.DataArray*, *coordinate_system_name: str*)

Assign the given variable to the designated ADI coordinate system.

Parameters

- **variable** (*xr.DataArray*) – A data variable from an xarray dataset
- **coordinate_system_name** (*str*) – The name of one of the process's coordinate systems as specified in the PCM process definition.

static assign_output_datastream_to_variable(*variable: xarray.DataArray*, *output_datastream_name: str*, *variable_name_in_datastream: str = None*)

Assign the given variable to the designated output datastream.

Parameters

- **variable** (*xr.DataArray*) – A data variable from an xarray dataset
- **output_datastream_name** (*str*) – An output datastream name as specified in PCM process definition
- **variable_name_in_datastream** (*str*) – The name of the variable as it should appear in the output datastream. If not specified, then the name of the given variable will be used.

static convert_units(*xr_datasets: List[xarray.Dataset]*, *old_units: str*, *new_units: str*, *variable_names: List[str] = None*, *converter_function: Callable = None*)

For the specified variables, convert the units from old_units to new_units. For applicable variables, this conversion will include changing the units attribute value and optionally converting all the data values if a converter function is provided.

This method is needed for special cases where the units conversion is not supported by udunits and the default ADI converters.

Parameters

- **xr_datasets** (*List[xr.Dataset]*) – One or more xarray datasets upon which to apply the conversion
- **old_units** (*str*) – The old units (e.g., 'degree F')
- **new_units** (*str*) – The new units (e.g., 'K')
- **variable_names** (*List[str]*) – A list of specific variable names to convert. If not specified, it converts all variables with the given old_units to new_units.
- **()** (*converter_function*) – A function to run on an Xarray variable (i.e., DataArray that converts a variable's values from old_units to new_units. If not specified, then only the units attribute value will be changed. This could happen if we just want to change the units attribute value because of a typo.

  The function should take one parameter, an xarray.DataArray, and operate in place on the variable's values.

property debug_level(*self*) → int

Get the debug level passed on the command line when running the process.

Returns *int* – the debug level

static drop_transform_metadata(*dataset: xarray.Dataset*, *variable_names: List[str]*) → xarray.Dataset

This method removes all associated companion variables that are generated byt the transformation process (if they exist), as well as transformation attributes, but it does not remove the original variable.

Parameters

- **dataset** (*xr.Dataset*) – The dataset containing the transformed variables.
- **variable_names** (*List[str]*) – The variable names for which to remove transformation metadata.

Returns *xr.Dataset* – A new dataset with the transform companion variables and metadata removed.

static drop_variables(*dataset: xarray.Dataset*, *variable_names: List[str]*) → xarray.Dataset

This method removes the given variables plus all associated companion variables that were added as part of the transformation step (if they exist).

Parameters

- **dataset** (*xr.Dataset*) – The dataset containing the given variables.
- **variable_names** (*List[str]*) – The variable names to remove.

Returns *xr.Dataset* – A new dataset with the given variables and their transform companion variables removed.

property facility(*self*) → str

Get the facility where this invocation of the process is running

Returns *str* – The facility where this process is running

static find_retrieved_variable(*retrieved_variable_name*) → Optional[*adi_py.utils.DatastreamIdentifier*]

Find the input datastream where the given retrieved variable came from. We may need this if there are complex retrieval rules and the given variable may be retrieved from different datastreams depending upon the site/facility where this process runs. We need to get the DatastreamIdentifier so we can load the correct xarray dataset if we need to modify the data values.

Parameters **retrieved_variable_name** (*str*) – The name of the retrieved variable to find

Returns A DatastreamIdentifier containing all the information needed to look up the given dataset or None if the retrieved variable was not found.

finish_process_hook(*self*)

This hook will be called once just after the main data processing loop finishes. This function should be used to clean up any temporary files used.

static get_bad_qc_mask(*dataset: xarray.Dataset*, *variable_name: str*, *include_indeterminate: bool = False*, *bit_numbers: List[int] = None*) → numpy.ndarray

Get a mask of same shape as the variable's data which contains True values for each data point that has a corresponding bad qc bit set.

Parameters

- **dataset** (*xr.Dataset*) – The dataset containing the variables
- **variable_name** (*str*) – The variable name to check qc for
- **include_indeterminate** (*bool*) – Whether to include indeterminate bits when determining the mask. By default this is False and only bad bits are used to compute the mask.
- **bit_numbers** (*List(int)*) – The specific bit numbers to include in the qc check (i.e., 1,2,3,4, etc.). Note that if not specified, all bits will be used to compute the mask.

Returns *np.ndarray* – An array of same shape as the variable consisting of True/False values, where each True indicates that the corresponding data point had bad (or indeterminate if include_indeterminate is specified) qc for the specified bit numbers (all bits if bit_numbers not specified).

static get_companion_transform_variable_names(*dataset: xarray.Dataset*, *variable_name: str*) → List[str]

For the given variable, get a list of the companion/ancillary variables that were added as a result of the ADI transformation.

Parameters

- **dataset** (*xr.Dataset*) – The dataset
- **variable_name** (*str*) – The name of a data variable in the dataset

Returns A list of string companion variable names that were created from the transform engine. This is used for cleaning up associated variables when a variable is deleted from a dataset.

static get_datastream_files(*datastream_name: str*, *begin_date: int*, *end_date: int*) → List[str]

See `utils.get_datastream_files()`

static get_dsid(*datastream_name: str*, *site: str = None*, *facility: str = None*, *dataset_type: adi_py.constants.ADIDatasetType = None*) → Optional[int]

Gets the corresponding dataset id for the given datastream (input or output)

Parameters

- **datastream_name** (*str*) – The name of the datastream to find

- **site** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Site is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by site.
- **facility** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Facility is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by facility.
- **dataset_type** (*ADIDatasetType*) – The type of the dataset to convert (RETRIEVED, TRANSFORMED, OUTPUT)

Returns *Optional[int]* – The dataset id or None if not found

static get_missing_value_mask(**args*) → xarray.DataArray

Get True/False mask of same shape as passed variable(s) which is used to select data points for which one or more of the values of any of the specified variables are missing.

Parameters ***args** (*xr.DataArray*) – Pass one or more xarray variables to check for missing values. All variables in the list must have the same shape.

Returns *xr.DataArray* – An array of True/False values of the same shape as the input variables where each True represents the case where one or more of the variables has a missing_value at that index.

static get_non_missing_value_mask(**args*) → xarray.DataArray

Get a True/False mask of same shape as passed variable(s) that is used to select data points for which none of the values of any of the specified variables are missing.

Parameters ***args** (*xr.DataArray*) – Pass one or more xarray variables to check. All variables in the list must have the same shape.

Returns *xr.DataArray* – An array of True/False values of the same shape as the input variables where each True represents the case where all variables passed in have non-missing value data at that index.

static get_nsamples(*xr_var: xarray.DataArray*) → int

Get the ADI sample count for the given variable (i.e., the length of the first dimension or 1 if the variable has no dimensions)

Parameters **xr_var** (*xr.DataArray*) –

Returns *int* – The ADI sample count

static get_output_dataset(*output_datastream_name: str*) → Union[xarray.Dataset, List[xarray.Dataset]]

Get an ADI output dataset converted to an xarray.Dataset.

Parameters **output_datastream_name** (*str*) – The name of one of the process' output datastreams as specified in the PCM.

Returns *Union[xr.Dataset, List[xr.Dataset]]* – Most of the time, return a single xr.Dataset. If the process is using file-based processing or if there are multiple files for the same datastream and a dimensionality conflict prevented the files from being merged, then return a List[XArray.Dataset], one for each file.

static get_output_dataset_by_dsid(*dsid: int*) → Union[xarray.Dataset, List[xarray.Dataset]]

static get_qc_variable(*dataset: xarray.Dataset*, *variable_name: str*) → xarray.DataArray

Return the companion qc variable for the given data variable.

Parameters

- **dataset** (*xr.Dataset*) –

- **variable_name** (*str*) –

Returns *xr.DataArray* – The companion qc variable or None if it doesn't exist

get_quicklooks_file_name(*self*, *datastream_name: str*, *begin_date: int*, *description: str = None*, *ext: str = 'png'*, *mkdirs: bool = False*)

Create a properly formatted file name where a quicklooks plot should be saved for the given processing interval. For example:

${QUICKLOOK_DATA}/ena/enamfrsrcldod1minC1.c1/2021/01/01/enamfrsrcldod1minC1.c1.20210101.000000.lwp.png

Parameters

- **datastream_name** (*str*) – The name of the datastream which this plot applies to. For example, *mfrsrcldod1min.c1*
- **begin_date** (*int*) – The begin timestamp of the current processing interval as passed to the quicklook hook function
- **description** (*str*) – The description of the plot to be used in the file name For example, in the file *enamfrsrcldod1minC1.c1.20210101.000000.lwp.png*, the description is 'lwp'.
- **ext** (*str*) – The file extension for the image. Default is 'png'
- **mkdirs** (*bool*) – If True, then the folder path to the quicklooks file will be automatically created if it does not exist. Default is False.

Returns *str* – The full path to where the quicklooks file should be saved.

static get_retrieved_dataset(*input_datastream_name: str*, *site: str = None*, *facility: str = None*) → Union[xarray.Dataset, List[xarray.Dataset]]

Get an ADI retrieved dataset converted to an xarray.Dataset.

Parameters

- **input_datastream_name** (*str*) – The name of one of the process' input datastreams as specified in the PCM.
- **site** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Site is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by site.
- **facility** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Facility is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by facility.

Returns *Union[xr.Dataset, List[xr.Dataset]]* – Most of the time, return a single xr.Dataset. If the process is using file-based processing or if there are multiple files for the same datastream and a dimensionality conflict prevented the files from being merged, then return a List[XArray.Dataset], one for each file.

static get_retrieved_dataset_by_dsid(*dsid: int*) → Union[xarray.Dataset, List[xarray.Dataset]]

static get_source_ds_name(*xr_var: xarray.DataArray*) → str

For the given variable, get name of the input datastream where it came from :param xr_var: :type xr_var: xr.DataArray

Returns: str

static get_source_var_name(*xr_var: xarray.DataArray*) → str

For the given variable, get the name of the variable used in the input datastream :param xr_var: :type xr_var: xr.DataArray

Returns: str

static get_transformed_dataset(*input_datastream_name: str*, *coordinate_system_name: str*, *site: str = None*, *facility: str = None*) → Union[xarray.Dataset, List[xarray.Dataset]]

Get an ADI transformed dataset converted to an xarray.Dataset.

Parameters

- **input_datastream_name** (*str*) – The name of one of the process' input datastreams as specified in the PCM.
- **coordinate_system_name** (*str*) – A coordinate system specified in the PCM or None if no coordinate system was specified.
- **site** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Site is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by site.
- **facility** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Facility is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by facility.

Returns *Union[xr.Dataset, List[xr.Dataset]]* – Most of the time, return a single xr.Dataset. If the process is using file-based processing or if there are multiple files for the same datastream and a dimensionality conflict prevented the files from being merged, then return a List[XArray.Dataset], one for each file.

static get_transformed_dataset_by_dsid(*dsid: int*, *coordinate_system_name: str*) → Union[xarray.Dataset, List[xarray.Dataset]]

property include_debug_dumps(*self*) → bool

Setting controlling whether this process should provide debug dumps of the data after each hook.

Returns *bool* – Whether debug dumps should be automatically included. If True and debug level is > 1, then debug dumps will be performed automatically before and after each code hook.

init_process_hook(*self*)

This hook will will be called once just before the main data processing loop begins and before the initial database connection is closed.

property location(*self*) → dsproc3.PyProcLoc

Get the location where this invocation of the process is running.

Returns *dsproc.PyProcLoc* – A class containing the alt, lat, and lon where the process is running.

post_retrieval_hook(*self*, *begin_date: int*, *end_date: int*)

This hook will will be called once per processing interval just after data retrieval, but before the retrieved observations are merged and QC is applied.

Parameters

- **begin_date** (*int*) – the begin time of the current processing interval
- **end_date** (*int*) – the end time of the current processing interval

post_transform_hook(*self*, *begin_date: int*, *end_date: int*)

This hook will be called once per processing interval just after data transformation, but before the output datasets are created.

Parameters

- **begin_date** (*int*) – the begin time of the current processing interval
- **end_date** (*int*) – the end time of the current processing interval

pre_retrieval_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will will be called once per processing interval just prior to data retrieval.

Parameters

- **begin_date** (*int*) – the begin time of the current processing interval
- **end_date** (*int*) – the end time of the current processing interval

pre_transform_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will be called once per processing interval just prior to data transformation,and after the retrieved observations are merged and QC is applied.

Parameters

- **begin_date** (*int*) – the begin time of the current processing interval
- **end_date** (*int*) – the end time of the current processing interval

process_data_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will be called once per processing interval just after the output datasets are created, but before they are stored to disk.

Parameters

- **begin_date** (*int*) – the begin time of the current processing interval
- **end_date** (*int*) – the end time of the current processing interval

property process_model(*self*) → int
The processing model to use. It can be one of:

dsproc.PM_GENERIC dsproc.PM_INGEST dsproc.PM_RETRIEVER_INGEST
dsproc.PM_RETRIEVER_VAP dsproc.PM_TRANSFORM_INGEST
dsproc.PM_TRANSFORM_VAP

Default value is PM_TRANSFORM_VAP. Subclasses can override in their constructor.

Returns *int* – The processing modelj (see dsproc.ProcModel cdeftype)

property process_name(*self*) → str
The name of the process that is currently being run.

Returns *str* – the name of the current process

property process_names(*self*) → List[str]
The name(s) of the process(es) that could run this code. Subclasses must define the self._names field in their constructor.

Returns *List[str]* – One or more process names

property process_version(*self*) → str
The version of this process's code. Subclasses must define the self._process_version field in their constructor.

Returns *str* – The process version

quicklook_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will be called once per processing interval just after all data is stored.

Parameters

- **begin_date** (*int*) – the begin timestamp of the current processing interval
- **end_date** (*int*) – the end timestamp of the current processing interval

static record_qc_results(*xr_dataset: xarray.Dataset*, *variable_name: str*, *bit_number: int = None*, *test_results: numpy.ndarray = None*)

For the given variable, add bitwise test results to the companion qc variable for the given test.

Parameters

- **xr_dataset** (*xr.Dataset*) – The xr dataset
- **variable_name** (*str*) – The name of the data variable (e.g., rh_ambient).
- **bit_number** (*int*) – The bit/test number to record Note that bit numbering starts at 1 (i.e., 1, 2, 3, 4, etc.)
- **test_results** (*np.ndarray*) – A ndarray mask of the same shape as the variable with True/False values for each data point. True means the test failed for that data point. False means the test passed for that data point.

property rollup_qc(*self*) → bool

ADI setting controlling whether all the qc bits are rolled up into a single 0/1 value or not.

Returns *bool* – Whether this process should rollup qc or not

run(*self*) → int

Run the process.

Returns

int – The processing status:

- 1 if an error occurred
- 0 if successful

static set_datastream_flags(*dsid: int*, *flags: int*)

Apply a set of ADI control flags to a datastream as identified by the dsid. Multiple flags can be combined together using a bitwise OR (e.g., dsproc.DS_STANDARD_QC | dsproc.DS_FILTER_NANS). The allowed flags are identified below:

dsproc.DS_STANDARD_QC = Apply standard QC before storing a dataset.

dsproc.DS_FILTER_NANS = Replace NaN and Inf values with missing values before storing a dataset.

dsproc.DS_OVERLAP_CHECK = Check for overlap with previously processed data. This flag will be ignored and the overlap check will be skipped if reprocessing mode is enabled, or asynchronous processing mode is enabled.

dsproc.DS_PRESERVE_OBS = Preserve distinct observations when retrieving data. Only observations that start within the current processing interval will be read in.

dsproc.DS_DISABLE_MERGE = Do not merge multiple observations in retrieved data. Only data for the current processing interval will be read in.

dsproc.DS_SKIP_TRANSFORM = Skip the transformation logic for all variables in this datastream.

dsproc.DS_ROLLUP_TRANS_QC = Consolidate the transformation QC bits for all variables when mapped to the output datasets.

dsproc.DS_SCAN_MODE = Enable scan mode for datastream that are not expected to be continuous. This prevents warning messages from being generated when data is not found within a processing interval. Instead, a message will be written to the log file indicating that the procesing interval was skipped.

dsproc.DS_OBS_LOOP = Loop over observations instead of time intervals. This also sets the DS_PRESERVE_OBS flag.

dsproc.DS_FILTER_VERSIONED_FILES = Check for files with .v# version extensions and filter out lower versioned files. Files without a version extension take precedence.

Call self.get_dsid() to obtain the dsid value for a specific datastream. If the flags value is < 0, then the following default flags will be set:

dsprc.DS_STANDARD_QC 'b' level datastreams dsproc.DS_FILTER_NANS 'a' and 'b' level datastreams dsproc.DS_OVERLAP_CHECK all output datastreams dsproc.DS_FILTER_VERSIONED_FILES input datastreams that are not level '0'

Parameters

- **dsid** (*int*) – Datastream ID
- **flags** (*int*) – Flags to set

Returns *int* – The processing modelj (see dsproc.ProcModel cdeftype)

static set_datastream_split_mode(*output_datastream_name: str*, *split_mode:* adi_py.constants.SplitMode, *split_start: int*, *split_interval: int*)

This method should be called in your init_process_hook if you need to change the size of the output file for a given datastream. For example, to create monthly output files.

Parameters

- **output_datastream_name** (*str*) – The name of the output datastream whose file output size will be changed.
- **split_mode** (*SplitMode*) – One of the options from the SplitMode enum
- **split_start** (*int*) – Depends on the split_mode selected
- **split_interval** (*int*) – Depends on the split_mode selected

static set_retriever_time_offsets(*input_datastream_name: str*, *begin_offset: int*, *end_offset: int*)

This method should be called in your init_process_hook if you need to override the offsets per input datastream. By default, PCM only allows you to set global offsets that apply to all datastreams. If you need to change only one datastream, then you can do it via this method.

Parameters

- **input_datastream_name** (*str*) – The specific input datastream to change the processing interval for.
- **begin_offset** (*int*) – Seconds of data to fetch BEFORE the process interval starts
- **end_offset** (*int*) – Seconds of data to fetch AFTER the process interval ends

static shift_output_interval(*output_datastream_name: str*, *hours: int*)

This method should be called in your init_process_hook (i.e., before the processing loop begins) if you need to shift the output interval to account for the timezone difference at the data location. For example, if you shift the output interval by -6 hours at SGP, the file will be split at 6:00 a.m. GMT.

Parameters

- **output_datastream_name** (*str*) – The name of the output datastream whose file output will be shifted.
- **hours** (*int*) – Number of hours to shift

static shift_processing_interval(*seconds: int*)

This method should be called in your init_process_hook (i.e., before the processing loop begins) if you need to shift the processing interval.

Parameters **seconds** (*int*) – Number of seconds to shift

property site(*self*) → str

Get the site where this invocation of the process is running

Returns *str* – The site where this process is running

static sync_datasets(**args: xarray.Dataset*)

Sync the contents of one or more XArray.Datasets with the corresponding ADI data structure.

Important: This method MUST be called at the end of a hook function if any changes have been made to the XArray Dataset so that updates can be pushed back to ADI.

Important: This dataset must have been previously loaded via one of the get_*_dataset methods in order to have the correct embedded metadata to be able to sync to ADI. Specifically, this will include datastream name, coordinate system, dataset type, and obs_index.

See also:

- get_retrieved_dataset
- get_transformed_dataset
- get_output_dataset

Parameters ***args** (*xr.Dataset*) – One or more xr.Datasets to sync. **Note:** These datasets must have been previously loaded via one of the get_*_dataset methods in order to have the correct embedded metadata to be able to sync to ADI. Specifically, this will include datastream name, coordinate system, dataset type, and obs_index.

static variables_exist(*xr_dataset: xarray.Dataset*, *variable_names: numpy.ndarray = np.array([])*) → numpy.ndarray

Check if the given variables exist in the given dataset.

Parameters

- **xr_dataset** (*xr.Dataset*) – The dataset
- **variable_names** (*np.ndarray[str]*) – The variable names to check. Any array-like object can be provided, but ndarrays work best in order to easily select missing or existing variables from the results array (mask).

Returns *np.ndarray* – Array of same length as variable_names where each value is True or False. True if the variable exists. Use np.ndarray.all() to check if all the variables exist.

3.1.1.5 adi_py.utils

This module provides a variety of functions that are used by the Process class to serialize/deserialize between ADI and XArray. Most of these utility functions are used within Process class methods and are generally not intended to be called directly by developers when implementing a specific Process subclass.

3.1.1.5.1 Classes

DatastreamIdentifier	NamedTuple class that holds various information used to identify a specific

3.1.1.5.2 Functions

add_vartag_attributes	For the given ADI Variable, extract the source_ds_name and source_var_name
adi_hook_exception_handler	Python function decorator used to consistently handle exceptions in hooks
get_adi_var_as_dict	Convert the given adi variable to a dictionary that can be used to create
get_cds_type	For a given Python data value, convert the data type into the corresponding
get_dataset_dims	
get_dataset_vars	
get_datastream_files	Return the full path to each data file found for the given datastream
get_datastream_id	Gets the corresponding dataset id for the given datastream (input or output)
get_empty_ndarray_for_var	For the given ADI variable object, initialize an empty numpy ndarray data
get_time_data_as_datetime64	Get the time values from dsproc as seconds since 1970, then convert those
get_xr_dataset	Get an ADI dataset converted to an xarray.Dataset.
is_empty_function	Evaluates a given function to see if the code contains anything more than
sync_xarray	Carefully inspect the xr.Dataset and synchronize any changes back to the
sync_xr_dataset	Sync the contents of the given XArray.Dataset with the corresponding ADI
to_xarray	Convert the specified CDS.Group into an XArray dataset.

Function Descriptions

class adi_py.utils.**DatastreamIdentifier**

Bases: NamedTuple

NamedTuple class that holds various information used to identify a specific ADI dataset.

datastream_name :str

dsid :int

facility :str

site :str

---

adi_py.utils.**add_vartag_attributes**(*xr_var_attrs: Dict*, *adi_var: cds3.Var*)
For the given ADI Variable, extract the source_ds_name and source_var_name from the ADI var tags and add them to the attributes Dict for to be used for the Xarray variable.

Note: Currently we are not including the coordinate system and output targets as part of the XArray variable's attributes since these are unlikely to be changed. If a user creates a new variable, then they should call the corresponding Process methods assign_coordinate_system_to_variable or assign_output_datastream_to_variable to add the new variable to the designated coordinate system or output datastream, respectively.

Parameters

- **xr_var_attrs** (*Dict*) – A Dictionary of attributes to be assigned to the XArray variable.
- **adi_var** (*cds3.Var*) – The original ADI variable object.

adi_py.utils.**adi_hook_exception_handler**(*hook_func: Callable*, *pre_hook_func: Callable = None*, *post_hook_func: Callable = None*) → Callable
Python function decorator used to consistently handle exceptions in hooks so that they return the proper integer value to ADI core. Also used to ensure that consistent logging and debug dumps happen for hook methods.

Parameters

- **hook_func** (*Callable*) – The original hook function implemented by the developer.
- **pre_hook_func** (*Callable*) – An optional function to be invoked right before the hook function (i.e., to do debug dumps)
- **post_hook_func** (*Callable*) – An optional function to be invoked right after the hook function (i.e., to do debug dumps

Returns *Callable* – Decorator function that wraps the original hook function to provide built-in, ADI-compliant logging and exception handling.

adi_py.utils.**get_adi_var_as_dict**(*adi_var: cds3.Var*) → Dict
Convert the given adi variable to a dictionary that can be used to create an xarray dataarray.

Parameters **adi_var** (*cds3.Var*) – An ADI variable object

Returns *Dict* – A Dictionary representation of the variable that can be used in the XArray.DataArray constructor to initialize a corresponding XArray variable.

adi_py.utils.**get_cds_type**(*value: Any*) → int
For a given Python data value, convert the data type into the corresponding ADI CDS data type.

Parameters **value** (*Any*) – Can be a single value, a List of values, or a numpy.ndarray of values.

Returns *int* – The corresponding CDS data type

adi_py.utils.**get_dataset_dims**(*adi_dataset: cds3.Group*) → List[cds3.Dim]

adi_py.utils.**get_dataset_vars**(*adi_dataset: cds3.Group*) → List[cds3.Var]

adi_py.utils.**get_datastream_files**(*dsid: int*, *begin_date: int*, *end_date: int*) → List[str]
Return the full path to each data file found for the given datastream and time range.

Parameters

- **dsid** (*int*) – the datastream id (call get_dsid() to retrieve)
- **begin_date** (*int*) – the begin timestamp of the current processing interval (seconds since 1970)
- **end_date** (*int*) – the end timestamp of the current processing interval (seconds since 1970)

Returns *List[str]* – A list of file paths that match the datastream query.

adi_py.utils.**get_datastream_id**(*datastream_name: str*, *site: str = None*, *facility: str = None*, *dataset_type: adi_py.constants.ADIDatasetType = None*) → Optional[int]

Gets the corresponding dataset id for the given datastream (input or output)

Parameters

- **datastream_name** (*str*) – The name of the datastream to find
- **site** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Site is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by site.
- **facility** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Facility is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by facility.
- **dataset_type** (*ADIDatasetType*) – The type of the dataset to convert (RETRIEVED, TRANSFORMED, OUTPUT)

Returns *Optional[int]* – The dataset id or None if not found

adi_py.utils.**get_empty_ndarray_for_var**(*adi_var: cds3.Var*, *attrs: Dict = None*) → numpy.ndarray

For the given ADI variable object, initialize an empty numpy ndarray data array with the correct shape and data type. All values will be filled with the appropriate fill value. The rules for selecting a fill value are as follows:

- If this is a qc variable, the missing value bit flag will be used. If no missing value bit, then the failed transformation bit flag will be used. If no transformation failed bit, then use _FillValue. If no _FillValue, then use the netcdf default fill value for integer data type.
- Else if a missing_value attribute is available, missing_value will be used
- Else if a _FillValue attribute is available, _FillValue will be used
- Else use the netcdf default fill value for the variable's data type

Parameters

- **adi_var** (*cds3.Var*) – The ADI variable object
- **attrs** (*Dict*) – A Dictionary of attributes that will be assigned to the variable when it is converted to XArray. If not provided, it will be created from the ADI variable's attrs.

Returns *np.ndarray* – An empty ndarray of the same shape as the variable.

adi_py.utils.**get_time_data_as_datetime64**(*time_var: cds3.Var*) → numpy.ndarray

Get the time values from dsproc as seconds since 1970, then convert those values to datetime64 with microsecond precision.

Parameters **time_var** (*cds3.Var*) – An ADI time variable object

Returns *np.ndarray* – An ndarray of the same shape as the variable with time values converted to the np.datetime64 data type with microsecond precision.

adi_py.utils.**get_xr_dataset**(*dataset_type:* adi_py.constants.ADIDatasetType, *dsid: int = None*, *datastream_name: str = None*, *site: str = None*, *facility: str = None*, *coordsys_name: str = None*) → Optional[Union[xarray.Dataset, List[xarray.Dataset]]]

Get an ADI dataset converted to an xarray.Dataset.

Parameters

- **dataset_type** (*ADIDatasetType*) – The type of the dataset to convert (RETRIEVED, TRANSFORMED, OUTPUT)
- **dsid** (*int*) – If the dsid is known, you can use it to look up the adi dataset. If it is not known, then use datastream_name, and optionally site/facility to identify the dataset.
- **datastream_name** (*str*) – The name of one of the process' datastreams as specified in the PCM.
- **coordsys_name** (*str*) – Optional parameter used only to find TRANSFORMED datasets. Must be a coordinate system specified in the PCM or None if no coordinate system was specified.
- **site** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Site is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by site.
- **facility** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Facility is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by facility.

Returns *Optional[Union[xr.Dataset, List[xr.Dataset]]]* – Most of the time, return a single xr.Dataset. If the process is using file-based processing or if there are multiple files for the same datastream and a dimensionality conflict prevented the files from being merged, then return a List[XArray.Dataset], one for each file. If no dataset exists for the specified datastream/site/fac/coord system, then return None.

adi_py.utils.**is_empty_function**(*func: Callable*) → bool

Evaluates a given function to see if the code contains anything more than doctrings and 'pass'. If not, it is considered an 'empty' function.

Parameters **func** (*Callable*) –

Returns *bool* – True if the function is empty, otherwise False.

adi_py.utils.**sync_xarray**(*xr_dataset: xarray.Dataset*, *adi_dataset: cds3.Group*)

Carefully inspect the xr.Dataset and synchronize any changes back to the given ADI dataset.

Parameters

- **xr_dataset** (*xr.Dataset*) – The XArray dataset to sync
- **adi_dataset** (*csd3.Group*) – The ADI dataset where changes will be applied

adi_py.utils.**sync_xr_dataset**(*xr_dataset: xarray.Dataset*)

Sync the contents of the given XArray.Dataset with the corresponding ADI data structure.

Parameters **xr_dataset** (*xr.Dataset*) – The xr.Dataset(s) to sync.

adi_py.utils.**to_xarray**(*adi_dataset: cds3.Group*) → xarray.Dataset

Convert the specified CDS.Group into an XArray dataset. Attributes will be copied, but the DataArrays for each variable will be backed by an np.ndarray that links directly to the C ADI data via np.PyArray_SimpleNewFromData

Parameters **adi_dataset** (*cds3.Group*) – An ADI dataset object.

Returns *xr.Dataset* – The corresponding XArray dataset object.

3.1.1.6 adi_py.xarray_accessors

3.1.1.6.1 Classes

ADIDataArrayAccessor	Used to apply special ADI functions to an xarray data array (i.e., variable)
ADIDatasetAccessor	Used to apply special ADI functions to an xarray dataset with the

class adi_py.xarray_accessors.**ADIDataArrayAccessor**(*xarray_obj*)

Used to apply special ADI functions to an xarray data array (i.e., variable) with the namespace 'adi'

Class Methods

assign_coordinate_system
assign_output_datastream
nsamples
source_ds_name
source_var_name

Method Descriptions

assign_coordinate_system(*self*, *coordinate_system_name: str*)

assign_output_datastream(*self*, *output_datastream_name: str*, *variable_name_in_datastream: str = None*)

property nsamples(*self*) → int

property source_ds_name(*self*) → str

property source_var_name(*self*) → str

class adi_py.xarray_accessors.**ADIDatasetAccessor**(*xarray_obj*)

Used to apply special ADI functions to an xarray dataset with the namespace 'adi'

Class Methods

add_qc_variable
add_variable
convert_units
drop_transform_metadata

continues on next page

Table 18 – continued from previous page

drop_variables
get_companion_transform_variable_names
get_qc_variable
record_qc_results
variables_exist

Method Descriptions

add_qc_variable(*self*, *variable_name: str*)

add_variable(*self*, *variable_name: str*, *dim_names: List[str]*, *data: numpy.ndarray*, *long_name: str = None*, *standard_name: str = None*, *units: str = None*, *valid_min=None*, *valid_max=None*, *missing_value: numpy.ndarray = None*, *fill_value=None*)

convert_units(*self*, *old_units: str*, *new_units: str*, *variable_names: List[str] = None*, *converter_function: Callable = None*)

drop_transform_metadata(*self*, *variable_names: List[str]*) → xarray.Dataset

drop_variables(*self*, *variable_names: List[str]*) → xarray.Dataset

get_companion_transform_variable_names(*self*, *variable_name: str*) → List[str]

get_qc_variable(*self*, *variable_name: str*)

record_qc_results(*self*, *variable_name: str*, *bit_number: int = None*, *test_results: numpy.ndarray = None*)

variables_exist(*self*, *variable_names: List[str] = []*) → numpy.ndarray

3.1.1.7 Classes

ADIAtts	
ADIDataArrayAccessor	Used to apply special ADI functions to an xarray data array (i.e., variable)
ADIDatasetAccessor	Used to apply special ADI functions to an xarray dataset with the
ADIDatasetType	Used to easily reference different types of ADI datasets.
ADILogger	This class provides python-like logging API facade around the dsproc
BitAssessment	Used to easily reference bit assessment values used in ADI QC
DatastreamIdentifier	NamedTuple class that holds various information used to identify a specific
Process	The base class for running an ADI process in Python. All Python processes
SpecialXrAttributes	Enumerates the special XArray variable attributes that are assigned

continues on next page

Table 19 – continued from previous page

SplitMode	Enumerates the split mode which is used to define the output file size
TransformAttributes	Used to easily reference transformation metadata attrs used in ADI QC

exception adi_py.**SkipProcessingIntervalException**

Bases: Exception

Processes should throw this exception if the current processing interval should be skipped. All other exceptions will be considered to fail the process.

Initialize self. See help(type(self)) for accurate signature.

---

class adi_py.**ADIAtts**

ANCILLARY_VARIABLES = ancillary_variables

DESCRIPTION = description

FILL_VALUE = ['_FillValue']

LONG_NAME = long_name

MISSING_VALUE = missing_value

STANDARD_NAME = standard_name

UNITS = units

VALID_MAX = valid_max

VALID_MIN = valid_min

---

class adi_py.**ADIDataArrayAccessor**(*xarray_obj*)

Used to apply special ADI functions to an xarray data array (i.e., variable) with the namespace 'adi'

Class Methods

assign_coordinate_system	
assign_output_datastream	
nsamples	
source_ds_name	
source_var_name	

Method Descriptions

assign_coordinate_system(*self*, *coordinate_system_name: str*)

assign_output_datastream(*self*, *output_datastream_name: str*, *variable_name_in_datastream: str = None*)

property nsamples(*self*) → int

property source_ds_name(*self*) → str

property source_var_name(*self*) → str

class adi_py.**ADIDatasetAccessor**(*xarray_obj*)

Used to apply special ADI functions to an xarray dataset with the namespace 'adi'

Class Methods

add_qc_variable
add_variable
convert_units
drop_transform_metadata
drop_variables
get_companion_transform_variable_names
get_qc_variable
record_qc_results
variables_exist

Method Descriptions

add_qc_variable(*self*, *variable_name: str*)

add_variable(*self*, *variable_name: str*, *dim_names: List[str]*, *data: numpy.ndarray*, *long_name: str = None*, *standard_name: str = None*, *units: str = None*, *valid_min=None*, *valid_max=None*, *missing_value: numpy.ndarray = None*, *fill_value=None*)

convert_units(*self*, *old_units: str*, *new_units: str*, *variable_names: List[str] = None*, *converter_function: Callable = None*)

drop_transform_metadata(*self*, *variable_names: List[str]*) → xarray.Dataset

drop_variables(*self*, *variable_names: List[str]*) → xarray.Dataset

get_companion_transform_variable_names(*self*, *variable_name: str*) → List[str]

get_qc_variable(*self*, *variable_name: str*)

record_qc_results(*self*, *variable_name: str*, *bit_number: int = None*, *test_results: numpy.ndarray = None*)

variables_exist(*self*, *variable_names: List[str] = []*) → numpy.ndarray

class adi_py.**ADIDatasetType**

Bases: enum.Enum

Used to easily reference different types of ADI datasets.

OUTPUT = 3

RETRIEVED = 1

TRANSFORMED = 2

---

class adi_py.**ADILogger**

This class provides python-like logging API facade around the dsproc logging methods.

Class Methods

debug	
error	
exception	Use this method to log the stack trace of any raised exception to the process's
info	
warning	

Method Descriptions

static debug(*message*, *debug_level=1*)

static error(*message*)

static exception(*message*)

Use this method to log the stack trace of any raised exception to the process's ADI log file.

Parameters **message** (-) – str An optional additional message to log, in addition to the stack trace.

static info(*message*)

static warning(*message*)

class adi_py.**BitAssessment**

Bases: enum.Enum

Used to easily reference bit assessment values used in ADI QC

BAD = Bad

INDETERMINATE = Indeterminate

---

class adi_py.**DatastreamIdentifier**

Bases: NamedTuple

NamedTuple class that holds various information used to identify a specific ADI dataset.

datastream_name :str

dsid :int

facility :str

site :str

---

class adi_py.**Process**

The base class for running an ADI process in Python. All Python processes should extend this class.

Class Methods

add_qc_variable	Add a companion qc variable for the given variable
add_variable	Create a new variable in the given xarray dataset with the specified dimensions,
assign_coordinate_system_to_variable	Assign the given variable to the designated ADI coordinate system.
assign_output_datastream_to_variable	Assign the given variable to the designated output datastream.
convert_units	For the specified variables, convert the units from old_units to new_units.
debug_level	Get the debug level passed on the command line when running the process.
drop_transform_metadata	This method removes all associated companion variables that are generated
drop_variables	This method removes the given variables plus all associated companion
facility	Get the facility where this invocation of the process is running
find_retrieved_variable	Find the input datastream where the given retrieved variable came
finish_process_hook	This hook will be called once just after the main data processing loop finishes. This function should be used
get_bad_qc_mask	Get a mask of same shape as the variable's data which contains True values
get_companion_transform_variable_names	For the given variable, get a list of the companion/ancillary variables
get_datastream_files	See *utils.get_datastream_files()*
get_dsid	Gets the corresponding dataset id for the given datastream (input or output)
get_missing_value_mask	Get True/False mask of same shape as passed variable(s) which is used to
get_non_missing_value_mask	Get a True/False mask of same shape as passed variable(s) that is used to
get_nsamples	Get the ADI sample count for the given variable (i.e., the length
get_output_dataset	Get an ADI output dataset converted to an xarray.Dataset.
get_output_dataset_by_dsid	
get_qc_variable	Return the companion qc variable for the given data variable.
get_quicklooks_file_name	Create a properly formatted file name where a quicklooks plot should be
get_retrieved_dataset	Get an ADI retrieved dataset converted to an xarray.Dataset.
get_retrieved_dataset_by_dsid	

continues on next page

Table 28 – continued from previous page

get_source_ds_name	For the given variable, get name of the input datastream
get_source_var_name	For the given variable, get the name of the variable
get_transformed_dataset	Get an ADI transformed dataset converted to an xarray.Dataset.
get_transformed_dataset_by_dsid	
include_debug_dumps	Setting controlling whether this process should provide debug dumps of the
init_process_hook	This hook will will be called once just before the main data processing loop begins and before the initial
location	Get the location where this invocation of the process is running.
post_retrieval_hook	This hook will will be called once per processing interval just after data retrieval,
post_transform_hook	This hook will be called once per processing interval just after data
pre_retrieval_hook	This hook will will be called once per processing interval just prior to data retrieval.
pre_transform_hook	This hook will be called once per processing interval just prior to data
process_data_hook	This hook will be called once per processing interval just after the output
process_model	The processing model to use. It can be one of:
process_name	The name of the process that is currently being run.
process_names	The name(s) of the process(es) that could run this code. Subclasses must
process_version	The version of this process's code. Subclasses must define the
quicklook_hook	This hook will be called once per processing interval just after all data
record_qc_results	For the given variable, add bitwise test results to the companion qc
rollup_qc	ADI setting controlling whether all the qc bits are rolled up into a
run	Run the process.
set_datastream_flags	Apply a set of ADI control flags to a datastream as identified by the
set_datastream_split_mode	This method should be called in your init_process_hook if you need to
set_retriever_time_offsets	This method should be called in your init_process_hook if you need to override
shift_output_interval	This method should be called in your init_process_hook (i.e., before the
shift_processing_interval	This method should be called in your init_process_hook (i.e., before the
site	Get the site where this invocation of the process is running
sync_datasets	Sync the contents of one or more XArray.Datasets with the corresponding ADI
variables_exist	Check if the given variables exist in the given dataset.

Method Descriptions

static add_qc_variable(*dataset: xarray.Dataset*, *variable_name: str*)

Add a companion qc variable for the given variable

Parameters

- **dataset** (*xr.Dataset*) –
- **variable_name** (*str*) –

Returns The newly created DataArray

static add_variable(*dataset: xarray.Dataset*, *variable_name: str*, *dim_names: List[str]*, *data: numpy.ndarray*, *long_name: str = None*, *standard_name: str = None*, *units: str = None*, *valid_min: Any = None*, *valid_max: Any = None*, *missing_value: numpy.ndarray = None*, *fill_value: Any = None*)

Create a new variable in the given xarray dataset with the specified dimensions, data, and attributes.

Important: If you want to add the created variable to a given coordinate system, then you follow this with a call to assign_coordinate_system_to_variable. Similarly, if you want to add the created variable to a given output datastream, then you should follow this with a call to assign_output_datastream_to_variable

---

See also:

- assign_coordinate_system_to_variable
- assign_output_datastream_to_variable

Parameters

- **dataset** (*xr.Dataset*) – The xarray dataset to add the new variable to
- **variable_name** (*str*) – The name of the variable
- **dim_names** (*List[str]*) – A list of dimension names for the variable
- **data** (*np.ndarray*) – A multidimensional array of the variable's data Must have the same shape as the dimensions.
- **long_name** (*str*) – The long_name attribute for the variable
- **standard_name** (*str*) – The standard_name attribute for the variable
- **units** (*str*) – The units attribute for the variable
- **valid_min** (*Any*) – The valid_min attribute for the variable. Must be the same data type as the variable.
- **valid_max** (*Any*) – The valid_max attribute for the variable Must be the same data type as the variable.
- **missing_value** (*np.ndarray*) – An array of possible missing_value attributes for the variable. Must be the same data type as the variable.
- **()** (*fill_value*) – The fill_value attribute for the variable. Must be the same data type as the variable.

Returns The newly created variable (i.e., xr.DataArray object)

static assign_coordinate_system_to_variable(*variable: xarray.DataArray*, *coordinate_system_name: str*)

Assign the given variable to the designated ADI coordinate system.

Parameters

- **variable** (*xr.DataArray*) – A data variable from an xarray dataset
- **coordinate_system_name** (*str*) – The name of one of the process's coordinate systems as specified in the PCM process definition.

static assign_output_datastream_to_variable(*variable: xarray.DataArray*, *output_datastream_name: str*, *variable_name_in_datastream: str = None*)

Assign the given variable to the designated output datastream.

Parameters

- **variable** (*xr.DataArray*) – A data variable from an xarray dataset
- **output_datastream_name** (*str*) – An output datastream name as specified in PCM process definition
- **variable_name_in_datastream** (*str*) – The name of the variable as it should appear in the output datastream. If not specified, then the name of the given variable will be used.

static convert_units(*xr_datasets: List[xarray.Dataset]*, *old_units: str*, *new_units: str*, *variable_names: List[str] = None*, *converter_function: Callable = None*)

For the specified variables, convert the units from old_units to new_units. For applicable variables, this conversion will include changing the units attribute value and optionally converting all the data values if a converter function is provided.

This method is needed for special cases where the units conversion is not supported by udunits and the default ADI converters.

Parameters

- **xr_datasets** (*List[xr.Dataset]*) – One or more xarray datasets upon which to apply the conversion
- **old_units** (*str*) – The old units (e.g., 'degree F')
- **new_units** (*str*) – The new units (e.g., 'K')
- **variable_names** (*List[str]*) – A list of specific variable names to convert. If not specified, it converts all variables with the given old_units to new_units.
- **()** (*converter_function*) – A function to run on an Xarray variable (i.e., DataArray that converts a variable's values from old_units to new_units. If not specified, then only the units attribute value will be changed. This could happen if we just want to change the units attribute value because of a typo.

  The function should take one parameter, an xarray.DataArray, and operate in place on the variable's values.

property debug_level(*self*) → int

Get the debug level passed on the command line when running the process.

Returns *int* – the debug level

static drop_transform_metadata(*dataset: xarray.Dataset*, *variable_names: List[str]*) → xarray.Dataset

This method removes all associated companion variables that are generated byt the transformation process (if they exist), as well as transformation attributes, but it does not remove the original variable.

Parameters

- **dataset** (*xr.Dataset*) – The dataset containing the transformed variables.
- **variable_names** (*List[str]*) – The variable names for which to remove transformation metadata.

Returns *xr.Dataset* – A new dataset with the transform companion variables and metadata removed.

static drop_variables(*dataset: xarray.Dataset*, *variable_names: List[str]*) → xarray.Dataset

This method removes the given variables plus all associated companion variables that were added as part of the transformation step (if they exist).

Parameters

- **dataset** (*xr.Dataset*) – The dataset containing the given variables.
- **variable_names** (*List[str]*) – The variable names to remove.

Returns *xr.Dataset* – A new dataset with the given variables and their transform companion variables removed.

property facility(*self*) → str

Get the facility where this invocation of the process is running

Returns *str* – The facility where this process is running

static find_retrieved_variable(*retrieved_variable_name*) → Optional[*adi_py.utils.DatastreamIdentifier*]

Find the input datastream where the given retrieved variable came from. We may need this if there are complex retrieval rules and the given variable may be retrieved from different datastreams depending upon the site/facility where this process runs. We need to get the DatastreamIdentifier so we can load the correct xarray dataset if we need to modify the data values.

Parameters **retrieved_variable_name** (*str*) – The name of the retrieved variable to find

Returns A DatastreamIdentifier containing all the information needed to look up the given dataset or None if the retrieved variable was not found.

finish_process_hook(*self*)

This hook will be called once just after the main data processing loop finishes. This function should be used to clean up any temporary files used.

static get_bad_qc_mask(*dataset: xarray.Dataset*, *variable_name: str*, *include_indeterminate: bool = False*, *bit_numbers: List[int] = None*) → numpy.ndarray

Get a mask of same shape as the variable's data which contains True values for each data point that has a corresponding bad qc bit set.

Parameters

- **dataset** (*xr.Dataset*) – The dataset containing the variables
- **variable_name** (*str*) – The variable name to check qc for
- **include_indeterminate** (*bool*) – Whether to include indeterminate bits when determining the mask. By default this is False and only bad bits are used to compute the mask.
- **bit_numbers** (*List(int)*) – The specific bit numbers to include in the qc check (i.e., 1,2,3,4, etc.). Note that if not specified, all bits will be used to compute the mask.

Returns *np.ndarray* – An array of same shape as the variable consisting of True/False values, where each True indicates that the corresponding data point had bad (or indeterminate if

include_indeterminate is specified) qc for the specified bit numbers (all bits if bit_numbers not specified).

static get_companion_transform_variable_names(*dataset: xarray.Dataset*, *variable_name: str*) → List[str]

For the given variable, get a list of the companion/ancillary variables that were added as a result of the ADI transformation.

Parameters

- **dataset** (*xr.Dataset*) – The dataset
- **variable_name** (*str*) – The name of a data variable in the dataset

Returns A list of string companion variable names that were created from the transform engine. This is used for cleaning up associated variables when a variable is deleted from a dataset.

static get_datastream_files(*datastream_name: str*, *begin_date: int*, *end_date: int*) → List[str]

See *utils.get_datastream_files()*

static get_dsid(*datastream_name: str*, *site: str = None*, *facility: str = None*, *dataset_type:* adi_py.constants.ADIDatasetType *= None*) → Optional[int]

Gets the corresponding dataset id for the given datastream (input or output)

Parameters

- **datastream_name** (*str*) – The name of the datastream to find
- **site** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Site is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by site.
- **facility** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Facility is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by facility.
- **dataset_type** (ADIDatasetType) – The type of the dataset to convert (RETRIEVED, TRANSFORMED, OUTPUT)

Returns *Optional[int]* – The dataset id or None if not found

static get_missing_value_mask(**args*) → xarray.DataArray

Get True/False mask of same shape as passed variable(s) which is used to select data points for which one or more of the values of any of the specified variables are missing.

Parameters ***args** (*xr.DataArray*) – Pass one or more xarray variables to check for missing values. All variables in the list must have the same shape.

Returns *xr.DataArray* – An array of True/False values of the same shape as the input variables where each True represents the case where one or more of the variables has a missing_value at that index.

static get_non_missing_value_mask(**args*) → xarray.DataArray

Get a True/False mask of same shape as passed variable(s) that is used to select data points for which none of the values of any of the specified variables are missing.

Parameters ***args** (*xr.DataArray*) – Pass one or more xarray variables to check. All variables in the list must have the same shape.

Returns *xr.DataArray* – An array of True/False values of the same shape as the input variables where each True represents the case where all variables passed in have non-missing value data at that index.

static get_nsamples(*xr_var: xarray.DataArray*) → int

Get the ADI sample count for the given variable (i.e., the length of the first dimension or 1 if the variable has no dimensions)

Parameters **xr_var** (*xr.DataArray*) –

Returns *int* – The ADI sample count

static get_output_dataset(*output_datastream_name: str*) → Union[xarray.Dataset, List[xarray.Dataset]]

Get an ADI output dataset converted to an xarray.Dataset.

Parameters **output_datastream_name** (*str*) – The name of one of the process' output datastreams as specified in the PCM.

Returns *Union[xr.Dataset, List[xr.Dataset]]* – Most of the time, return a single xr.Dataset. If the process is using file-based processing or if there are multiple files for the same datastream and a dimensionality conflict prevented the files from being merged, then return a List[XArray.Dataset], one for each file.

static get_output_dataset_by_dsid(*dsid: int*) → Union[xarray.Dataset, List[xarray.Dataset]]

static get_qc_variable(*dataset: xarray.Dataset*, *variable_name: str*) → xarray.DataArray

Return the companion qc variable for the given data variable.

Parameters

- **dataset** (*xr.Dataset*) –
- **variable_name** (*str*) –

Returns *xr.DataArray* – The companion qc variable or None if it doesn't exist

get_quicklooks_file_name(*self*, *datastream_name: str*, *begin_date: int*, *description: str = None*, *ext: str = 'png'*, *mkdirs: bool = False*)

Create a properly formatted file name where a quicklooks plot should be saved for the given processing interval. For example:

${QUICKLOOK_DATA}/ena/enamfrsrcldod1minC1.c1/2021/01/01/enamfrsrcldod1minC1.c1.20210101.000000.lwp.png

Parameters

- **datastream_name** (*str*) – The name of the datastream which this plot applies to. For example, *mfrsrcldod1min.c1*
- **begin_date** (*int*) – The begin timestamp of the current processing interval as passed to the quicklook hook function
- **description** (*str*) – The description of the plot to be used in the file name For example, in the file *enamfrsrcldod1minC1.c1.20210101.000000.lwp.png*, the description is 'lwp'.
- **ext** (*str*) – The file extension for the image. Default is 'png'
- **mkdirs** (*bool*) – If True, then the folder path to the quicklooks file will be automatically created if it does not exist. Default is False.

Returns *str* – The full path to where the quicklooks file should be saved.

static get_retrieved_dataset(*input_datastream_name: str*, *site: str = None*, *facility: str = None*) → Union[xarray.Dataset, List[xarray.Dataset]]

Get an ADI retrieved dataset converted to an xarray.Dataset.

Parameters

- **input_datastream_name** (*str*) – The name of one of the process' input datastreams as specified in the PCM.
- **site** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Site is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by site.
- **facility** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Facility is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by facility.

Returns *Union[xr.Dataset, List[xr.Dataset]]* – Most of the time, return a single xr.Dataset. If the process is using file-based processing or if there are multiple files for the same datastream and a dimensionality conflict prevented the files from being merged, then return a List[XArray.Dataset], one for each file.

static get_retrieved_dataset_by_dsid(*dsid: int*) → Union[xarray.Dataset, List[xarray.Dataset]]

static get_source_ds_name(*xr_var: xarray.DataArray*) → str

For the given variable, get name of the input datastream where it came from :param xr_var: :type xr_var: xr.DataArray

Returns: str

static get_source_var_name(*xr_var: xarray.DataArray*) → str

For the given variable, get the name of the variable used in the input datastream :param xr_var: :type xr_var: xr.DataArray

Returns: str

static get_transformed_dataset(*input_datastream_name: str*, *coordinate_system_name: str*, *site: str = None*, *facility: str = None*) → Union[xarray.Dataset, List[xarray.Dataset]]

Get an ADI transformed dataset converted to an xarray.Dataset.

Parameters

- **input_datastream_name** (*str*) – The name of one of the process' input datastreams as specified in the PCM.
- **coordinate_system_name** (*str*) – A coordinate system specified in the PCM or None if no coordinate system was specified.
- **site** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Site is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by site.
- **facility** (*str*) – Optional parameter used only to find some input datasets (RETRIEVED or TRANSFORMED). Facility is only required if the retrieval rules in the PCM specify two different rules for the same datastream that differ by facility.

Returns *Union[xr.Dataset, List[xr.Dataset]]* – Most of the time, return a single xr.Dataset. If the process is using file-based processing or if there are multiple files for the same datastream and a dimensionality conflict prevented the files from being merged, then return a List[XArray.Dataset], one for each file.

static get_transformed_dataset_by_dsid(*dsid: int*, *coordinate_system_name: str*) → Union[xarray.Dataset, List[xarray.Dataset]]

property include_debug_dumps(*self*) → bool

Setting controlling whether this process should provide debug dumps of the data after each hook.

Returns *bool* – Whether debug dumps should be automatically included. If True and debug level is > 1, then debug dumps will be performed automatically before and after each code hook.

init_process_hook(*self*)
This hook will will be called once just before the main data processing loop begins and before the initial database connection is closed.

property location(*self*) → dsproc3.PyProcLoc
Get the location where this invocation of the process is running.

Returns *dsproc.PyProcLoc* – A class containing the alt, lat, and lon where the process is running.

post_retrieval_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will will be called once per processing interval just after data retrieval, but before the retrieved observations are merged and QC is applied.

Parameters

- **begin_date** (*int*) – the begin time of the current processing interval
- **end_date** (*int*) – the end time of the current processing interval

post_transform_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will be called once per processing interval just after data transformation, but before the output datasets are created.

Parameters

- **begin_date** (*int*) – the begin time of the current processing interval
- **end_date** (*int*) – the end time of the current processing interval

pre_retrieval_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will will be called once per processing interval just prior to data retrieval.

Parameters

- **begin_date** (*- int*) – the begin time of the current processing interval
- **end_date** (*- int*) – the end time of the current processing interval

pre_transform_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will be called once per processing interval just prior to data transformation,and after the retrieved observations are merged and QC is applied.

Parameters

- **begin_date** (*int*) – the begin time of the current processing interval
- **end_date** (*int*) – the end time of the current processing interval

process_data_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will be called once per processing interval just after the output datasets are created, but before they are stored to disk.

Parameters

- **begin_date** (*int*) – the begin time of the current processing interval
- **end_date** (*int*) – the end time of the current processing interval

property process_model(*self*) → int
The processing model to use. It can be one of:

dsproc.PM_GENERIC dsproc.PM_INGEST dsproc.PM_RETRIEVER_INGEST
dsproc.PM_RETRIEVER_VAP dsproc.PM_TRANSFORM_INGEST
dsproc.PM_TRANSFORM_VAP

Default value is PM_TRANSFORM_VAP. Subclasses can override in their constructor.

Returns *int* – The processing modelj (see dsproc.ProcModel cdeftype)

property process_name(*self*) → str
The name of the process that is currently being run.

Returns *str* – the name of the current process

property process_names(*self*) → List[str]
The name(s) of the process(es) that could run this code. Subclasses must define the self._names field in their constructor.

Returns *List[str]* – One or more process names

property process_version(*self*) → str
The version of this process's code. Subclasses must define the self._process_version field in their constructor.

Returns *str* – The process version

quicklook_hook(*self*, *begin_date: int*, *end_date: int*)
This hook will be called once per processing interval just after all data is stored.

Parameters

- **begin_date** (*int*) – the begin timestamp of the current processing interval
- **end_date** (*int*) – the end timestamp of the current processing interval

static record_qc_results(*xr_dataset: xarray.Dataset*, *variable_name: str*, *bit_number: int = None*, *test_results: numpy.ndarray = None*)
For the given variable, add bitwise test results to the companion qc variable for the given test.

Parameters

- **xr_dataset** (*xr.Dataset*) – The xr dataset
- **variable_name** (*str*) – The name of the data variable (e.g., rh_ambient).
- **bit_number** (*int*) – The bit/test number to record Note that bit numbering starts at 1 (i.e., 1, 2, 3, 4, etc.)
- **test_results** (*np.ndarray*) – A ndarray mask of the same shape as the variable with True/False values for each data point. True means the test failed for that data point. False means the test passed for that data point.

property rollup_qc(*self*) → bool
ADI setting controlling whether all the qc bits are rolled up into a single 0/1 value or not.

Returns *bool* – Whether this process should rollup qc or not

run(*self*) → int
Run the process.

Returns

int – The processing status:

- 1 if an error occurred
- 0 if successful

static set_datastream_flags(*dsid: int*, *flags: int*)

Apply a set of ADI control flags to a datastream as identified by the dsid. Multiple flags can be combined together using a bitwise OR (e.g., dsproc.DS_STANDARD_QC | dsproc.DS_FILTER_NANS). The allowed flags are identified below:

dsproc.DS_STANDARD_QC = Apply standard QC before storing a dataset.

dsproc.DS_FILTER_NANS = Replace NaN and Inf values with missing values before storing a dataset.

dsproc.DS_OVERLAP_CHECK = Check for overlap with previously processed data. This flag will be ignored and the overlap check will be skipped if reprocessing mode is enabled, or asynchronous processing mode is enabled.

dsproc.DS_PRESERVE_OBS = Preserve distinct observations when retrieving data. Only observations that start within the current processing interval will be read in.

dsproc.DS_DISABLE_MERGE = Do not merge multiple observations in retrieved data. Only data for the current processing interval will be read in.

dsproc.DS_SKIP_TRANSFORM = Skip the transformation logic for all variables in this datastream.

dsproc.DS_ROLLUP_TRANS_QC = Consolidate the transformation QC bits for all variables when mapped to the output datasets.

dsproc.DS_SCAN_MODE = Enable scan mode for datastream that are not expected to be continuous. This prevents warning messages from being generated when data is not found within a processing interval. Instead, a message will be written to the log file indicating that the procesing interval was skipped.

dsproc.DS_OBS_LOOP = Loop over observations instead of time intervals. This also sets the DS_PRESERVE_OBS flag.

dsproc.DS_FILTER_VERSIONED_FILES = Check for files with .v# version extensions and filter out lower versioned files. Files without a version extension take precedence.

Call self.get_dsid() to obtain the dsid value for a specific datastream. If the flags value is < 0, then the following default flags will be set:

dsprc.DS_STANDARD_QC 'b' level datastreams dsproc.DS_FILTER_NANS 'a' and 'b' level datastreams dsproc.DS_OVERLAP_CHECK all output datastreams dsproc.DS_FILTER_VERSIONED_FILES input datastreams that are not level '0'

Parameters

- **dsid** (*int*) – Datastream ID
- **flags** (*int*) – Flags to set

Returns *int* – The processing modelj (see dsproc.ProcModel cdeftype)

static set_datastream_split_mode(*output_datastream_name: str*, *split_mode:* adi_py.constants.SplitMode, *split_start: int*, *split_interval: int*)

This method should be called in your init_process_hook if you need to change the size of the output file for a given datastream. For example, to create monthly output files.

Parameters

- **output_datastream_name** (*str*) – The name of the output datastream whose file output size will be changed.
- **split_mode** (SplitMode) – One of the options from the SplitMode enum

- **split_start** (*int*) – Depends on the split_mode selected
- **split_interval** (*int*) – Depends on the split_mode selected

static set_retriever_time_offsets(*input_datastream_name: str*, *begin_offset: int*, *end_offset: int*)
This method should be called in your init_process_hook if you need to override the offsets per input datastream. By default, PCM only allows you to set global offsets that apply to all datastreams. If you need to change only one datastream, then you can do it via this method.

Parameters

- **input_datastream_name** (*str*) – The specific input datastream to change the processing interval for.
- **begin_offset** (*int*) – Seconds of data to fetch BEFORE the process interval starts
- **end_offset** (*int*) – Seconds of data to fetch AFTER the process interval ends

static shift_output_interval(*output_datastream_name: str*, *hours: int*)
This method should be called in your init_process_hook (i.e., before the processing loop begins) if you need to shift the output interval to account for the timezone difference at the data location. For example, if you shift the output interval by -6 hours at SGP, the file will be split at 6:00 a.m. GMT.

Parameters

- **output_datastream_name** (*str*) – The name of the output datastream whose file output will be shifted.
- **hours** (*int*) – Number of hours to shift

static shift_processing_interval(*seconds: int*)
This method should be called in your init_process_hook (i.e., before the processing loop begins) if you need to shift the processing interval.

Parameters **seconds** (*int*) – Number of seconds to shift

property site(*self*) → str
Get the site where this invocation of the process is running

Returns *str* – The site where this process is running

static sync_datasets(**args: xarray.Dataset*)
Sync the contents of one or more XArray.Datasets with the corresponding ADI data structure.

Important: This method MUST be called at the end of a hook function if any changes have been made to the XArray Dataset so that updates can be pushed back to ADI.

Important: This dataset must have been previously loaded via one of the get_*_dataset methods in order to have the correct embedded metadata to be able to sync to ADI. Specifically, this will include datastream name, coordinate system, dataset type, and obs_index.

See also:

- get_retrieved_dataset
- get_transformed_dataset
- get_output_dataset

Parameters ***args** (*xr.Dataset*) – One or more xr.Datasets to sync. **Note:** These datasets must have been previously loaded via one of the get_*_dataset methods in order to have the correct embedded metadata to be able to sync to ADI. Specifically, this will include datastream name, coordinate system, dataset type, and obs_index.

static variables_exist(*xr_dataset: xarray.Dataset*, *variable_names: numpy.ndarray = np.array([])*) → numpy.ndarray

Check if the given variables exist in the given dataset.

Parameters

- **xr_dataset** (*xr.Dataset*) – The dataset
- **variable_names** (*np.ndarray[str]*) – The variable names to check. Any array-like object can be provided, but ndarrays work best in order to easily select missing or existing variables from the results array (mask).

Returns *np.ndarray* – Array of same length as variable_names where each value is True or False. True if the variable exists. Use np.ndarray.all() to check if all the variables exist.

class adi_py.**SpecialXrAttributes**

Enumerates the special XArray variable attributes that are assigned temporarily to help sync data between xarray and adi.

COORDINATE_SYSTEM = __coordsys_name

DATASET_TYPE = __dataset_type

DATASTREAM_DSID = __datastream_dsid

OBS_INDEX = __obs_index

OUTPUT_TARGETS = __output_targets

SOURCE_DS_NAME = __source_ds_name

SOURCE_VAR_NAME = __source_var_name

---

class adi_py.**SplitMode**

Bases: enum.Enum

Enumerates the split mode which is used to define the output file size used when storing values. See dsproc.set_datastream_split_mode().

SPLIT_ON_DAYS

SPLIT_ON_HOURS

SPLIT_ON_MONTHS

SPLIT_ON_STORE

---

class adi_py.**TransformAttributes**

Used to easily reference transformation metadata attrs used in ADI QC

CELL_TRANSFORM = cell_transform

CHAPTER

FOUR

WELCOME TO ADI PYTHON

This site provides documentation for implementing ARM processes via Python. Specifically it describes the new Python ADI bindings which are fully integrated with XArray to provide a more Pythonic programming experience.

Please follow the steps in *Getting Started* to get started developing a project with the new Python API.

Note

The new ADI Python bindings are currently only deployed on dev servers for beta testing. In addition, the API and examples currently only support VAPs. Ingest support is coming soon.

PYTHON MODULE INDEX

a

INDEX

D

E

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W