
libera_utils

Release 2.5.2

Libera SDC Team

Mar 06, 2025

CONTENTS:

1	User Docs	3
2	Developer Docs	17
3	API	31
4	Version Changes	209
5	Libera Science Data Processing Utilities	213
6	Developing Libera Utils	215
	Python Module Index	217
	Index	219

Version: 2.5.2

This section is for developers including Libera Utils in their code. If you are a Libera L2 algorithm developer, you are in the right place.

1.1 Basic Usage

1.1.1 Command Line Interface

The CLI is installed as an executable in your virtual environment during installation of `libera_utils`.

Top Level Command `libera-utils`

This is the top level command that contains all the nested sub-commands.

```
usage: libera-utils [-h] [--version] {make-kernel,ecr-upload,step-function-trigger} ...

Libera SDC utilities CLI

options:
  -h, --help            show this help message and exit
  --version              print current version of the CLI

subcommands:
  sub-commands for libera-utils CLI

  {make-kernel,ecr-upload,step-function-trigger}
  make-kernel           generate SPICE kernel from telemetry data
  ecr-upload            Upload docker image to ECR repository for a specific algorithm
  step-function-trigger
                       Manually trigger a specific step function
```

Sub-Command `ecr-upload`

This is a tool to upload a docker image to AWS ECR. The image name and tag identify the local docker image while the `--ecr-image-tags` option specifies the tags to apply to the image in the ECR (remote tags). If `--ecr-image-tags` is not provided, only the latest tag is applied by default. If `--ecr-image-tags` is specified, you must include latest explicitly.

```
usage: libera-utils ecr-upload [-h] [--ecr-image-tags ECR_IMAGE_TAGS [ECR_IMAGE_TAGS ...
  ↪]] [--ignore-docker-config] image_name image_tag algorithm_name
```

(continues on next page)

(continued from previous page)

```
positional arguments:
  image_name      Image name of image to upload (image-name:image-tag)
  image_tag       Image tag of image to upload (image-name:image-tag)
  algorithm_name  Algorithm name that matches an ECR repo name, inputs to names:
                 ['l2-cloud-fraction', 'l2-ssw-toa', 'libera-adms', 'l2-ssw-
                 ↪surface-flux', 'l2-far-ir-toa-flux', 'l1c-unfiltered',
                 'spice-azel', 'spice-jpss', 'l1b-rad', 'l1b-cam']

options:
  -h, --help          show this help message and exit
  --ecr-image-tags ECR_IMAGE_TAGS [ECR_IMAGE_TAGS ...]
                     List of tags to apply to the uploaded image in the ECR (e.g. `--
                 ↪ecr-image-tags latest 1.3.4`) Note, latest is applied if this option is not set. If it
                 ↪is set, you must specify
                     latest if you want it tagged as such in the ECR.
  --ignore-docker-config
                     Ignore the standard docker config.json to bypass the credential
                 ↪store
```

Example usage:

```
libera-utils ecr-upload recently-built-ssw-sfc-flux latest l2-ssw-surface-flux --ecr-
                 ↪image-tags latest --ignore-docker-config
```

To get a list of specific algorithm names allowed in this command, run `libera-utils ecr-upload -h`

Sub-Command `make-kernel jpss-spk`

```
usage: libera-utils make-kernel jpss-spk [-h] --outdir OUTDIR [--overwrite] [-v] packet_
                 ↪data_filepaths [packet_data_filepaths ...]
```

```
positional arguments:
  packet_data_filepaths
                     paths to L0 packet files
```

```
options:
  -h, --help          show this help message and exit
  --outdir OUTDIR, -o OUTDIR
                     output directory for generated SPK
  --overwrite         force overwriting an existing kernel if it exists
  -v, --verbose       set DEBUG level logging output
```

Sub-Command `make-kernel jpss-ck`

```
usage: libera-utils make-kernel jpss-ck [-h] --outdir OUTDIR [--overwrite] [-v] packet_
                 ↪data_filepaths [packet_data_filepaths ...]
```

```
positional arguments:
  packet_data_filepaths
                     paths to L0 packet files
```

```
options:
```

(continues on next page)

(continued from previous page)

```

-h, --help          show this help message and exit
--outdir OUTDIR, -o OUTDIR
                    output directory for generated CK
--overwrite         force overwriting an existing kernel if it exists
-v, --verbose       set DEBUG level logging output

```

Sub-Command make-kernel azel-ck

```
usage: libera-utils make-kernel azel-ck [-h] [--azimuth] [--elevation] --outdir OUTDIR [-
↳ --overwrite] [--csv] [-v] packet_data_filepaths [packet_data_filepaths ...]
```

positional arguments:

```

packet_data_filepaths
                    paths to L0 packet files

```

options:

```

-h, --help          show this help message and exit
--azimuth           generate ck for Azimuth
--elevation         generate ck for Elevation
--outdir OUTDIR, -o OUTDIR
                    output directory for generated CK
--overwrite         force overwriting an existing kernel if it exists
--csv              the provided Az and El packet_data_filepaths are ASCII csv files.
↳ instead of binary CCSDS
-v, --verbose       set DEBUG level logging output (otherwise set by LIBSDP_STREAM_
↳ LOG_LEVEL)

```

Sub-Command step-function-trigger

```
usage: libera-utils step-function-trigger [-h] [-w] [-v] algorithm_name applicable_day
```

positional arguments:

```

algorithm_name      Algorithm name you want to run
applicable_day      Day of data you want to rerun. Format of date: YYYY-MM-DD

```

options:

```

-h, --help          show this help message and exit
-w, --wait_for_finish
                    Block command line until step function completes (may be a long
↳ time)
-v, --verbose       Prints out the result of the step_function_trigger run

```

1.2 File Handling

See the [smart_open API documentation here](#) Also see [Working with NetCDF4 Files](#)

The libera-utils smart_open function has the capability to read and write files to/from a local directory or S3 bucket transparently. It supports a [context manager pattern](#) and the usual modes for reading/writing/binary provided by most Python filelike objects:

```

from libera_utils.io.smart_open import smart_open
from libera_utils.io.filenaming import LiberaDataProductFilename

f = LiberaDataProductFilename("s3://some-bucket/LIBERA_L1B_RAD_V1-2-3_20250102T120000_
↳20250103T120000_R25005112233.nc")
with smart_open(f.path, "r") as filehandler:
    # Work with file contents
    pass

```

1.3 File Naming

The Libera Utils Filename classes allow reliable file naming, checking, and path management to support conformity with the Libera filenaming conventions. Each type of filename contains regex that validates every definition or update of the internally tracked filename string. These classes transparently support both S3 paths and local filepaths, including dynamic switching between the two, to simplify the transition between local development environments and AWS.

Full specifics including all available file naming classes are available *in the filenaming API documentation here*

Below is an example test using a LiberaDataProductFilename instance to manage a filename string, including switching between S3 and local paths to show the flexibility of the classes.

```

from pathlib import Path
from cloudpathlib import S3Path
from libera_utils.io import filenaming

p = filenaming.LiberaDataProductFilename(
    'LIBERA_L2_CLOUD-FRACTION_V1-2-3_20270102T112233_20270102T122233_R27002112233.nc')
# Add an S3 prefix
p.path = S3Path('s3://bucket') / p.path
assert isinstance(p.path, S3Path)
# Change prefix to local
p.path = Path('/tmp/path') / p.path.name
assert isinstance(p.path, Path)
# Remove basepath altogether
p.path = p.path.name
assert isinstance(p.path, Path)
# Check that providing a bad value for a basepath doesn't pollute the instance's valid_
↳path
try:
    p.path = '/bad/prefix' + p.path.name # The missing / will make this fail regex_
↳validation
    raise Exception('The previous line should have raised a ValueError')
except ValueError as e:
    assert "failed validation against regex pattern" in str(e)
assert p.path.name == 'LIBERA_L2_CLOUD-FRACTION_V1-2-3_20270102T112233_20270102T122233_
↳R27002112233.nc'

```

1.4 Working with NetCDF4 Files

NetCDF4 is a complete redesign of the NetCDF file format based on HDF5 data structures. i.e. all NetCDF4 files are HDF5 files with some additional requirements and limitation of functionality. Note that not all HDF5 files are NetCDF4 files. For more information on NetCDF5 and the underlying HDF5 structures, see the documentation [here](#).

There are several python packages (and libraries in other languages) that support reading and writing NetCDF4 files. The SDC is using the Xarray python library.

The official documentation for Xarray is [here](#). It includes a much more comprehensive user guide with code examples.

Xarray builds on the numpy package, introducing labels for multidimensional arrays in python. These labels come in the form of coordinates, dimensions, and attributes. Xarray is broken into two main data structures: DataArrays and DataSets. DataArrays are contained within DataSets, such that a single DataSet can hold multiple DataArrays. DataSets can then be written to NetCDF4 files.

1.4.1 Reading NetCDF4 Files

To read NetCDF4 files we can use Xarray as well. NetCDF4 files have similar structure to HDF5 files. NetCDF4 DataSets can have DataSets nested within one another. Here is an example of how to access each DataSet/Group.

```
import xarray

with xarray.open_dataset("filename", group='/') as ds:
    print(ds) # print the highest level group
```

1.4.2 Creating and Using DataArrays

See documentation on `DataArray.to_netcdf` [here](#).

DataArrays are arrays that can handle multiple dimensions with named or labeled axes. These DataArray objects add metadata such as dimension names, coordinates, and attributes. DataArrays can be created from numpy arrays, numpy-like arrays, pandas Series, and pandas DataFrames.

```
import xarray as xr
import numpy as np
import pandas as pd

# Create the coordinates
time = pd.date_range(start='2022-01-01', periods=10, freq='D') # 10 daily time steps
lat = np.linspace(-90, 90, 5) # 5 latitude points from -90 to 90
lon = np.linspace(-180, 180, 8) # 8 longitude points from -180 to 180

# Create random data
data = np.random.random((len(time), len(lat), len(lon)))

# Create the DataArray
data_array = xr.DataArray(data,
                          coords=[time, lat, lon],
                          dims=['time', 'lat', 'lon'])

# Display the DataArray
print(data_array)

print(data_array.values) # the data in the object
print(data_array.dims) # access the dimensions
print(data_array.coords) # access the coors attribute
print(data_array.attrs) # access metadata about the DataArray
```

You can create DataArrays across more dimensions as well. The number of variables in `dims` and `coords` should be equal for multiple dimensions. You can also modify the DataArrays values with scalars.

```
data_array.values = data_array.values * 2 # multiply the entire array by 2
```

1.4.3 Writing DataSets as NetCDF4

See documentation on `Dataset.to_netcdf` [here](#).

Creating DataSets is similar to creating DataArrays, provide the data variables themselves, along with the coords, dims and attributes you want to include. The data variables should be a dictionary with each key being the name of the data and each value can be a DataSet, pandas dataframe or numpy array. Coords and Dims should be a dictionary as well.

For this example, we are creating the DataArrays first, then will add them all into one DataSet and write that to a NetCDF4 file. When creating DataSets, Xarray will often infer the dims from the coords and data variables given, if you do not pass any while creating DataSets. When writing to the NetCDF4 files if different variables “lie” on different dimensions, they will smash them together and replace the extra values (when viewed in a file viewer) with zeros. When writing a file using Xarray, using the engine “h5netcdf” will write the file faster.

```
import random
import pandas
import numpy
import xarray as xr

data_length = random.randint(1000,2000) # creating random vector length to simulate data

# creating multiple data fields to simulate fake data
times = pandas.date_range("2014-09-06", periods=data_length)
short_wave = numpy.random.rand(data_length)
long_wave = numpy.random.rand(data_length)
total_radiance = numpy.random.rand(data_length)
split_radiance = numpy.random.rand(data_length)

# creating the DataSets from the created fields
short_wave = xr.DataArray(short_wave, coords=[times], dims=['times'])
long_wave = xr.DataArray(long_wave, coords=[times], dims=['times'])
total_rad = xr.DataArray(total_radiance, coords=[times], dims=['times'])
split_rad = xr.DataArray(split_radiance, coords=[times], dims=['times'])

# create the DataSet
ds = xr.Dataset({
    'short_wave': short_wave,
    'long_wave': long_wave,
    'total_radiance': total_rad,
    'split_rad': split_rad
},
    coords={'times': times},
)

# Write some metadata
ds.attrs["ALGORITHM_VERSION"] = "3.14.159"

# write to a NetCDF4 file
ds.to_netcdf('filename', group="/", mode='a', engine='h5netcdf')
```

You can specify group structure with group keyword, similar to a filesystem path (/groups/are/paths). When writing multiple DataSets to a file or if you need to append them, use keyword “mode” with value “a” to append.

1.5 Making and Using Manifest Files

All science algorithms that run on the Libera Science Data Center system need capabilities for dealing with Manifest Files. Specifics on the usage of manifest files can be found in the [Manifest API documentation here](#)

The `Manifest` class is designed to handle reading, writing, and interacting with manifest files during processing. It performs such tasks as validating manifest file structure and naming conventions as well as storing the manifest contents as easily accessible python objects and providing helper methods for common tasks related to manifest file handling.

```
from libera_utils.io.manifest import Manifest

# Manifest filenames are passed into your Docker image CLI as its only argument
input_manifest = Manifest.from_file("s3://some-dropbox/LIBERA_INPUT_MANIFEST_
↳20270102T122233.json")
# Read from manifest file to do processing

# Create an output manifest named according to the input manifest (timestamp matches for
↳traceability)
output_manifest = Manifest.output_manifest_from_input_manifest(input_manifest)

# Add files. This will raise a credentials error because it tries to checksum the file
↳but can't access S3
# without credentials provided (your Docker images will have proper credentials
↳attached).
output_manifest.add_files(
    "s3://some-dropbox/LIBERA_L2_CLOUD-FRACTION_V1-2-3_20270102T122233_20270102T122233_
↳R27002112233.nc"
)

# Automatically generates a proper output manifest filename and writes it to the path
↳specified,
# usually this path is retrieved from the environment, like `os.environ["PROCESSING_
↳DROPTBOX"]`.
output_manifest.write("s3://some-dropbox/")
```

1.6 Logging

High quality logging is an important part of operational processing and the Libera SDC Team has made logging setup as painless as possible, while also offering a high degree of configuration for processing algorithms. See below for a general discussion of logging principles followed by some example use cases.

See the [libera_utils.logutil API documentation here](#)

1.6.1 Logging vs. print

Printing is a valid way to log from your code. However, it is limited in a few major ways:

1. You get no logging information from library code you have pulled in as dependencies.
2. There is no easy way to automate adding context to print statements such as current function, line, module, etc.
3. Formatting is only a convention with print calls, which makes log analysis and monitoring difficult.
4. There is no easy way to control the verbosity of your print statements.
5. You can only send messages to the console (stdout/stderr).

When using the Libera Utils logging module, you get:

1. Fine-grained information from all the libraries you are using.
2. Configurable standard context added to logs such as time, severity level, module, line number, function name, etc.
3. Consistent formatting to make logs easily searchable.
4. Ability to easily turn logging on/off from one place in the code.
5. Send log messages to multiple configurable destinations (console, file, etc).

See examples of these use cases in the code examples throughout this page.

1.6.2 Logging Levels in Python

- DEBUG - Detailed information, typically of interest only when diagnosing problems.
- INFO - Confirmation that things are working as expected.
- WARNING - An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
- ERROR - Due to a more serious problem, the software has not been able to perform some function.
- CRITICAL - A serious error, indicating that the program itself may be unable to continue running.

1.6.3 Setting Up Logging in Applications

The `libera_utils.logutil` module provides utilities for configuring logging easily for file-based, stream (stdout/stderr), and custom AWS CloudWatch logging.

Simplest Logging Setup

This example allows all messages through to the console at the specified level. This does not filter out DEBUG logs from verbose libraries.

```
"""Simplest logging setup"""
import logging
from datetime import datetime, timezone
import boto3
from botocore.exceptions import NoCredentialsError
from libera_utils.logutil import configure_task_logging

logger = logging.getLogger(__name__)

if __name__ == "__main__":
    task_id = f'processing-task-{{datetime.now(tz=timezone.utc).strftime("%Y-%m-%dT%H:%M:
↵%S")}}'
    configure_task_logging(task_id, console_log_level="DEBUG")

    logger.debug("test debug message")

    try:
        # The following will demonstrate why we might want to filter out debug messages
        buckets = boto3.client('s3').list_buckets()
        logger.info(buckets)
```

(continues on next page)

(continued from previous page)

```
except NoCredentialsError:
    logger.error("No credentials found")
```

produces

```
2024-04-23 07:54:37,523 INFO      [libera_utils.logutil:logutil.py:257 in configure_task_
↳ logging()]: Console logging configured at level DEBUG.
2024-04-23 07:54:37,523 DEBUG     [__main__:scratch_11.py:15 in <module>()]: test debug_
↳ message
2024-04-23 07:54:37,523 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from creating-client-class.iot-data to creating-client-class.iot-
↳ data-plane
2024-04-23 07:54:37,524 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from before-call.apigateway to before-call.api-gateway
2024-04-23 07:54:37,524 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from request-created.machinelearning.Predict to request-created.
↳ machine-learning.Predict
2024-04-23 07:54:37,524 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from before-parameter-build.autoscaling.CreateLaunchConfiguration_
↳ to before-parameter-build.auto-scaling.CreateLaunchConfiguration
2024-04-23 07:54:37,525 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from before-parameter-build.route53 to before-parameter-build.
↳ route-53
2024-04-23 07:54:37,525 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from request-created.cloudsearchdomain.Search to request-created.
↳ cloudsearch-domain.Search
2024-04-23 07:54:37,525 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from docs.*.autoscaling.CreateLaunchConfiguration.complete-section_
↳ to docs.*.auto-scaling.CreateLaunchConfiguration.complete-section
2024-04-23 07:54:37,526 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from before-parameter-build.logs.CreateExportTask to before-
↳ parameter-build.cloudwatch-logs.CreateExportTask
2024-04-23 07:54:37,526 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from docs.*.logs.CreateExportTask.complete-section to docs.*.
↳ cloudwatch-logs.CreateExportTask.complete-section
2024-04-23 07:54:37,526 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from before-parameter-build.cloudsearchdomain.Search to before-
↳ parameter-build.cloudsearch-domain.Search
2024-04-23 07:54:37,526 DEBUG     [botocore.hooks:hooks.py:482 in _alias_event_name()]:_
↳ Changing event name from docs.*.cloudsearchdomain.Search.complete-section to docs.*.
↳ cloudsearch-domain.Search.complete-section
...and much much more
```

Notice the huge volume of DEBUG messages originating from loggers in the botocore package.

Filtered Logging Setup

Now we want to alter the code above to take advantage of the ability to reduce the amount of DEBUG spam from libraries that we are not interested in. Note the use of `__main__` in the `limit_debug_loggers` tuple. This allows debug messages that originate at the level of a runnable python script that is wrapped in the standard `if __name__=="__main__"` guard.

```

"""Simplest logging setup"""
import logging
from datetime import datetime, timezone
import boto3
from botocore.exceptions import NoCredentialsError
from libera_utils.logutil import configure_task_logging

logger = logging.getLogger(__name__)

if __name__ == "__main__":
    task_id = f'processing-task-{{datetime.now(tz=timezone.utc).strftime("%Y-%m-%dT%H:%M:%S")}}'
    configure_task_logging(task_id, limit_debug_loggers=("__main__", "libera_utils"),
    console_log_level="DEBUG")

    logger.debug("test debug message")

    try:
        # The following will demonstrate why we might want to filter out debug messages
        buckets = boto3.client('s3').list_buckets()
        logger.info(buckets)
    except NoCredentialsError:
        logger.error("No credentials found")

```

produces

```

2024-04-23 07:52:56,009 INFO      [libera_utils.logutil:logutil.py:257 in configure_task_
↳ logging()]: Console logging configured at level DEBUG.
2024-04-23 07:52:56,009 DEBUG    [__main__:scratch_11.py:15 in <module>()]: test debug_
↳ message
2024-04-23 07:52:56,186 ERROR    [__main__:scratch_11.py:22 in <module>()]: No_
↳ credentials found

```

Contrived Runnable Example

This illustrates how the `limit_debug_loggers` kwarg works for preventing other libraries from logging at DEBUG level while still allowing DEBUG messages from libraries you care about.

```

"""Logging setup contrived example"""
from datetime import datetime, timezone
import logging
from libera_utils.logutil import configure_task_logging

task_id = f'processing-task-{{datetime.now(tz=timezone.utc).strftime("%Y-%m-%dT%H:%M:%S")}}'
↳
configure_task_logging(task_id, limit_debug_loggers=("my_package",), console_log_
↳ level=logging.DEBUG)
# my_log is a logger from inside your library (name prefixed with your library name).
my_log = logging.getLogger('my_package.my_application')
# The following is an example. External libraries will create their own loggers_
↳ internally.
external_library_log = logging.getLogger('some_spammy_library')

```

(continues on next page)

(continued from previous page)

```
my_log.debug('subtle but important message gets passed through')
external_library_log.debug('this external library debug spam gets filtered out')
external_library_log.info('and external library info messages still get through')
```

Configuring Stream Logging

Stream logging needs only a level and it defaults to INFO. Stream logging cannot be disabled but is easily ignored.

Note: You can change the default formatter for stream logging from plaintext to json by passing `console_log_json=True` to `configure_task_logging`. This is convenient for logging in AWS services that push their stdout logs to CloudWatch.

```
"""Example of setting up console logging (JSON formatted)"""
import logging
from libera_utils.logutil import configure_task_logging

configure_task_logging("test-task-id-1", console_log_json=True, console_log_
↳ level=logging.DEBUG)
```

Configuring Filesystem Logging

Filesystem logging needs only a log directory (it always logs at DEBUG level). If you don't pass `log_dir`, no file-based logging will occur. The log directory must exist and will not be dynamically created.

Note: Logging to a directory is really only useful for local testing. Any logs written to a directory in a docker container will evaporate upon completion of the docker container process.

```
"""Example of setting up file-based logging"""
from pathlib import Path
from libera_utils.logutil import configure_task_logging

configure_task_logging("test-task-id-1", log_dir=Path("/tmp"))
```

1.6.4 Logging Exceptions

The Python logging module provides logging calls associated with each level. In addition, it provides a logging call for logging exceptions that includes the current stack trace for debugging.

```
"""Example logging calls"""
import logging

logger = logging.getLogger(__name__)

if __name__ == "__main__":
    logging.basicConfig(level=logging.DEBUG)
    logger.debug("test debug")
    logger.info("test info")
    logger.warning("test warning")
    logger.error("test error")
    logger.critical("test critical")

    try:
        raise ValueError("example exception to log")
```

(continues on next page)

(continued from previous page)

```

except ValueError:
    logger.exception("encountered exception") # This logs the message followed by
↳ the exception traceback
    # raise # Optional re-raise of exception after logging it

```

1.6.5 Concept: Module Level Logging

(AKA library logging)

Module level logging is the practice of defining a logger at the top of each module (.py file) and using that logger object for the entire module. Modules in libera_utils should all have module level loggers when appropriate. e.g.

```

"""Example of module level logger instantiation"""
import logging

logger = logging.getLogger(__name__)

# Module code

```

One advantage of module level logging is that it provides a logger, named for the module from which it is logging but doesn't configure the logger at the module level. Since much of this code is intended to be reused by others, we avoid configuring loggers in reusable code. If a logger is needed in a context, it should be configured at the "application level". That is, the top level application (e.g. CLI tool) that is running a process should take care of logging configuration and assume that each module has generic module level loggers configured for the internal code to use.

Another advantage of module level logging is that our loggers come out with automatically structured names like libera_utils.db.database and they all start with libera_utils. This allows us to treat those loggers differently than those named, for example, some_spammy_library.emit_spam. In our logging setup, we allow users to turn off debug messages for all loggers that aren't named with specific prefixes. This allows us to pass up debug messages from our code but ignore debug messages from dependency code (AWS boto APIs in particular spam a LOT of debug messages).

1.6.6 Fully Customized Logging

If you want complete control over your logging configuration, you can use our provided `configure_static_logging` function, which reads a YAML configuration file that represents a Python logging configuration. This is a completely static configuration and should be supplied as part of your processing algorithm application code.

```

"""Example of configuring logging with static config file"""
import logging
from pathlib import Path
from libera_utils.logutil import configure_static_logging

config = Path("/path/to/config_file.yml")
configure_static_logging(config)

logger = logging.getLogger()
logger.info("handling depends on your supplied configuration")

```

An example of a logging config file:

```

# Example parameterized logging configuration
version: 1

```

(continues on next page)

(continued from previous page)

```
disable_existing_loggers: False
formatters:
  json:
    format: '{"time": "%(asctime)s",
              "level": "%(levelname)s",
              "module": "%(filename)s",
              "function": "%(funcName)s",
              "line": %(lineno)d,
              "message": "%(message)s"}'

  plaintext:
    format: "%(asctime)s %(levelname)-9.9s [% (filename)s:%(lineno)s in
↳ %(funcName)s()]: %(message)s"
handlers:
  console:
    class: logging.StreamHandler
    formatter: plaintext
    level: INFO
    stream: ext://sys.stdout
  logfile:
    class: logging.handlers.RotatingFileHandler
    formatter: plaintext
    level: DEBUG
    filename: /tmp/libera_utils_test_log.log
    maxBytes: 1000000
    backupCount: 3
root:
  level: INFO
  propagate: True
  handlers: [console, logfile]
loggers:
  libera_utils:
    qualname: libera_utils
    level: DEBUG
    handlers: []
```


DEVELOPER DOCS

This section is for developers working on the Libera Utils package but also includes reference documentation for the tools used for development and how to use them (e.g. git and Docker).

2.1 Setting Up a Development Environment

2.1.1 Managing Multiple Base Python Versions

In order to develop with multiple different versions of Python and create virtual environments associated with different versions of Python, you will need multiple base Python interpreters. There are several ways to manage this including Conda, PyEnv, and building Python from source. We recommended using Conda and outline the steps below for using Conda to manage multiple base Python installations.

1. Install miniconda according to the [official documentation](#). If you already have miniconda or anaconda installed, you can skip this step.
2. Optionally run `conda config --set auto_activate_base false` to add a configuration to your `.condarc` file to disable auto-activation of the base conda environment on shell startup.
3. Create a conda environment with your preferred version of python: `conda create -n conda-python3.11 python=3.11`
 - Note: Name this environment with a convention that makes sense to you for a base interpreter. *Do not delete this conda environment!* Deleting it will break all subsequent virtual environments based on it.
4. The Python interpreter provided by your new conda environment is a full base interpreter and you can use it to create virtual environments. You can find the full path to the base interpreter by running something similar to the following (run `conda env list` to see why this works):

```
PATH_TO_PYTHON=$(conda env list | grep "conda-python3.11" | awk '{print $2}')/bin/  
↪python  
$PATH_TO_PYTHON -m venv path/to/new/venv
```

2.1.2 Installing Poetry

Poetry is a command line tool that helps manage a python development environment, including package management, virtual environment management, and package building.

Poetry official installation instructions can be found here: <https://python-poetry.org/docs/#installation>

To ensure that Poetry is always available and in the PATH it is recommended to install Poetry with your pre-installed system python interpreter rather than as a package in a conda environment or in a virtual environment. The specific version of python with which you install Poetry is inconsequential (as long as it is currently supported by Poetry). If your system python is not supported by Poetry, you can install Poetry in your conda base environment. Just remember

that Poetry will only be available when that environment is activated. Things can get a bit confusing when you have a conda environment active and a derived virtual environment activated on top of it.

Once poetry is installed, check that it works by running `poetry --version`. You should get something like

```
Poetry version 1.8.3
```

Installing Poetry with System Python

Ensure that all your virtual environments and conda environments are deactivated and that `which python3` refers to your system python interpreter (usually `/usr/bin/python3`).

```
curl -sSL https://install.python-poetry.org | python3 -
```

2.1.3 Configuring Poetry

We recommend creating your own virtual environments in locations of your choosing (common to create them in the project directory, as `venv` or `.venv`).

2.1.4 Setting Up Development Virtual Environment(s)

Poetry will dynamically detect the presence of an activated virtual environment and use that if present. If none is present, Poetry will automatically create one for you in your user cache location (e.g. on mac in `~/Library/Caches/pypoetry/virtualenvs`). This can be confusing so we recommend creating your venv and letting poetry use it when activated.

1. Deactivate all Conda environments and virtual environments
2. Activate the conda environment you wish to use for your base python interpreter (e.g. `conda activate conda-python3.11`)
3. Create a virtual environment anywhere you wish. `python -m venv path/to/venv`.
4. Activate newly created venv: `source path/to/venv/bin/activate`
5. [Recommended]: Configure your IDE to recognize the correct poetry-managed virtual environment for the version you wish to develop with.
6. Run `poetry env info` and verify that Poetry is recognizing your virtual environment properly:

```
Virtualenv
Python:      3.9.9
Implementation: CPython
Path:        /Users/myuser/path/to/libera_utils/venv
Valid:       True
```

Changing Python Versions

It is common to recreate your virtual environment on a regular basis in order to use different python versions. You can do this by making sure that `python` points to the base interpreter you wish to use (e.g. 3.12) and going through the steps above to create a new venv (you can name it differently) and activating it for poetry to use.

Installing Dependencies

1. Run `poetry lock && poetry install` in the same directory as the `pyproject.toml` file. You should see poetry solving the dependency tree and then installing dependencies. This also installs dev group dependencies, as specified in `pyproject.toml`. Lastly you should see it installing the local package.

2. To install optional “extra” dependencies, run `poetry install -E extra_name1 -E extra_name2`. These extra dependencies are specified in `pyproject.toml` under `[tool.poetry.extras]`. Note that any subsequent `poetry install` command without `--extras` will implicitly uninstall any previously installed extras.
3. To install dependency “groups” (think labels), which may or may not be optional, use the `--with` and `--without` flags for Poetry. e.g. `poetry install --with docgen` will install the dependencies for the optional group “docgen”.
4. Verify that the `libera_utils` package was installed correctly by running `libera-utils --version`. This runs the `libera-utils` command line utility that is included in the package. This can also be run with `poetry run libera-utils --version`.
5. Next, *go run the tests*.

2.2 Configuration

Configurations are stored in a JSON file `config.json` but we allow overriding those values with environment variables. If, at any point in the code, the config is queried (`config.get(key)`) for a key that isn’t present in the top level of the JSON file, a warning is issued to add a default value to the JSON file. This helps ensure that we can track all possible configuration values in the JSON config rather than having to bookkeep them elsewhere.

2.3 Testing

Testing is run with `pytest`. To run all tests, make sure the dev dependencies are installed and run:

```
# Run all unit and integration tests
pytest tests
```

Pytest configuration is stored in `pyproject.toml` under `[tools.pytest.ini_options]`.

Tests are stored in the `tests` directory and are divided into `integration` and `unit` tests. Unit tests check for expected behavior of small pieces of the package (e.g. a single function). Integration tests check for “larger” behavior across wider swaths of the package, testing that components function together cohesively. Integration test modules contain the `pytest` mark `pytestmark = pytest.mark.integration`. This allows you to selectively exclude running integration tests with

```
pytest -m "not integration" tests
```

Unit tests heavily utilize `pytest` parameterization in order to run a single test against many sets of inputs and outputs without duplicating code (see `[parameterize]` (<https://docs.pytest.org/en/stable/example/parametrize.html#set-marks-or-test-id-for-individual-parametrized-test>)).

We also heavily utilize custom `pytest` `[fixtures]` (<https://docs.pytest.org/en/stable/how-to/fixtures.html#requesting-fixtures>), defined in `tests/plugins`. These plugins are made available to `pytest` in `conftest.py`, the main test configuration file for `pytest`.

In order to better ensure test independence, we use `pytest-randomly` to randomize the order of tests. The random seed used to set the order is printed at the beginning of a test run. You can re-run the tests with a specific random seed by running `pytest --randomly-seed=<seed>`.

2.3.1 Generating Coverage and Test Reports

With coverage for generating reports on code coverage:

```
# Create coverage data (stored in .coverage)
pytest --cov=libera_utils --junit-xml=junit.xml
# Generate interactive HTML coverage report
pytest --cov-report=html:coverage_report --cov=libera_utils
# Generate Corbertura-compatible XML report
pytest --cov-report=xml:coverage.xml --cov=libera_utils
```

2.3.2 Testing in Docker

To run the unit tests in docker, run

```
docker-compose up [--build] --exit-code-from=tests tests --attach=tests
```

This runs the tests container service defined in the docker-compose.yml file. The --build option forces docker to rebuild the testing container image before running (e.g. if things have changed).

Copying Test Report Artifacts from Docker

When we run tests in Docker on Jenkins, we often want to copy and save Corbertura and JUnit test reports. Jenkins has facility for doing this easily with

```
always {
    junit '**/*junit.xml'
    cobertura coberturaReportFile: '**/*coverage.xml'
}
```

The challenge when running in Docker is to make these test artifacts available to Jenkins. By default these files exist only inside the Docker container so we must copy them out. Do this with

```
docker-compose --exit-code-from tests up tests
docker-compose cp tests:/path/to/report.xml .
```

2.3.3 Static Analysis

NASA requirements document NPR7150.2C requires that we perform static analysis of our codebase to check for common vulnerabilities and statically detectable code weaknesses and vulnerabilities (CWEs and CVEs).

Pylint for Code Style

[Pylint](#) is a powerful and highly configurable static analysis tool that analyzes Python code for coding standards (e.g. PEP8), code smells, type errors, and violations of common best practices. Together, these violations are common code weaknesses and we strive to eliminate them.

To run pylint locally, run

```
pylint libera_sdp
```

from the repo root. This will lint the libera_sdp directory and automatically pick up our pylintrc configuration file.

To run pylint inside Docker, run

```
docker-compose up [--build] linting
```

This starts up the linting service described in docker-compose.yml, which runs pylint inside a testing docker container.

Bandit for Automated Security Testing (AST)

Bandit is a security vulnerability tester that analyzes Python code for common weaknesses, including the “official” CWE set provided by the [MITRE Corp.](#).

To run bandit locally, run

```
bandit -r libera_sdp
```

from the repo root. This will recursively (-r) analyze the libera_sdp directory and report results.

To run bandit inside Docker, run

```
docker-compose up [--build] ast
```

This starts up the ast service described in `docker-compose.yml`, which runs bandit inside a testing docker container.

Static Analysis pre-commit Git Hook

The static analysis checking can be annoying when you only catch it after you have committed, rebased, and put up a PR. To alleviate that suffering, here is a pre-commit git hook that will execute before every commit and refuse to continue until you have fixed your static analysis problems.

```
#!/bin/sh

# .git/hooks/pre-commit

# Redirect output to stderr.
exec 1>&2

# Run pylint.
echo "Running pylint..."
files_to_check=$(git diff --name-only --cached --diff-filter=AM libera_utils | grep '.py$'
→)
if [ -n "$files_to_check" ]; then
    pylint $files_to_check
else
    echo "No .py file changes to lint"
fi

# Capture the output of pylint.
pylint_exit_code=$?

# If pylint returns a non-zero exit code, cancel the commit.
if [ $pylint_exit_code -ne 0 ]; then
    echo "Pylint check failed, aborting commit..."
    exit 1
fi

# Otherwise, proceed with the commit.
echo "Pylint check passed, proceeding with commit..."

echo "Running bandit..."
bandit -r --quiet libera_utils
bandit_exit_code=$?

# If Bandit returned a non-zero exit code, cancel the commit.
```

(continues on next page)

(continued from previous page)

```
if [ $bandit_exit_code -ne 0 ]; then
    echo "Bandit check failed, aborting commit..."
    exit 1
fi

# Otherwise, proceed
echo "Bandit AST passed, proceeding with commit..."

exit 0
```

2.4 Build and Release

2.4.1 Local package building for CLI testing

To build the package locally for testing especially for the cli interface, use the following steps:

1. Ensure that you have activated a virtual environment where you would like libera-utils to be installed.
2. run `python -m pip install .` from the root of the repository.
3. You should now be able to run the `libera-utils --version` command from the command line as see that the version number matches the one in `pyproject.toml`.

2.4.2 Release Process

[Atlassian Git Workflow Reference](#):

1. Create a release candidate branch named according to the version to be released. This branch is used to polish the release while work continues on dev (towards the next release). The naming convention is `release/X.Y.Z`
2. Bump the version of the package to the version you are about to release, either manually by editing `pyproject.toml` or by running `poetry version X.Y.Z` or bumping according to a valid bump rule like `poetry version patch` (see poetry docs).
3. Open a PR to merge the release branch into main. This informs the rest of the team how the release process is progressing as you polish the release branch.
4. When you are satisfied that the release branch is ready, merge the PR into main. This should be a purely “fast-forward” merge. Do not delete the release branch when merging as you will need it later.
5. Check out the main branch, pull the merged changes, and tag the newly created merge commit with the desired version `X.Y.Z` and push the tag upstream.

```
git tag -a X.Y.Z -m "version release X.Y.Z"
git push origin X.Y.Z
```

6. Checkout the tag you just created (ensures the correct version is recorded in the build artifacts) and build the package (see below). Check that the version of the built artifacts is as you expect (should match the version git tag).
7. Optionally distribute the artifacts to PyPI/Nexus if desired (see below).
8. Open a PR to merge `release/X.Y.Z` back into dev so that any changes made during the release process are also captured in dev. This should be a purely “fast-forward” merge.

2.4.3 Building and Distribution to Public PyPI

1. Ensure that poetry is installed by running `poetry --version`.
2. Checkout the tag of the version you are releasing.
3. To build the distribution archives, run `poetry build`.
4. To upload the wheel to PyPI, first set your environment variables with the API token for the correct PyPI account:

```
export PYPI_USERNAME=__token__
export PYPI_TOKEN=<Your API Token>
```

Then run `poetry publish --username $PYPI_USERNAME --password $PYPI_TOKEN`. You will need the account information for the `liberasdc` PyPI account.

2.5 Documentation Generation for Libera Utils

2.5.1 Creating Documentation with Sphinx

Sphinx documentation is a python library for generating documentation from docstring comments throughout a codebase in combination with other rst or md pages separate from the generated documentation. We use it to generate automatic API documentation for the codebase as well as building developer documentation and user documentation from markdown and restructured text documents (such as this one).

You will need `make` installed on your machine in order to build sphinx docs.

Compatible Comments and Docstrings

Docstrings in this project are expected in the [Numpy docstring format](#).

Writing Custom Documentation Pages

If you wish to add pages that are not part of the generation from the code such as reference pages, use the following steps.

1. Create your files and folders and add them to the project folder `doc/source/`
2. Edit the `doc/source/index.rst` file to add the file you have just created to the toctree (table of contents tree)
3. If you added developer documentation, instead of editing the `index.rst` edit the `doc/source/developer-docs.rst` document or create a new folder and associated rst document containing a toctree that includes your new file.

The folder structure is an organizational convention but the actual page structure comes from the toctree structure in the rst files (note that these documents could also be markdown but rst has better support for TOC listings).

Note: This project is configured to read and interpret both markdown (`.md`) and reStructured Text (`.rst`) documents. The following are two basic guides to writing proper comments that will create well formatted outputs.

- [reStructured Text](#)
- [Markdown](#)

If you have need to translate between rst and markdown, use [pandoc](#).

2.5.2 Building HTML Documentation Locally

This should all be done while in the virtual environment that is configured for this project. See the *dev environment setup documentation* for poetry instructions.

1. Run `poetry update --with docgen`
 - This ensures that the poetry install is up-to-date, including the “docgen” dependency group.
2. Run `poetry install --with docgen`
 - This installs all project dependencies, including the `pyproject.toml` docgen group (e.g. Sphinx, etc.)
 - This also ensures that the project itself is installed with its most recent version (as defined in `pyproject.toml`)
3. Navigate to the Sphinx document source folder `cd ./doc`
4. Build the html files in the build folder using the make command `make html`
5. Open the generated `build/index.html` file to go to the documentation homepage.

Oneliner for an IDE configuration:

```
# Must be run from the `doc` working directory  
make html && open build/html/index.html
```

2.5.3 Automatic Documentation Publishing with ReadTheDocs

We publish our documentation using ReadTheDocs. Our configuration file for readthedocs is located in `.readthedocs.yaml`. The Libera SDC team has a ReadTheDocs account containing an organization and each team member has a login to view build status and private/hidden doc builds. Only versions set to Public are visible without a login.

ReadTheDocs responds to webhooks sent by both Bitbucket and Jenkins.

Webhook from Bitbucket

The Bitbucket webhook for ReadTheDocs goes directly to ReadTheDocs to trigger a build.

Webhook from Jenkins

ReadTheDocs expects specific payload structure to its webhook API. When Bitbucket sends a payload, it triggers a build for the default branch (`main`) because the structure isn’t supported by ReadTheDocs. In order to get specific branch builds, we send a webhook for PR updates to Jenkins, which reformats the payload into the correct structure for ReadTheDocs, which triggers a build for the specific PR branch being modified.

Publishing Official Release Docs

ReadTheDocs automatically builds and publishes new versions for tags, but it requires a trigger to re-fetch any new tags. This means a build must be triggered after the release process tag is applied for a new official docs version to be built.

To do this, log in to ReadTheDocs and click “Add a New Version”, then at the bottom, click “Resync Versions”. This will trigger ReadTheDocs to fetch the new tag from Bitbucket and build it as an official version.

2.6 Git Usage

2.6.1 Basic Workflow

We use [trunk-based development](#) as our basic git workflow. The following is a general order of events:

1. Create a branch from `main` for a feature. Name it based on your feature or ticket. e.g. `feature/LIBSDC-XXX-add-something-cool`
2. Add commits to your feature branch.
3. (Optional) Put up a Draft PR to signify a work in progress while still allowing team members visibility. This also allows automated tests to run based on changes to the PR.
4. Ensure your branch is rebased and commits are squashed onto the latest commits in `main`. This may involve separate squashing and rebasing operations to minimize conflicts (i.e. squash first, then rebase a single commit).
5. Put up a PR to merge into `main`
6. Wait for review.
7. Merge using the “fast-forward only” strategy in Bitbucket. If properly squashed, should only add 1 commit.

If any of this doesn't make sense, don't just run commands! Ask someone! You can really hose your local repo state and even lose your work if you don't know what you're doing.

2.6.2 Reasons for Rebasing

Rebasing allows us to keep our git history completely linear and avoids creating unnecessary merge commits. Merge commits are dangerous because git doesn't actually know what the code is doing and can “resolve” merge conflicts in ways that result in broken code. No commit in `main` should ever contain broken code if we can help it.

2.6.3 Reasons for Squashing

Squashing reduces the total number of commits in the repository to ~1 per pull request. Since each commit contains a full snapshot of the codebase, this drastically reduces the amount of storage necessary to host our git repo and makes life slightly easier for our beloved Web Team.

2.6.4 Release Branches

Release branches are created in preparation for a tagged release from `main`. They are really just special feature branches that allow us to ensure the repo metadata (version, dependencies, changelog, docs, etc.) are in good shape for a formal release. When we merge a release branch, we delete it and perform the final release from the latest commit in `main` by tagging that commit with the version and deploying build artifacts to PyPI. See the [build and release docs](#) for more details on our release process.

2.7 Git LFS (Large File Storage) Usage

We use Git LFS to store large files in a way that doesn't blow up the size of our repo on the git server. Usually each commit contains a snapshot of the repository at that point in time. If you store a 100MB file, your entire repo size will be 100MB * number of commits, which can easily balloon to a big number. Incidentally, this is also why we squash PRs to reduce the number of commits in the main branch over time.

[Git LFS Documentation](#)

[Tutorial on Using Git LFS](#)

[Atlassian Docs on Git LFS](#)

2.7.1 Set Up

Install Git LFS according to the Git LFS official documentation (linked above).

Run the following to initialize Git LFS for your user:

```
git lfs install
```

2.7.2 A Note on Git LFS Authentication and Git GUIs

Git LFS authenticates separately from Git. Historically, Git LFS supported only HTTP auth, which was a huge pain because best practice is to use SSH to authenticate with Git servers (note that GitHub has actually deprecated HTTP authentication). This situation meant that even if you were using SSH for git auth, you still had to provide and store HTTP credentials for Git LFS.

However, as of Git LFS v3.0, [SSH authentication is supported!](#)

Git GUIs will require you to configure authentication for both Git and Git LFS. It is recommended to use SSH for both but the configuration processes for GUI programs (and IDEs) are different so we're not addressing it here. GLHF!

2.7.3 Tracking New Files in LFS

LFS keeps track of which files are stored in LFS via the `.gitattributes` file.

To track specific files:

```
git lfs track "<pattern>" # The double quotes matter to prevent shell expansion
# e.g. track all files in every directory named test_data
git lfs track "**/test_data/*"
# This ^ grabs all the filepaths that match and writes them directly into .gitattributes
```

To track files based on a pattern in `.gitattributes`:

```
# .gitattributes
# Track all netCDF files that live anywhere inside a test_data directory
**/test_data/**/*nc
```

[Documentation on Pattern Syntax](#)

2.7.4 Tracking Existing Files in LFS

If you have already committed a file and you wish to move that file to Git LFS, you can:

Add it specifically using `git lfs track` e.g.

```
git lfs track my_large_file.big
```

This method appears to have some magic sugar behind it that automatically removes and re-adds the file to git tracking.

Alternatively, you can add the appropriate pattern to `.gitattributes` and then remove and re-add the file to git history so Git LFS picks it up.

```
echo "**/*.big" >> .gitattributes # Add pattern to .gitattributes
git rm --cached my_tracked_file.big # Remove file from git tracking
git add my_tracked_file.big # Git LFS should pick it up at this point
```

This method is preferable if you want your files generally tracked by pattern rather than individually.

NOTE: As a bit of a trick, you can combine the above strategies by running `git lfs track` on the pattern you wish to store in Git LFS. Then replace the individual records added to `.gitattributes` with the appropriate generic pattern that matches the specific files to be tracked.

2.7.5 Useful Git LFS Commands

List all files in the current git ref (branch, commit, tag, etc.) currently managed by Git LFS:

```
git lfs ls-files
```

Update the files in your `.git/lfs` directory with the version for your current ref:

```
git lfs fetch
```

Convert local pointer files to full files (from `.git/lfs` directory):

```
git lfs checkout
```

Combine fetch and pull into one step:

```
git lfs pull
```

2.8 Usage of SPICE

Fun fact: SPICE stands for “Spacecraft Planet Instrument C-matrix Events,” which were the original primary concerns of those developing the library. The “C” stands for Camera because early instruments using SPICE were cameras.

2.8.1 Static Kernels Generated at Libera SDC

These kernels are part of the package data, are manually edited, and change rarely.

Frame Kernel (FK)

e.g. `libera_fk_v01.tif`

Contains reference frame definitions for JPSS and Libera.

Spacecraft Clock Kernel (SCLK)

e.g. `jpss_sclk_v01.tsc`

Contains specification of the spacecraft clock on JPSS.

Instrument Kernel (IK)

e.g. `libera_cam_ik_v01.ti`, `libera_rad_ik_v01.ti`

Contains geometry specification data of the Libera instruments.

2.8.2 Dynamic Kernels Generated at Libera SDC

These kernels are binary generated kernels. They are ancillary data products and are created as part of pipeline processing.

JPSS Ephemeris Kernel (SPK)

e.g. libera_jpss_20210408t235850_20210409t015849_vM3m14p159_r25365125959.bsp

Contains ephemeris data – coordinates in ITRF93 frame – for the JPSS spacecraft body.

JPSS Attitude Kernel (CK)

e.g. libera_jpss_20210408t235850_20210409t015849_vM3m14p159_r25365125959.bc

Contains attitude data – quaternion rotations from J2000 – for the JPSS spacecraft body.

Azimuth Rotation Mechanism Attitude Kernel (CK)

e.g. libera_azrot_20210408t235850_20210409t015849_vM3m14p159_r25365125959.bc

Contains attitude data for the Libera Azimuth Rotation mechanism.

Elevation Scan Mechanism Attitude Kernel (CK)

e.g. libera_elscan_20210408t235850_20210409t015849_vM3m14p159_r25365125959.bc

Contains attitude data for the Libera Elevation Scan mechanism.

2.8.3 Kernels Retrieved from NAIF

These kernels are generated at NAIF. We download them as needed using the `KernelFileCache` class, which is configured with a NAIF index page URL and a regex string that finds the proper download URL on the index page. The downloaded file is put in a cache so the download only occurs after the cached data is of a specified age. If we want to effectively cache some kernels indefinitely, we can put them in an S3 bucket and retrieve them from there instead of from the NAIF website.

Leapseconds Kernel (LSK)

e.g. naif0012.tls

Contains leap second data used by time conversion routines.

Development Ephemeris Kernel (SPK)

e.g. de440s.bsp

Contains ephemeris data for planetary bodies. The version with “s” appended to the filename covers only more recent time (starts in the 1500s) to reduce filesize.

High Precision Earth Binary Planetary Constants Kernel (PCK)

e.g. earth_000101_211220_210926.bpc

Contains high precision orientation data for Earth in the ECEF ITRF93 reference frame.

ITRF93 is a more precise version of the standard IAU_EARTH reference frame provided in the text PCK below.

This kernel is regenerated several times per week so we should retrieve this from NAIF for every processing run.

ITFR93 Reference Frame Kernel for Earth

e.g. earth_assoc_itrf93.tf

Used to designate ITRF93 as the default body-fixed frame associated with the Earth.

Standard Text Planetary Constants Kernel (PCK)

e.g. `pck00010.tpc`

Contains orientation data and other planetary constants for planetary bodies.

2.9 Libraries Used for Kernel Handling

We use [Curryer](#) for kernel generation and [SpiceyPy](#) for generic kernel handling and calls to CSPICE (Curryer also uses SpiceyPy internally).

This is the API documentation for the entire Libera Utils library.

<i>libera_utils</i>	libera_utils
---------------------	--------------

3.1 libera_utils

libera_utils

Modules

<i>aws</i>	
<i>backports</i>	Module for python code that is included in later python releases but not in earlier ones
<i>cli</i>	Module for the Libera SDC utilities CLI
<i>config</i>	Configuration reader.
<i>db</i>	db module for dyanmodb operations.
<i>geolocation</i>	Module for performing geolocation tasks
<i>io</i>	
<i>kernel_maker</i>	Module containing CLI tool for creating SPICE kernels from packets
<i>logutil</i>	Logging utilities
<i>packets</i>	Module for reading packet data
<i>quality_flags</i>	Quality flag definitions
<i>spice_utils</i>	Modules for SPICE kernel creation, management, and usage
<i>time</i>	Module for dealing with time and time conventions
<i>version</i>	Module for anything related to package versioning

3.1.1 libera_utils.aws

Modules

<i>constants</i>	AWS ECR Repository/Algorithm names
<i>ecr_upload</i>	Module for uploading docker images to the ECR

continues on next page

Table 3 – continued from previous page

<i>processing_step_function_trigger</i>	Module for manually triggering a step function
<i>s3_utilities</i>	Module for S3 cli utilities
<i>utils</i>	Helper functions for AWS access

libera_utils.aws.constants

AWS ECR Repository/Algorithm names

Classes

<i>CkObject</i> (value)	Enum of valid CK objects
<i>DataLevel</i> (value)	Data product level
<i>DataProductIdentifier</i> (value)	Enumeration of data product canonical IDs used in AWS resource naming These IDs refer to the data products (files) themselves, NOT the processing steps (since processing steps may produce multiple products).
<i>LiberaAccountSuffix</i> (value)	Suffixes for the various account types
<i>LiberaApid</i> (value)	APIDs for L0 packets
<i>LiberaDataBucketName</i> (value)	Names of the data archive buckets
<i>ManifestType</i> (value)	Enumerated legal manifest type values
<i>ProcessingStepIdentifier</i> (value)	Enumeration of processing step IDs used in AWS resource naming and processing orchestration
<i>SpkObject</i> (value)	Enum of valid SPK objects

libera_utils.aws.constants.CkObject

class libera_utils.aws.constants.**CkObject**(value)

Bases: `Enum`

Enum of valid CK objects

`__init__`(*args, **kws)

Attributes

<i>data_product_id</i>	DataProductIdentifier for CKs associated with this CK object
<i>processing_step_id</i>	ProcessingStepIdentifier for the processing step that produces CKs for this CK object
JPSS	
AZROT	
ELSCAN	

property *data_product_id*: `DataProductIdentifier`

DataProductIdentifier for CKs associated with this CK object

property *processing_step_id*: `ProcessingStepIdentifier`

ProcessingStepIdentifier for the processing step that produces CKs for this CK object

libera_utils.aws.constants.DataLevel**class** libera_utils.aws.constants.DataLevel(*value*)Bases: `StrEnum`

Data product level

Methods

<i>capitalize()</i>	Return a capitalized version of the string.
<i>casefold()</i>	Return a version of the string suitable for caseless comparisons.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>encode</i> ([, encoding, errors])	Encode the string using the codec registered for encoding.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>expandtabs</i> ([, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>format</i> (*args, **kwargs)	Return a formatted version of S, using substitutions from args and kwargs.
<i>format_map</i> (mapping)	Return a formatted version of S, using substitutions from mapping.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>isalnum()</i>	Return True if the string is an alpha-numeric string, False otherwise.
<i>isalpha()</i>	Return True if the string is an alphabetic string, False otherwise.
<i>isascii()</i>	Return True if all characters in the string are ASCII, False otherwise.
<i>isdecimal()</i>	Return True if the string is a decimal string, False otherwise.
<i>isdigit()</i>	Return True if the string is a digit string, False otherwise.
<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.
<i>join</i> (iterable, /)	Concatenate any number of strings.

continues on next page

Table 6 – continued from previous page

<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower</i> (/)	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>maketrans</i> (x[, y, z])	Return a translation table usable for <code>str.translate()</code> .
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>removeprefix</i> (prefix, /)	Return a str with the given prefix string removed if present.
<i>removesuffix</i> (suffix, /)	Return a str with the given suffix string removed if present.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rpartition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>rsplit</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>split</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>splitlines</i> (/[, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase</i> (/)	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>title</i> (/)	Return a version of the string where each word is titlecased.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper</i> (/)	Return a copy of the string converted to uppercase.
<i>zfill</i> (width, /)	Pad a numeric string with zeros on the left, to fill a field of the given width.

`__init__`(*args, **kws)

Methods

<i>encode</i> (/[, encoding, errors])	Encode the string using the codec registered for encoding.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.

continues on next page

Table 7 – continued from previous page

<code>split(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>rsplit(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>join(iterable, /)</code>	Concatenate any number of strings.
<code>capitalize(/)</code>	Return a capitalized version of the string.
<code>casefold(/)</code>	Return a version of the string suitable for caseless comparisons.
<code>title(/)</code>	Return a version of the string where each word is titlecased.
<code>center(width[, fillchar])</code>	Return a centered string of length <code>width</code> .
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring <code>sub</code> in string <code>S[start:end]</code> .
<code>expandtabs(/[, tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>partition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>index(sub[, start[, end]])</code>	Return the lowest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>ljust(width[, fillchar])</code>	Return a left-justified string of length <code>width</code> .
<code>lower(/)</code>	Return a copy of the string converted to lowercase.
<code>lstrip([chars])</code>	Return a copy of the string with leading whitespace removed.
<code>rfind(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rindex(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rjust(width[, fillchar])</code>	Return a right-justified string of length <code>width</code> .
<code>rstrip([chars])</code>	Return a copy of the string with trailing whitespace removed.
<code>rpartition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>splitlines(/[, keepends])</code>	Return a list of the lines in the string, breaking at line boundaries.
<code>strip([chars])</code>	Return a copy of the string with leading and trailing whitespace removed.
<code>swapcase(/)</code>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<code>translate(table, /)</code>	Replace each character in the string using the given translation table.
<code>upper(/)</code>	Return a copy of the string converted to uppercase.
<code>startswith(prefix[, start[, end]])</code>	Return True if <code>S</code> starts with the specified prefix, False otherwise.
<code>endswith(suffix[, start[, end]])</code>	Return True if <code>S</code> ends with the specified suffix, False otherwise.
<code>removeprefix(prefix, /)</code>	Return a str with the given prefix string removed if present.
<code>removesuffix(suffix, /)</code>	Return a str with the given suffix string removed if present.

continues on next page

Table 7 – continued from previous page

<i>isascii()</i>	Return True if all characters in the string are ASCII, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.
<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>isdecimal()</i>	Return True if the string is a decimal string, False otherwise.
<i>isdigit()</i>	Return True if the string is a digit string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isalpha()</i>	Return True if the string is an alphabetic string, False otherwise.
<i>isalnum()</i>	Return True if the string is an alpha-numeric string, False otherwise.
<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>zfill(width, l)</i>	Pad a numeric string with zeros on the left, to fill a field of the given width.
<i>format(*args, **kwargs)</i>	Return a formatted version of S, using substitutions from args and kwargs.
<i>format_map(mapping)</i>	Return a formatted version of S, using substitutions from mapping.
<i>maketrans(x[, y, z])</i>	Return a translation table usable for str.translate().

Attributes

L0
SPICE
CAL
L1B
L2

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', / (Positional-only parameter separator (PEP 570)))

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(/, *encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if *S* ends with the specified suffix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(/, *tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum(/)

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha(/)

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii(/)

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal(/)

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit(/)

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier(/)

Return True if the string is a valid Python identifier, False otherwise.

Call `keyword.iskeyword(s)` to test whether string `s` is a reserved identifier, such as “def” or “class”.

islower(/)

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric(/)

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable(/)

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace(/)

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle(/)

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper(/)

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(*iterable*, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `','.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(*width*, *fillchar*=' ', /)

Return a left-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

lower(/)

Return a copy of the string converted to lowercase.

lstrip(*chars*=None, /)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

static maketrans(*x*, *y*=<unrepresentable>, *z*=<unrepresentable>, /)

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count*=-1, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(/, sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(/, sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the front of the string and works to the end.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(/, keepends=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith(prefix[, start[, end]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip(chars=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase(/)

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title(/)

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(table, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper(/)

Return a copy of the string converted to uppercase.

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

libera_utils.aws.constants.DataProductIdentifier**class libera_utils.aws.constants.DataProductIdentifier(value)**

Bases: `StrEnum`

Enumeration of data product canonical IDs used in AWS resource naming These IDs refer to the data products (files) themselves, NOT the processing steps (since processing steps may produce multiple products).

In general these names are of the form <level>-<source>-<type>

Methods

<code>capitalize(/)</code>	Return a capitalized version of the string.
<code>casefold(/)</code>	Return a version of the string suitable for caseless comparisons.
<code>center(width[, fillchar])</code>	Return a centered string of length width.
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<code>encode(/[, encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>expandtabs(/[, tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.

continues on next page

Table 9 – continued from previous page

<i>format_map</i> (mapping)	Return a formatted version of S, using substitutions from mapping.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>isalnum</i> (/)	Return True if the string is an alpha-numeric string, False otherwise.
<i>isalpha</i> (/)	Return True if the string is an alphabetic string, False otherwise.
<i>isascii</i> (/)	Return True if all characters in the string are ASCII, False otherwise.
<i>isdecimal</i> (/)	Return True if the string is a decimal string, False otherwise.
<i>isdigit</i> (/)	Return True if the string is a digit string, False otherwise.
<i>isidentifier</i> (/)	Return True if the string is a valid Python identifier, False otherwise.
<i>islower</i> (/)	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric</i> (/)	Return True if the string is a numeric string, False otherwise.
<i>isprintable</i> (/)	Return True if the string is printable, False otherwise.
<i>isspace</i> (/)	Return True if the string is a whitespace string, False otherwise.
<i>istitle</i> (/)	Return True if the string is a title-cased string, False otherwise.
<i>isupper</i> (/)	Return True if the string is an uppercase string, False otherwise.
<i>join</i> (iterable, /)	Concatenate any number of strings.
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower</i> (/)	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>maketrans</i> (x[, y, z])	Return a translation table usable for str.translate().
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>removeprefix</i> (prefix, /)	Return a str with the given prefix string removed if present.
<i>removesuffix</i> (suffix, /)	Return a str with the given suffix string removed if present.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rpartition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>rsplit</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.

continues on next page

Table 9 – continued from previous page

<i>split</i> ([, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>splitlines</i> ([, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase</i> (/)	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>title</i> (/)	Return a version of the string where each word is titlecased.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper</i> (/)	Return a copy of the string converted to uppercase.
<i>zfill</i> (width, /)	Pad a numeric string with zeros on the left, to fill a field of the given width.

`__init__(*args, **kws)`

Methods

<i>encode</i> ([, encoding, errors])	Encode the string using the codec registered for encoding.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
<i>split</i> ([, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>rsplit</i> ([, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>join</i> (iterable, /)	Concatenate any number of strings.
<i>capitalize</i> (/)	Return a capitalized version of the string.
<i>casefold</i> (/)	Return a version of the string suitable for caseless comparisons.
<i>title</i> (/)	Return a version of the string where each word is titlecased.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>expandtabs</i> ([, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower</i> (/)	Return a copy of the string converted to lowercase.

continues on next page

Table 10 – continued from previous page

<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>rpartition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>splitlines</i> ([, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase</i> (/)	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper</i> (/)	Return a copy of the string converted to uppercase.
<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>removeprefix</i> (prefix, /)	Return a str with the given prefix string removed if present.
<i>removesuffix</i> (suffix, /)	Return a str with the given suffix string removed if present.
<i>isascii</i> (/)	Return True if all characters in the string are ASCII, False otherwise.
<i>islower</i> (/)	Return True if the string is a lowercase string, False otherwise.
<i>isupper</i> (/)	Return True if the string is an uppercase string, False otherwise.
<i>istitle</i> (/)	Return True if the string is a title-cased string, False otherwise.
<i>isspace</i> (/)	Return True if the string is a whitespace string, False otherwise.
<i>isdecimal</i> (/)	Return True if the string is a decimal string, False otherwise.
<i>isdigit</i> (/)	Return True if the string is a digit string, False otherwise.
<i>isnumeric</i> (/)	Return True if the string is a numeric string, False otherwise.
<i>isalpha</i> (/)	Return True if the string is an alphabetic string, False otherwise.
<i>isalnum</i> (/)	Return True if the string is an alpha-numeric string, False otherwise.
<i>isidentifier</i> (/)	Return True if the string is a valid Python identifier, False otherwise.
<i>isprintable</i> (/)	Return True if the string is printable, False otherwise.

continues on next page

Table 10 – continued from previous page

<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>maketrans(x[, y, z])</code>	Return a translation table usable for <code>str.translate()</code> .
<code>validate(product_name)</code>	Validate a product name string used by the DAG or the processing orchestration system.
<code>dump([chunk_number])</code>	Convert the <code>DataProductIdentifier</code> to a string suitable for matching with a DAG key or in the processing orchestration system.

Attributes

<code>l0_cr</code>
<code>l0_rad_pds</code>
<code>l0_cam_pds</code>
<code>l0_azel_pds</code>
<code>l0_jpss_pds</code>
<code>spice_az_ck</code>
<code>spice_el_ck</code>
<code>spice_jpss_ck</code>
<code>spice_jpss_spk</code>
<code>cal_rad</code>
<code>cal_cam</code>
<code>l1b_rad</code>
<code>l1b_cam</code>
<code>anc_adm</code>

capitalize(/)

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold(/)

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

dump(*chunk_number*: int | None = None) → str

Convert the DataProductIdentifier to a string suitable for matching with a DAG key or in the processing orchestration system.

The *chunk_number* can be specified when the data product represents a PDS file that is typically provided in 12 2-hour chunks per day. In that case, the *chunk_number* appears as a suffix to the orchestration product name.

encode(/, *encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with `codecs.register_error` that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if *S* ends with the specified *suffix*, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(/, *tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call `keyword.iskeyword(s)` to test whether string `s` is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper(/)

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower(/)

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans(x, y=<unrepresentable>, z=<unrepresentable>, /)

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(suffix, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(old, new, count=-1, /)

Return a copy with all occurrences of substring old replaced by new.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument count is given, only the first count occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in S where substring *sub* is found, such that *sub* is contained within S[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in S where substring *sub* is found, such that *sub* is contained within S[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises ValueError when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(/, *sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

split(/, *sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the front of the string and works to the end.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(/, *keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None*, /)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase(/)

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title(/)

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper(/)

Return a copy of the string converted to uppercase.

classmethod validate(*product_name: str*) → tuple[*DataProductIdentifier*, int | None]

Validate a product name string used by the DAG or the processing orchestration system.

If successful, returns a tuple containing the `DataProductIdentifier` and the `chunk_number`, which can be None if the input string does not contain a valid chunk number.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

libera_utils.aws.constants.LiberaAccountSuffix

class libera_utils.aws.constants.LiberaAccountSuffix(*value*)

Bases: `StrEnum`

Suffixes for the various account types

Methods

<i>capitalize()</i>	Return a capitalized version of the string.
<i>casefold()</i>	Return a version of the string suitable for caseless comparisons.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>encode</i> (/[, encoding, errors])	Encode the string using the codec registered for encoding.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>expandtabs</i> (/[, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>format</i> (*args, **kwargs)	Return a formatted version of S, using substitutions from args and kwargs.
<i>format_map</i> (mapping)	Return a formatted version of S, using substitutions from mapping.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>isalnum()</i>	Return True if the string is an alpha-numeric string, False otherwise.
<i>isalpha()</i>	Return True if the string is an alphabetic string, False otherwise.
<i>isascii()</i>	Return True if all characters in the string are ASCII, False otherwise.
<i>isdecimal()</i>	Return True if the string is a decimal string, False otherwise.
<i>isdigit()</i>	Return True if the string is a digit string, False otherwise.
<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.
<i>join</i> (iterable, /)	Concatenate any number of strings.
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower()</i>	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>maketrans</i> (x[, y, z])	Return a translation table usable for str.translate().
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.

continues on next page

Table 12 – continued from previous page

<code>removeprefix(prefix, /)</code>	Return a str with the given prefix string removed if present.
<code>removesuffix(suffix, /)</code>	Return a str with the given suffix string removed if present.
<code>replace(old, new[, count])</code>	Return a copy with all occurrences of substring <code>old</code> replaced by <code>new</code> .
<code>rfind(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rindex(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rjust(width[, fillchar])</code>	Return a right-justified string of length <code>width</code> .
<code>rpartition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>rsplit(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>rstrip([chars])</code>	Return a copy of the string with trailing whitespace removed.
<code>split(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>splitlines(/[, keepends])</code>	Return a list of the lines in the string, breaking at line boundaries.
<code>startswith(prefix[, start[, end]])</code>	Return <code>True</code> if <code>S</code> starts with the specified prefix, <code>False</code> otherwise.
<code>strip([chars])</code>	Return a copy of the string with leading and trailing whitespace removed.
<code>swapcase(/)</code>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<code>title(/)</code>	Return a version of the string where each word is titlecased.
<code>translate(table, /)</code>	Replace each character in the string using the given translation table.
<code>upper(/)</code>	Return a copy of the string converted to uppercase.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.

`__init__(*args, **kws)`

Methods

<code>encode(/[, encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>replace(old, new[, count])</code>	Return a copy with all occurrences of substring <code>old</code> replaced by <code>new</code> .
<code>split(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>rsplit(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>join(iterable, /)</code>	Concatenate any number of strings.
<code>capitalize(/)</code>	Return a capitalized version of the string.

continues on next page

Table 13 – continued from previous page

<i>casefold()</i>	Return a version of the string suitable for caseless comparisons.
<i>title()</i>	Return a version of the string where each word is titlecased.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>expandtabs</i> (/[, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower()</i>	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>rpartition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>splitlines</i> (/[, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase()</i>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper()</i>	Return a copy of the string converted to uppercase.
<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>removeprefix</i> (prefix, /)	Return a str with the given prefix string removed if present.
<i>removesuffix</i> (suffix, /)	Return a str with the given suffix string removed if present.
<i>isascii()</i>	Return True if all characters in the string are ASCII, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.

continues on next page

Table 13 – continued from previous page

<code>istitle()</code>	Return True if the string is a title-cased string, False otherwise.
<code>isspace()</code>	Return True if the string is a whitespace string, False otherwise.
<code>isdecimal()</code>	Return True if the string is a decimal string, False otherwise.
<code>isdigit()</code>	Return True if the string is a digit string, False otherwise.
<code>isnumeric()</code>	Return True if the string is a numeric string, False otherwise.
<code>isalpha()</code>	Return True if the string is an alphabetic string, False otherwise.
<code>isalnum()</code>	Return True if the string is an alpha-numeric string, False otherwise.
<code>isidentifier()</code>	Return True if the string is a valid Python identifier, False otherwise.
<code>isprintable()</code>	Return True if the string is printable, False otherwise.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>maketrans(x[, y, z])</code>	Return a translation table usable for <code>str.translate()</code> .

Attributes

STAGE
PROD
DEV
TEST

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

encode(/, encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(suffix[, start[, end]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(/, tabsize=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(*args, **kwargs) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(mapping) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum(/)

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha(/)

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii(/)

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(*width*, *fillchar*=' ', /)

Return a left-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

lower(/)

Return a copy of the string converted to lowercase.

lstrip(*chars*=None, /)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

static maketrans(*x*, *y*=<unrepresentable>, *z*=<unrepresentable>, /)

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count*=-1, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(/, sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(/, sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the front of the string and works to the end.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(/, keepends=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith(prefix[, start[, end]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip(chars=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase(/)

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title(/)

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(table, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper(/)

Return a copy of the string converted to uppercase.

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

libera_utils.aws.constants.LiberaApid

class libera_utils.aws.constants.LiberaApid(*value*)

Bases: `Enum`

APIDs for L0 packets

`__init__`(*args, **kws)

Attributes

JPSS_ATTITUDE_EPHEMERIS

FILTERED_RADIOMETER

FILTERED_AZEL

CAMERA

libera_utils.aws.constants.LiberaDataBucketName

class libera_utils.aws.constants.LiberaDataBucketName(*value*)

Bases: `StrEnum`

Names of the data archive buckets

Methods

<i>capitalize()</i>	Return a capitalized version of the string.
<i>casefold()</i>	Return a version of the string suitable for caseless comparisons.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>encode</i> (/[, encoding, errors])	Encode the string using the codec registered for encoding.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>expandtabs</i> (/[, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>format</i> (*args, **kwargs)	Return a formatted version of S, using substitutions from args and kwargs.
<i>format_map</i> (mapping)	Return a formatted version of S, using substitutions from mapping.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>isalnum()</i>	Return True if the string is an alpha-numeric string, False otherwise.
<i>isalpha()</i>	Return True if the string is an alphabetic string, False otherwise.
<i>isascii()</i>	Return True if all characters in the string are ASCII, False otherwise.
<i>isdecimal()</i>	Return True if the string is a decimal string, False otherwise.
<i>isdigit()</i>	Return True if the string is a digit string, False otherwise.
<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.
<i>join</i> (iterable, /)	Concatenate any number of strings.
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower()</i>	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>maketrans</i> (x[, y, z])	Return a translation table usable for str.translate().
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.

continues on next page

Table 16 – continued from previous page

<i>removeprefix</i> (prefix, /)	Return a str with the given prefix string removed if present.
<i>removesuffix</i> (suffix, /)	Return a str with the given suffix string removed if present.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rpartition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>rsplit</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>split</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>splitlines</i> (/[, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase</i> (/)	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>title</i> (/)	Return a version of the string where each word is titlecased.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper</i> (/)	Return a copy of the string converted to uppercase.
<i>zfill</i> (width, /)	Pad a numeric string with zeros on the left, to fill a field of the given width.

`__init__`(*args, **kws)

Methods

<i>encode</i> (/[, encoding, errors])	Encode the string using the codec registered for encoding.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
<i>split</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>rsplit</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>join</i> (iterable, /)	Concatenate any number of strings.
<i>capitalize</i> (/)	Return a capitalized version of the string.

continues on next page

Table 17 – continued from previous page

<i>casefold()</i>	Return a version of the string suitable for caseless comparisons.
<i>title()</i>	Return a version of the string where each word is titlecased.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>expandtabs</i> (/[, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower()</i>	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>rpartition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>splitlines</i> (/[, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase()</i>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper()</i>	Return a copy of the string converted to uppercase.
<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>removeprefix</i> (prefix, /)	Return a str with the given prefix string removed if present.
<i>removesuffix</i> (suffix, /)	Return a str with the given suffix string removed if present.
<i>isascii()</i>	Return True if all characters in the string are ASCII, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.

continues on next page

Table 17 – continued from previous page

<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>isdecimal()</i>	Return True if the string is a decimal string, False otherwise.
<i>isdigit()</i>	Return True if the string is a digit string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isalpha()</i>	Return True if the string is an alphabetic string, False otherwise.
<i>isalnum()</i>	Return True if the string is an alpha-numeric string, False otherwise.
<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>zfill(width, /)</i>	Pad a numeric string with zeros on the left, to fill a field of the given width.
<i>format(*args, **kwargs)</i>	Return a formatted version of S, using substitutions from args and kwargs.
<i>format_map(mapping)</i>	Return a formatted version of S, using substitutions from mapping.
<i>maketrans(x[, y, z])</i>	Return a translation table usable for str.translate().

Attributes

L0_ARCHIVE_BUCKET
SPICE_ARCHIVE_BUCKET
ANCILLARY_ARCHIVE_BUCKET
L1B_ARCHIVE_BUCKET
L2_ARCHIVE_BUCKET

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

encode(/, *encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with `codecs.register_error` that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → bool

Return True if *S* ends with the specified suffix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs(/, *tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum(/)

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha(/)

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii(/)

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(*width*, *fillchar*=' ', /)

Return a left-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

lower(/)

Return a copy of the string converted to lowercase.

lstrip(*chars*=None, /)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

static maketrans(*x*, *y*=<unrepresentable>, *z*=<unrepresentable>, /)

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count*=-1, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(width, fillchar=' ', /)

Return a right-justified string of length width.

Padding is done using the specified fill character (default is a space).

rpartition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(/, sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(chars=None, /)

Return a copy of the string with trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

split(/, sep=None, maxsplit=-1)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including n r t f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the front of the string and works to the end.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(/, keepends=False)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless keepends is given and true.

startswith(prefix[, start[, end]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. prefix can also be a tuple of strings to try.

strip(chars=None, /)

Return a copy of the string with leading and trailing whitespace removed.

If chars is given and not None, remove characters in chars instead.

swapcase(/)

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title(/)

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(table, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper(/)

Return a copy of the string converted to uppercase.

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

libera_utils.aws.constants.ManifestType

class libera_utils.aws.constants.**ManifestType**(value)

Bases: `StrEnum`

Enumerated legal manifest type values

Methods

<code>capitalize(/)</code>	Return a capitalized version of the string.
<code>casefold(/)</code>	Return a version of the string suitable for caseless comparisons.
<code>center(width[, fillchar])</code>	Return a centered string of length width.
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<code>encode(/[, encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>expandtabs(/[, tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.

continues on next page

Table 19 – continued from previous page

<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>isalnum</i> (/)	Return True if the string is an alpha-numeric string, False otherwise.
<i>isalpha</i> (/)	Return True if the string is an alphabetic string, False otherwise.
<i>isascii</i> (/)	Return True if all characters in the string are ASCII, False otherwise.
<i>isdecimal</i> (/)	Return True if the string is a decimal string, False otherwise.
<i>isdigit</i> (/)	Return True if the string is a digit string, False otherwise.
<i>isidentifier</i> (/)	Return True if the string is a valid Python identifier, False otherwise.
<i>islower</i> (/)	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric</i> (/)	Return True if the string is a numeric string, False otherwise.
<i>isprintable</i> (/)	Return True if the string is printable, False otherwise.
<i>isspace</i> (/)	Return True if the string is a whitespace string, False otherwise.
<i>istitle</i> (/)	Return True if the string is a title-cased string, False otherwise.
<i>isupper</i> (/)	Return True if the string is an uppercase string, False otherwise.
<i>join</i> (iterable, /)	Concatenate any number of strings.
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower</i> (/)	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>maketrans</i> (x[, y, z])	Return a translation table usable for str.translate().
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>removeprefix</i> (prefix, /)	Return a str with the given prefix string removed if present.
<i>removesuffix</i> (suffix, /)	Return a str with the given suffix string removed if present.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rpartition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>rsplit</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>split</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.

continues on next page

Table 19 – continued from previous page

<i>splitlines</i> ([, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase</i> (/)	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>title</i> (/)	Return a version of the string where each word is titlecased.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper</i> (/)	Return a copy of the string converted to uppercase.
<i>zfill</i> (width, /)	Pad a numeric string with zeros on the left, to fill a field of the given width.

`__init__(*args, **kws)`

Methods

<i>encode</i> ([, encoding, errors])	Encode the string using the codec registered for encoding.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
<i>split</i> ([, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>rsplit</i> ([, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>join</i> (iterable, /)	Concatenate any number of strings.
<i>capitalize</i> (/)	Return a capitalized version of the string.
<i>casefold</i> (/)	Return a version of the string suitable for caseless comparisons.
<i>title</i> (/)	Return a version of the string where each word is titlecased.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>expandtabs</i> ([, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower</i> (/)	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.

continues on next page

Table 20 – continued from previous page

<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>rpartition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>splitlines</i> ([, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase</i> (/)	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper</i> (/)	Return a copy of the string converted to uppercase.
<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>removeprefix</i> (prefix, /)	Return a str with the given prefix string removed if present.
<i>removesuffix</i> (suffix, /)	Return a str with the given suffix string removed if present.
<i>isascii</i> (/)	Return True if all characters in the string are ASCII, False otherwise.
<i>islower</i> (/)	Return True if the string is a lowercase string, False otherwise.
<i>isupper</i> (/)	Return True if the string is an uppercase string, False otherwise.
<i>istitle</i> (/)	Return True if the string is a title-cased string, False otherwise.
<i>isspace</i> (/)	Return True if the string is a whitespace string, False otherwise.
<i>isdecimal</i> (/)	Return True if the string is a decimal string, False otherwise.
<i>isdigit</i> (/)	Return True if the string is a digit string, False otherwise.
<i>isnumeric</i> (/)	Return True if the string is a numeric string, False otherwise.
<i>isalpha</i> (/)	Return True if the string is an alphabetic string, False otherwise.
<i>isalnum</i> (/)	Return True if the string is an alpha-numeric string, False otherwise.
<i>isidentifier</i> (/)	Return True if the string is a valid Python identifier, False otherwise.
<i>isprintable</i> (/)	Return True if the string is printable, False otherwise.
<i>zfill</i> (width, /)	Pad a numeric string with zeros on the left, to fill a field of the given width.

continues on next page

Table 20 – continued from previous page

<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>maketrans(x[, y, z])</code>	Return a translation table usable for <code>str.translate()</code> .

Attributes

INPUT
<code>input</code>
OUTPUT
<code>output</code>

`capitalize()`

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

`casefold()`

Return a version of the string suitable for caseless comparisons.

`center(width, fillchar=' ', /)`

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

`count(sub[, start[, end]]) → int`

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

`encode(/, encoding='utf-8', errors='strict')`

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a `UnicodeEncodeError`. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with `codecs.register_error` that can handle `UnicodeEncodeErrors`.

`endswith(suffix[, start[, end]]) → bool`

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

`expandtabs(/, tabsize=8)`

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → str

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str

Return a formatted version of *S*, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

index(*sub*[, *start*[, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

isalnum(/)

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha(/)

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii(/)

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal(/)

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit(/)

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier(/)

Return True if the string is a valid Python identifier, False otherwise.

Call `keyword.iskeyword(s)` to test whether string *s* is a reserved identifier, such as “def” or “class”.

islower(/)

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

lstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans(x, y=<unrepresentable>, z=<unrepresentable>, /)

Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return string[len(prefix):]. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar=' '*, /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(/, *sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to `None` (the default value), will split on any whitespace character (including `\n` `\r` `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars=None, /*)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

split(*/, sep=None, maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to *None* (the default value), will split on any whitespace character (including `n r t f` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the front of the string and works to the end.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(*/, keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix[, start[, end]]*) → bool

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None, /*)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not *None*, remove characters in *chars* instead.

swapcase(*/*)

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title(*/*)

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table, /*)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or *None*.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to *None* are deleted.

upper(/)

Return a copy of the string converted to uppercase.

zfill(width, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

libera_utils.aws.constants.ProcessingStepIdentifier

class libera_utils.aws.constants.ProcessingStepIdentifier(*value*)

Bases: `StrEnum`

Enumeration of processing step IDs used in AWS resource naming and processing orchestration

In orchestration code, these are used as “NodeID” values to identify processing steps:

The `processing_step_node_id` values used in `libera_cdk` deployment stackbuilder module and the node names in `processing_system_dag.json` must match these.

They must also be passed to the `ecr_upload` module called by some `libera_cdk` integration tests.

Methods

<code>capitalize(/)</code>	Return a capitalized version of the string.
<code>casefold(/)</code>	Return a version of the string suitable for caseless comparisons.
<code>center(width[, fillchar])</code>	Return a centered string of length width.
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<code>encode(/[, encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>expandtabs(/[, tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>index(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>isalnum(/)</code>	Return True if the string is an alpha-numeric string, False otherwise.
<code>isalpha(/)</code>	Return True if the string is an alphabetic string, False otherwise.
<code>isascii(/)</code>	Return True if all characters in the string are ASCII, False otherwise.
<code>isdecimal(/)</code>	Return True if the string is a decimal string, False otherwise.
<code>isdigit(/)</code>	Return True if the string is a digit string, False otherwise.

continues on next page

Table 22 – continued from previous page

<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.
<i>join(iterable, /)</i>	Concatenate any number of strings.
<i>ljust(width[, fillchar])</i>	Return a left-justified string of length width.
<i>lower()</i>	Return a copy of the string converted to lowercase.
<i>lstrip([chars])</i>	Return a copy of the string with leading whitespace removed.
<i>maketrans(x[, y, z])</i>	Return a translation table usable for <code>str.translate()</code> .
<i>partition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>removeprefix(prefix, /)</i>	Return a str with the given prefix string removed if present.
<i>removesuffix(suffix, /)</i>	Return a str with the given suffix string removed if present.
<i>replace(old, new[, count])</i>	Return a copy with all occurrences of substring <code>old</code> replaced by <code>new</code> .
<i>rfind(sub[, start[, end]])</i>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<i>rindex(sub[, start[, end]])</i>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<i>rjust(width[, fillchar])</i>	Return a right-justified string of length width.
<i>rpartition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>rsplit(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<i>rstrip([chars])</i>	Return a copy of the string with trailing whitespace removed.
<i>split(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<i>splitlines(/[, keepends])</i>	Return a list of the lines in the string, breaking at line boundaries.
<i>startswith(prefix[, start[, end]])</i>	Return True if <code>S</code> starts with the specified prefix, False otherwise.
<i>strip([chars])</i>	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase()</i>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>title()</i>	Return a version of the string where each word is titlecased.
<i>translate(table, /)</i>	Replace each character in the string using the given translation table.

continues on next page

Table 22 – continued from previous page

<i>upper</i> (<i>l</i>)	Return a copy of the string converted to uppercase.
<i>zfill</i> (width, <i>l</i>)	Pad a numeric string with zeros on the left, to fill a field of the given width.
__init__ (*args, **kws)	
Methods	
<i>encode</i> (/[, encoding, errors])	Encode the string using the codec registered for encoding.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
<i>split</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>rsplit</i> (/[, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>join</i> (iterable, <i>l</i>)	Concatenate any number of strings.
<i>capitalize</i> (<i>l</i>)	Return a capitalized version of the string.
<i>casefold</i> (<i>l</i>)	Return a version of the string suitable for caseless comparisons.
<i>title</i> (<i>l</i>)	Return a version of the string where each word is titlecased.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>expandtabs</i> (/[, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>partition</i> (sep, <i>l</i>)	Partition the string into three parts using the given separator.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower</i> (<i>l</i>)	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>rpartition</i> (sep, <i>l</i>)	Partition the string into three parts using the given separator.
<i>splitlines</i> (/[, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.

continues on next page

Table 23 – continued from previous page

<code>swapcase()</code>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<code>translate(table, /)</code>	Replace each character in the string using the given translation table.
<code>upper()</code>	Return a copy of the string converted to uppercase.
<code>startswith(prefix[, start[, end]])</code>	Return True if S starts with the specified prefix, False otherwise.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>removeprefix(prefix, /)</code>	Return a str with the given prefix string removed if present.
<code>removesuffix(suffix, /)</code>	Return a str with the given suffix string removed if present.
<code>isascii()</code>	Return True if all characters in the string are ASCII, False otherwise.
<code>islower()</code>	Return True if the string is a lowercase string, False otherwise.
<code>isupper()</code>	Return True if the string is an uppercase string, False otherwise.
<code>istitle()</code>	Return True if the string is a title-cased string, False otherwise.
<code>isspace()</code>	Return True if the string is a whitespace string, False otherwise.
<code>isdecimal()</code>	Return True if the string is a decimal string, False otherwise.
<code>isdigit()</code>	Return True if the string is a digit string, False otherwise.
<code>isnumeric()</code>	Return True if the string is a numeric string, False otherwise.
<code>isalpha()</code>	Return True if the string is an alphabetic string, False otherwise.
<code>isalnum()</code>	Return True if the string is an alpha-numeric string, False otherwise.
<code>isidentifier()</code>	Return True if the string is a valid Python identifier, False otherwise.
<code>isprintable()</code>	Return True if the string is printable, False otherwise.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>maketrans(x[, y, z])</code>	Return a translation table usable for <code>str.translate()</code> .
<code>validate(processing_step)</code>	Validate a processing step string used by the DAG or the orchestration system.
<code>get_archive_bucket_name([account_suffix])</code>	Gets the archive bucket name for this processing step.
<code>dump([chunk_number])</code>	Convert the ProcessingStepIdentifier to a string suitable for matching with a DAG key or in the processing orchestration system.

Attributes

<code>ecr_name</code>	Get the manually-configured ECR name for this processing step
<code>l2_cam_cf</code>	
<code>l2_rad_ssw_toa</code>	
<code>adms</code>	
<code>l2_surface_flux</code>	
<code>l2_unfiltered</code>	
<code>spice_azel</code>	
<code>spice_jpss</code>	
<code>l1b_rad</code>	
<code>l1b_cam</code>	
<code>l0_jpss_pds</code>	
<code>l0_azel_pds</code>	
<code>l0_rad_pds</code>	
<code>l0_cam_pds</code>	
<code>l0_cr</code>	
<code>cal_rad</code>	
<code>cal_cam</code>	

capitalize()

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold()

Return a version of the string suitable for caseless comparisons.

center(*width*, *fillchar*=' ', /)

Return a centered string of length *width*.

Padding is done using the specified fill character (default is a space).

count(*sub*[, *start*[, *end*]]) → int

Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

dump(*chunk_number*: *int* | *None* = *None*) → *str*

Convert the ProcessingStepIdentifier to a string suitable for matching with a DAG key or in the processing orchestration system.

The *chunk_number* can be specified when the data product represents a PDS file that is typically provided in 12 2-hour chunks per day. In that case, the *chunk_number* appears as a suffix to the orchestration step identifier

property ecr_name: *str* | *None*

Get the manually-configured ECR name for this processing step

We name our ECRs in CDK because they are one of the few resources that humans will need to interact with on a regular basis.

encode(/, *encoding*='utf-8', *errors*='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(*suffix*[, *start*[, *end*]]) → *bool*

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. *suffix* can also be a tuple of strings to try.

expandtabs(/, *tabsize*=8)

Return a copy where all tab characters are expanded using spaces.

If *tabsize* is not given, a tab size of 8 characters is assumed.

find(*sub*[, *start*[, *end*]]) → *int*

Return the lowest index in S where substring *sub* is found, such that *sub* is contained within S[start:end]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format(**args*, ***kwargs*) → *str*

Return a formatted version of S, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → *str*

Return a formatted version of S, using substitutions from *mapping*. The substitutions are identified by braces ('{' and '}').

get_archive_bucket_name(*account_suffix*: *LiberaAccountSuffix* = *LiberaAccountSuffix.STAGE*) → *str* | *None*

Gets the archive bucket name for this processing step.

Buckets are named according to the level of data they are storing and which account they are in. This is expected to be used by the L2 developers who will most commonly be working with the stage account.

Parameters

account_suffix (*LiberaAccountSuffix*, *optional*) – Account suffix for the bucket name, by default *LiberaAccountSuffix.STAGE* (stage account)

Returns

The name of the archive bucket for this processing step

Return type

str

index(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum(/)

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha(/)

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii(/)

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal(/)

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit(/)

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier(/)

Return True if the string is a valid Python identifier, False otherwise.

Call keyword.iskeyword(s) to test whether string s is a reserved identifier, such as “def” or “class”.

islower(/)

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric(/)

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable(/)

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in repr() or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper()

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(iterable, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `','.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(width, fillchar=' ', /)

Return a left-justified string of length width.

Padding is done using the specified fill character (default is a space).

lower()

Return a copy of the string converted to lowercase.

rstrip(chars=None, /)

Return a copy of the string with leading whitespace removed.

If chars is given and not None, remove characters in chars instead.

static maketrans(x, y=<unrepresentable>, z=<unrepresentable>, /)

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(sep, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

removeprefix(prefix, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return string[:-len(suffix)]. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count=-1*, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar=' '*, /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(/, *sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to `None` (the default value), will split on any whitespace character (including `\n` `\r` `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars=None*, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not `None`, remove characters in *chars* instead.

split(/, *sep=None*, *maxsplit=-1*)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including `\n`, `\r`, `\t` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the front of the string and works to the end.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(/, *keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless `keepends` is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. `prefix` can also be a tuple of strings to try.

strip(*chars=None*, /)

Return a copy of the string with leading and trailing whitespace removed.

If `chars` is given and not None, remove characters in `chars` instead.

swapcase(/)

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title(/)

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper(/)

Return a copy of the string converted to uppercase.

classmethod validate(*processing_step: str*) → tuple[*ProcessingStepIdentifier*, int | None]

Validate a processing step string used by the DAG or the orchestration system.

If successful, returns a tuple containing the `ProcessingStepIdentifier` and the `chunk_number`, which can be None if the input string does not contain a valid chunk number.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

libera_utils.aws.constants.SpKObject**class** libera_utils.aws.constants.SpKObject(*value*)

Bases: Enum

Enum of valid SPK objects

`__init__(*args, **kwargs)`**Attributes**

<code>data_product_id</code>	DataProductIdentifier for SPKs associated with this SPK object
<code>processing_step_id</code>	ProcessingStepIdentifier for the processing step that produces SPKs for this SPK object
JPSS	

property `data_product_id`: *DataProductIdentifier*

DataProductIdentifier for SPKs associated with this SPK object

property `processing_step_id`: *ProcessingStepIdentifier*

ProcessingStepIdentifier for the processing step that produces SPKs for this SPK object

class libera_utils.aws.constants.CkObject(*value*)

Enum of valid CK objects

property `data_product_id`: *DataProductIdentifier*

DataProductIdentifier for CKs associated with this CK object

property `processing_step_id`: *ProcessingStepIdentifier*

ProcessingStepIdentifier for the processing step that produces CKs for this CK object

class libera_utils.aws.constants.DataLevel(*value*)

Data product level

Methods

<code>capitalize()</code>	Return a capitalized version of the string.
<code>casefold()</code>	Return a version of the string suitable for caseless comparisons.
<code>center(width[, fillchar])</code>	Return a centered string of length width.
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<code>encode([, encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>expandtabs([, tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].

continues on next page

Table 26 – continued from previous page

<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>index(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>isalnum()</code>	Return True if the string is an alpha-numeric string, False otherwise.
<code>isalpha()</code>	Return True if the string is an alphabetic string, False otherwise.
<code>isascii()</code>	Return True if all characters in the string are ASCII, False otherwise.
<code>isdecimal()</code>	Return True if the string is a decimal string, False otherwise.
<code>isdigit()</code>	Return True if the string is a digit string, False otherwise.
<code>isidentifier()</code>	Return True if the string is a valid Python identifier, False otherwise.
<code>islower()</code>	Return True if the string is a lowercase string, False otherwise.
<code>isnumeric()</code>	Return True if the string is a numeric string, False otherwise.
<code>isprintable()</code>	Return True if the string is printable, False otherwise.
<code>isspace()</code>	Return True if the string is a whitespace string, False otherwise.
<code>istitle()</code>	Return True if the string is a title-cased string, False otherwise.
<code>isupper()</code>	Return True if the string is an uppercase string, False otherwise.
<code>join(iterable, /)</code>	Concatenate any number of strings.
<code>ljust(width[, fillchar])</code>	Return a left-justified string of length width.
<code>lower()</code>	Return a copy of the string converted to lowercase.
<code>lstrip([chars])</code>	Return a copy of the string with leading whitespace removed.
<code>maketrans(x[, y, z])</code>	Return a translation table usable for str.translate().
<code>partition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>removeprefix(prefix, /)</code>	Return a str with the given prefix string removed if present.
<code>removesuffix(suffix, /)</code>	Return a str with the given suffix string removed if present.
<code>replace(old, new[, count])</code>	Return a copy with all occurrences of substring old replaced by new.
<code>rfind(sub[, start[, end]])</code>	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>rindex(sub[, start[, end]])</code>	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>rjust(width[, fillchar])</code>	Return a right-justified string of length width.
<code>rpartition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>rsplit(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using sep as the separator string.

continues on next page

Table 26 – continued from previous page

<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>split</i> ([, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>splitlines</i> ([, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase</i> ()	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>title</i> ()	Return a version of the string where each word is titlecased.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper</i> ()	Return a copy of the string converted to uppercase.
<i>zfill</i> (width, /)	Pad a numeric string with zeros on the left, to fill a field of the given width.

class `libera_utils.aws.constants.DataProductIdentifier`(*value*)

Enumeration of data product canonical IDs used in AWS resource naming. These IDs refer to the data products (files) themselves, NOT the processing steps (since processing steps may produce multiple products).

In general these names are of the form <level>-<source>-<type>

Methods

<i>capitalize</i> ()	Return a capitalized version of the string.
<i>casefold</i> ()	Return a version of the string suitable for caseless comparisons.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>encode</i> ([, encoding, errors])	Encode the string using the codec registered for encoding.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>expandtabs</i> ([, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>format</i> (*args, **kwargs)	Return a formatted version of S, using substitutions from args and kwargs.
<i>format_map</i> (mapping)	Return a formatted version of S, using substitutions from mapping.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>isalnum</i> ()	Return True if the string is an alpha-numeric string, False otherwise.

continues on next page

Table 27 – continued from previous page

<i>isalpha()</i>	Return True if the string is an alphabetic string, False otherwise.
<i>isascii()</i>	Return True if all characters in the string are ASCII, False otherwise.
<i>isdecimal()</i>	Return True if the string is a decimal string, False otherwise.
<i>isdigit()</i>	Return True if the string is a digit string, False otherwise.
<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.
<i>join(iterable, /)</i>	Concatenate any number of strings.
<i>ljust(width[, fillchar])</i>	Return a left-justified string of length width.
<i>lower()</i>	Return a copy of the string converted to lowercase.
<i>lstrip([chars])</i>	Return a copy of the string with leading whitespace removed.
<i>maketrans(x[, y, z])</i>	Return a translation table usable for <code>str.translate()</code> .
<i>partition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>removeprefix(prefix, /)</i>	Return a str with the given prefix string removed if present.
<i>removesuffix(suffix, /)</i>	Return a str with the given suffix string removed if present.
<i>replace(old, new[, count])</i>	Return a copy with all occurrences of substring <code>old</code> replaced by <code>new</code> .
<i>rfind(sub[, start[, end]])</i>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<i>rindex(sub[, start[, end]])</i>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<i>rjust(width[, fillchar])</i>	Return a right-justified string of length width.
<i>rpartition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>rsplit(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<i>rstrip([chars])</i>	Return a copy of the string with trailing whitespace removed.
<i>split(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<i>splitlines(/[, keepends])</i>	Return a list of the lines in the string, breaking at line boundaries.
<i>startswith(prefix[, start[, end]])</i>	Return True if <code>S</code> starts with the specified prefix, False otherwise.

continues on next page

Table 27 – continued from previous page

<code>strip([chars])</code>	Return a copy of the string with leading and trailing whitespace removed.
<code>swapcase()</code>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<code>title()</code>	Return a version of the string where each word is titlecased.
<code>translate(table, /)</code>	Replace each character in the string using the given translation table.
<code>upper()</code>	Return a copy of the string converted to uppercase.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.

dump(*chunk_number: int | None = None*) → str

Convert the DataProductIdentifier to a string suitable for matching with a DAG key or in the processing orchestration system.

The `chunk_number` can be specified when the data product represents a PDS file that is typically provided in 12 2-hour chunks per day. In that case, the `chunk_number` appears as a suffix to the orchestration product name.

classmethod validate(*product_name: str*) → tuple[DataProductIdentifier, int | None]

Validate a product name string used by the DAG or the processing orchestration system.

If successful, returns a tuple containing the DataProductIdentifier and the `chunk_number`, which can be None if the input string does not contain a valid chunk number.

class libera_utils.aws.constants.LiberaAccountSuffix(*value*)

Suffixes for the various account types

Methods

<code>capitalize()</code>	Return a capitalized version of the string.
<code>casefold()</code>	Return a version of the string suitable for caseless comparisons.
<code>center(width[, fillchar])</code>	Return a centered string of length width.
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<code>encode(/[, encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>expandtabs(/[, tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>index(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].

continues on next page

Table 28 – continued from previous page

<i>isalnum()</i>	Return True if the string is an alpha-numeric string, False otherwise.
<i>isalpha()</i>	Return True if the string is an alphabetic string, False otherwise.
<i>isascii()</i>	Return True if all characters in the string are ASCII, False otherwise.
<i>isdecimal()</i>	Return True if the string is a decimal string, False otherwise.
<i>isdigit()</i>	Return True if the string is a digit string, False otherwise.
<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.
<i>join(iterable, /)</i>	Concatenate any number of strings.
<i>ljust(width[, fillchar])</i>	Return a left-justified string of length width.
<i>lower()</i>	Return a copy of the string converted to lowercase.
<i>lstrip([chars])</i>	Return a copy of the string with leading whitespace removed.
<i>maketrans(x[, y, z])</i>	Return a translation table usable for <code>str.translate()</code> .
<i>partition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>removeprefix(prefix, /)</i>	Return a str with the given prefix string removed if present.
<i>removesuffix(suffix, /)</i>	Return a str with the given suffix string removed if present.
<i>replace(old, new[, count])</i>	Return a copy with all occurrences of substring <code>old</code> replaced by <code>new</code> .
<i>rfind(sub[, start[, end]])</i>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<i>rindex(sub[, start[, end]])</i>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<i>rjust(width[, fillchar])</i>	Return a right-justified string of length width.
<i>rpartition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>rsplit(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<i>rstrip([chars])</i>	Return a copy of the string with trailing whitespace removed.
<i>split(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<i>splitlines(/[, keepends])</i>	Return a list of the lines in the string, breaking at line boundaries.

continues on next page

Table 28 – continued from previous page

<i>startswith</i> (prefix[, start[, end]])	Return True if S starts with the specified prefix, False otherwise.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase</i> (/)	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>title</i> (/)	Return a version of the string where each word is titlecased.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper</i> (/)	Return a copy of the string converted to uppercase.
<i>zfill</i> (width, /)	Pad a numeric string with zeros on the left, to fill a field of the given width.

class libera_utils.aws.constants.LiberaApid(*value*)

APIIDs for L0 packets

class libera_utils.aws.constants.LiberaDataBucketName(*value*)

Names of the data archive buckets

Methods

<i>capitalize</i> (/)	Return a capitalized version of the string.
<i>casefold</i> (/)	Return a version of the string suitable for caseless comparisons.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>encode</i> (/[, encoding, errors])	Encode the string using the codec registered for encoding.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>expandtabs</i> (/[, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>format</i> (*args, **kwargs)	Return a formatted version of S, using substitutions from args and kwargs.
<i>format_map</i> (mapping)	Return a formatted version of S, using substitutions from mapping.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>isalnum</i> (/)	Return True if the string is an alpha-numeric string, False otherwise.
<i>isalpha</i> (/)	Return True if the string is an alphabetic string, False otherwise.
<i>isascii</i> (/)	Return True if all characters in the string are ASCII, False otherwise.
<i>isdecimal</i> (/)	Return True if the string is a decimal string, False otherwise.

continues on next page

Table 29 – continued from previous page

<i>isdigit()</i>	Return True if the string is a digit string, False otherwise.
<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.
<i>join(iterable, /)</i>	Concatenate any number of strings.
<i>ljust(width[, fillchar])</i>	Return a left-justified string of length width.
<i>lower()</i>	Return a copy of the string converted to lowercase.
<i>lstrip([chars])</i>	Return a copy of the string with leading whitespace removed.
<i>maketrans(x[, y, z])</i>	Return a translation table usable for str.translate().
<i>partition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>removeprefix(prefix, /)</i>	Return a str with the given prefix string removed if present.
<i>removesuffix(suffix, /)</i>	Return a str with the given suffix string removed if present.
<i>replace(old, new[, count])</i>	Return a copy with all occurrences of substring old replaced by new.
<i>rfind(sub[, start[, end]])</i>	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex(sub[, start[, end]])</i>	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust(width[, fillchar])</i>	Return a right-justified string of length width.
<i>rpartition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>rsplit(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using sep as the separator string.
<i>rstrip([chars])</i>	Return a copy of the string with trailing whitespace removed.
<i>split(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using sep as the separator string.
<i>splitlines(/[, keepends])</i>	Return a list of the lines in the string, breaking at line boundaries.
<i>startswith(prefix[, start[, end]])</i>	Return True if S starts with the specified prefix, False otherwise.
<i>strip([chars])</i>	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase()</i>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>title()</i>	Return a version of the string where each word is titlecased.

continues on next page

Table 29 – continued from previous page

<code>translate(table, /)</code>	Replace each character in the string using the given translation table.
<code>upper(/)</code>	Return a copy of the string converted to uppercase.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.

class `libera_utils.aws.constants.ManifestType(value)`

Enumerated legal manifest type values

Methods

<code>capitalize(/)</code>	Return a capitalized version of the string.
<code>casefold(/)</code>	Return a version of the string suitable for caseless comparisons.
<code>center(width[, fillchar])</code>	Return a centered string of length width.
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<code>encode(/[, encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>expandtabs(/[, tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>index(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>isalnum(/)</code>	Return True if the string is an alpha-numeric string, False otherwise.
<code>isalpha(/)</code>	Return True if the string is an alphabetic string, False otherwise.
<code>isascii(/)</code>	Return True if all characters in the string are ASCII, False otherwise.
<code>isdecimal(/)</code>	Return True if the string is a decimal string, False otherwise.
<code>isdigit(/)</code>	Return True if the string is a digit string, False otherwise.
<code>isidentifier(/)</code>	Return True if the string is a valid Python identifier, False otherwise.
<code>islower(/)</code>	Return True if the string is a lowercase string, False otherwise.
<code>isnumeric(/)</code>	Return True if the string is a numeric string, False otherwise.
<code>isprintable(/)</code>	Return True if the string is printable, False otherwise.
<code>isspace(/)</code>	Return True if the string is a whitespace string, False otherwise.

continues on next page

Table 30 – continued from previous page

<code>istitle()</code>	Return True if the string is a title-cased string, False otherwise.
<code>isupper()</code>	Return True if the string is an uppercase string, False otherwise.
<code>join(iterable, /)</code>	Concatenate any number of strings.
<code>ljust(width[, fillchar])</code>	Return a left-justified string of length width.
<code>lower()</code>	Return a copy of the string converted to lowercase.
<code>lstrip([chars])</code>	Return a copy of the string with leading whitespace removed.
<code>maketrans(x[, y, z])</code>	Return a translation table usable for <code>str.translate()</code> .
<code>partition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>removeprefix(prefix, /)</code>	Return a str with the given prefix string removed if present.
<code>removesuffix(suffix, /)</code>	Return a str with the given suffix string removed if present.
<code>replace(old, new[, count])</code>	Return a copy with all occurrences of substring <code>old</code> replaced by <code>new</code> .
<code>rfind(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rindex(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rjust(width[, fillchar])</code>	Return a right-justified string of length width.
<code>rpartition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>rsplit(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>rstrip([chars])</code>	Return a copy of the string with trailing whitespace removed.
<code>split(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>splitlines(/[, keepends])</code>	Return a list of the lines in the string, breaking at line boundaries.
<code>startswith(prefix[, start[, end]])</code>	Return True if <code>S</code> starts with the specified prefix, False otherwise.
<code>strip([chars])</code>	Return a copy of the string with leading and trailing whitespace removed.
<code>swapcase()</code>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<code>title()</code>	Return a version of the string where each word is titlecased.
<code>translate(table, /)</code>	Replace each character in the string using the given translation table.
<code>upper()</code>	Return a copy of the string converted to uppercase.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.

class `libera_utils.aws.constants.ProcessingStepIdentifier`(*value*)

Enumeration of processing step IDs used in AWS resource naming and processing orchestration

In orchestration code, these are used as “NodeID” values to identify processing steps:

The `processing_step_node_id` values used in `libera_cdk` deployment stackbuilder module and the node names in `processing_system_dag.json` must match these.

They must also be passed to the `ecr_upload` module called by some `libera_cdk` integration tests.

Methods

<code>capitalize()</code>	Return a capitalized version of the string.
<code>casefold()</code>	Return a version of the string suitable for caseless comparisons.
<code>center(width[, fillchar])</code>	Return a centered string of length width.
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<code>encode([, encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>expandtabs([, tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>index(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>isalnum()</code>	Return True if the string is an alpha-numeric string, False otherwise.
<code>isalpha()</code>	Return True if the string is an alphabetic string, False otherwise.
<code>isascii()</code>	Return True if all characters in the string are ASCII, False otherwise.
<code>isdecimal()</code>	Return True if the string is a decimal string, False otherwise.
<code>isdigit()</code>	Return True if the string is a digit string, False otherwise.
<code>isidentifier()</code>	Return True if the string is a valid Python identifier, False otherwise.
<code>islower()</code>	Return True if the string is a lowercase string, False otherwise.
<code>isnumeric()</code>	Return True if the string is a numeric string, False otherwise.
<code>isprintable()</code>	Return True if the string is printable, False otherwise.
<code>isspace()</code>	Return True if the string is a whitespace string, False otherwise.
<code>istitle()</code>	Return True if the string is a title-cased string, False otherwise.
<code>isupper()</code>	Return True if the string is an uppercase string, False otherwise.
<code>join(iterable, /)</code>	Concatenate any number of strings.
<code>ljust(width[, fillchar])</code>	Return a left-justified string of length width.
<code>lower()</code>	Return a copy of the string converted to lowercase.
<code>lstrip([chars])</code>	Return a copy of the string with leading whitespace removed.

continues on next page

Table 31 – continued from previous page

<code>maketrans(x[, y, z])</code>	Return a translation table usable for <code>str.translate()</code> .
<code>partition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>removeprefix(prefix, /)</code>	Return a str with the given prefix string removed if present.
<code>removesuffix(suffix, /)</code>	Return a str with the given suffix string removed if present.
<code>replace(old, new[, count])</code>	Return a copy with all occurrences of substring <code>old</code> replaced by <code>new</code> .
<code>rfind(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rindex(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rjust(width[, fillchar])</code>	Return a right-justified string of length <code>width</code> .
<code>rpartition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>rsplit(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>rstrip([chars])</code>	Return a copy of the string with trailing whitespace removed.
<code>split(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>splitlines(/[, keepends])</code>	Return a list of the lines in the string, breaking at line boundaries.
<code>startswith(prefix[, start[, end]])</code>	Return True if <code>S</code> starts with the specified prefix, False otherwise.
<code>strip([chars])</code>	Return a copy of the string with leading and trailing whitespace removed.
<code>swapcase(/)</code>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<code>title(/)</code>	Return a version of the string where each word is titlecased.
<code>translate(table, /)</code>	Replace each character in the string using the given translation table.
<code>upper(/)</code>	Return a copy of the string converted to uppercase.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.

`dump(chunk_number: int | None = None) → str`

Convert the ProcessingStepIdentifier to a string suitable for matching with a DAG key or in the processing orchestration system.

The `chunk_number` can be specified when the data product represents a PDS file that is typically provided in 12 2-hour chunks per day. In that case, the `chunk_number` appears as a suffix to the orchestration step identifier

property `ecr_name: str | None`

Get the manually-configured ECR name for this processing step

We name our ECRs in CDK because they are one of the few resources that humans will need to interact with on a regular basis.

get_archive_bucket_name (`account_suffix: LiberaAccountSuffix = LiberaAccountSuffix.STAGE`) → str | None

Gets the archive bucket name for this processing step.

Buckets are named according to the level of data they are storing and which account they are in. This is expected to be used by the L2 developers who will most commonly be working with the stage account.

Parameters

account_suffix (*LiberaAccountSuffix*, *optional*) – Account suffix for the bucket name, by default LiberaAccountSuffix.STAGE (stage account)

Returns

The name of the archive bucket for this processing step

Return type

str

classmethod validate(*processing_step: str*) → tuple[*ProcessingStepIdentifier*, int | None]

Validate a processing step string used by the DAG or the orchestration system.

If successful, returns a tuple containing the ProcessingStepIdentifier and the chunk_number, which can be None if the input string does not contain a valid chunk number.

class libera_utils.aws.constants.SpkObject(*value*)

Enum of valid SPK objects

property data_product_id: *DataProductIdentifier*

DataProductIdentifier for SPKs associated with this SPK object

property processing_step_id: *ProcessingStepIdentifier*

ProcessingStepIdentifier for the processing step that produces SPKs for this SPK object

libera_utils.aws.ecr_upload

Module for uploading docker images to the ECR

Functions

<i>build_docker_image</i> (<i>context_dir</i> , <i>image_name</i>)	Build a Docker image from a specified directory and tag it with a custom name.
<i>ecr_upload_cli_func</i> (<i>parsed_args</i>)	CLI handler function for ecr-upload CLI subcommand.
<i>get_ecr_docker_client</i> ([<i>region_name</i> , ...])	Perform programmatic docker login to the default ECR for the current AWS credential account (e.g. AWS_PROFILE) and return a DockerClient object for interacting with the ECR.
<i>push_image_to_ecr</i> (<i>image_name</i> , <i>image_tag</i> , ...)	Programmatically upload a docker image for a science algorithm to an ECR.

libera_utils.aws.ecr_upload.build_docker_image

libera_utils.aws.ecr_upload.build_docker_image(*context_dir: str | Path*, *image_name: str*, *tag: str = 'latest'*, *target: str | None = None*, *platform: str = 'linux/amd64'*) → None

Build a Docker image from a specified directory and tag it with a custom name.

Parameters

- **context_dir** (*Union[str, Path]*) – The path to the directory containing the Dockerfile and other build context.
- **image_name** (*str*) – The name to give the Docker image.
- **tag** (*str, optional*) – The tag to apply to the image (default is 'latest').
- **target** (*Optional[str]*) – Name of the target to build.
- **platform** (*str*) – Default "linux/amd64".

Raises

ValueError – If the specified directory does not exist or the build fails.

libera_utils.aws.ecr_upload.ecr_upload_cli_func

`libera_utils.aws.ecr_upload.ecr_upload_cli_func(parsed_args: Namespace) → None`

CLI handler function for ecr-upload CLI subcommand.

Parameters

parsed_args (*argparse.Namespace*) – Namespace of parsed CLI arguments

Return type

None

libera_utils.aws.ecr_upload.get_ecr_docker_client

`libera_utils.aws.ecr_upload.get_ecr_docker_client(region_name: str | None = None, dockercfg_path: Path | None = None) → DockerClient`

Perform programmatic docker login to the default ECR for the current AWS credential account (e.g. AWS_PROFILE) and return a DockerClient object for interacting with the ECR.

Parameters

- **region_name** (*Optional[str]*) – AWS region name. Each region has a separate default ECR. If region_name is None, boto3 uses the default region for the configured credentials.
- **dockercfg_path** (*Optional[Path]*) – Use a custom path for the Docker config file. (default `$HOME/.docker/config.json` if present, otherwise `$HOME/.dockercfg`)

Returns

Logged in docker client.

Return type

`docker.DockerClient`

libera_utils.aws.ecr_upload.push_image_to_ecr

`libera_utils.aws.ecr_upload.push_image_to_ecr(image_name: str, image_tag: str, algorithm_name: str | ProcessingStepIdentifier, * (Keyword-only parameters separator (PEP 3102)), ecr_image_tags: List[str] | None = None, region_name: str = 'us-west-2', ignore_docker_config: bool = False) → None`

Programmatically upload a docker image for a science algorithm to an ECR. ECR name is determined based on the algorithm name.

Parameters

- **image_name** (*str*) – Local name of the image
- **image_tag** (*str*) – Local tag of the image (often latest)

- **algorithm_name** (*Union*[*str*, *constants.ProcessingStepIdentifier*]) – Processing step ID string or object. Used to infer the ECR repository name. L0 processing step IDs are not allowed because they have no associated ECR.
- **ecr_image_tags** (*Optional*[*List*[*str*]]) – List of tags to apply to the pushed image in the ECR (e.g. [“1.3.4”, “latest”]). Default None, results in pushing only as “latest”.
- **region_name** (*str*) – AWS region. Used to infer the ECR name.
- **ignore_docker_config** (*bool*) – Default False. If True, creates a temporary docker config.json file to prevent using stored credentials.

Return type

None

Classes

<code>DockerConfigManager</code> (<code>[override_default_config]</code>)	Context manager object, suitable for use with <code>docker-py DockerClient.login</code>
---	---

libera_utils.aws.ecr_upload.DockerConfigManager

class `libera_utils.aws.ecr_upload.DockerConfigManager`(`override_default_config: bool = False`)

Bases: `object`

Context manager object, suitable for use with `docker-py DockerClient.login`

If `override_default_config` is True, `dockercfg_path` points to a temporary directory with a blank config. Otherwise, `dockercfg_path` is None, which allows `DockerClient.login` to use the default config location.

`__init__`(`override_default_config: bool = False`)

class `libera_utils.aws.ecr_upload.DockerConfigManager`(`override_default_config: bool = False`)

Context manager object, suitable for use with `docker-py DockerClient.login`

If `override_default_config` is True, `dockercfg_path` points to a temporary directory with a blank config. Otherwise, `dockercfg_path` is None, which allows `DockerClient.login` to use the default config location.

`libera_utils.aws.ecr_upload.build_docker_image`(`context_dir: str | Path`, `image_name: str`, `tag: str = 'latest'`, `target: str | None = None`, `platform: str = 'linux/amd64'`) → None

Build a Docker image from a specified directory and tag it with a custom name.

Parameters

- **context_dir** (*Union*[*str*, *Path*]) – The path to the directory containing the Dockerfile and other build context.
- **image_name** (*str*) – The name to give the Docker image.
- **tag** (*str*, *optional*) – The tag to apply to the image (default is ‘latest’).
- **target** (*Optional*[*str*]) – Name of the target to build.
- **platform** (*str*) – Default “linux/amd64”.

Raises

ValueError – If the specified directory does not exist or the build fails.

`libera_utils.aws.ecr_upload.ecr_upload_cli_func(parsed_args: Namespace) → None`

CLI handler function for ecr-upload CLI subcommand.

Parameters

`parsed_args` (*argparse.Namespace*) – Namespace of parsed CLI arguments

Return type

None

`libera_utils.aws.ecr_upload.get_ecr_docker_client(region_name: str | None = None, dockercfg_path: Path | None = None) → DockerClient`

Perform programmatic docker login to the default ECR for the current AWS credential account (e.g. AWS_PROFILE) and return a DockerClient object for interacting with the ECR.

Parameters

- **region_name** (*Optional[str]*) – AWS region name. Each region has a separate default ECR. If region_name is None, boto3 uses the default region for the configured credentials.
- **dockercfg_path** (*Optional[Path]*) – Use a custom path for the Docker config file. (default `$HOME/.docker/config.json` if present, otherwise `$HOME/.dockercfg`)

Returns

Logged in docker client.

Return type

`docker.DockerClient`

`libera_utils.aws.ecr_upload.push_image_to_ecr(image_name: str, image_tag: str, algorithm_name: str | ProcessingStepIdentifier, *, ecr_image_tags: List[str] | None = None, region_name: str = 'us-west-2', ignore_docker_config: bool = False) → None`

Programmatically upload a docker image for a science algorithm to an ECR. ECR name is determined based on the algorithm name.

Parameters

- **image_name** (*str*) – Local name of the image
- **image_tag** (*str*) – Local tag of the image (often latest)
- **algorithm_name** (*Union[str, constants.ProcessingStepIdentifier]*) – Processing step ID string or object. Used to infer the ECR repository name. L0 processing step IDs are not allowed because they have no associated ECR.
- **ecr_image_tags** (*Optional[List[str]]*) – List of tags to apply to the pushed image in the ECR (e.g. ["1.3.4", "latest"]). Default None, results in pushing only as "latest".
- **region_name** (*str*) – AWS region. Used to infer the ECR name.
- **ignore_docker_config** (*bool*) – Default False. If True, creates a temporary docker config.json file to prevent using stored credentials.

Return type

None

`libera_utils.aws.processing_step_function_trigger`

Module for manually triggering a step function

Functions

<code>step_function_trigger(parsed_args)</code>	Start a stepfunction to process a certain days data :param parsed_args: Namespace of parsed CLI arguments :type parsed_args: argparse.Namespace :param region_name: string of the AWS region name :type region_name: str
---	--

libera_utils.aws.processing_step_function_trigger.step_function_trigger

`libera_utils.aws.processing_step_function_trigger.step_function_trigger(parsed_args: Namespace)`

Start a stepfunction to process a certain days data :param parsed_args: Namespace of parsed CLI arguments :type parsed_args: argparse.Namespace :param region_name: string of the AWS region name :type region_name: str

Return type

None

`libera_utils.aws.processing_step_function_trigger.step_function_trigger(parsed_args: Namespace)`

Start a stepfunction to process a certain days data :param parsed_args: Namespace of parsed CLI arguments :type parsed_args: argparse.Namespace :param region_name: string of the AWS region name :type region_name: str

Return type

None

libera_utils.aws.s3_utilities

Module for S3 cli utilities

Functions

<code>s3_copy_cli_handler(parsed_args)</code>	CLI handler function for s3-utils cp CLI subcommand.
<code>s3_list_archive_files(processing_step, *, ...)</code>	List all files in an archive S3 bucket for a given processing step.
<code>s3_list_cli_handler(parsed_args)</code>	CLI handler function for s3-utils list CLI subcommand.
<code>s3_put_cli_handler(parsed_args)</code>	CLI handler function for s3-utils put CLI subcommand.
<code>s3_put_in_archive_for_processing_step(..., ...)</code>	Upload a file to the archive S3 bucket associated with a given processing step.

libera_utils.aws.s3_utilities.s3_copy_cli_handler

`libera_utils.aws.s3_utilities.s3_copy_cli_handler(parsed_args: Namespace) → None`

CLI handler function for s3-utils cp CLI subcommand.

libera_utils.aws.s3_utilities.s3_list_archive_files

`libera_utils.aws.s3_utilities.s3_list_archive_files(processing_step: str | ProcessingStepIdentifier, *, account_suffix: str | LiberaAccountSuffix | None = LiberaAccountSuffix.STAGE, print_out: bool | None = False) → list`

List all files in an archive S3 bucket for a given processing step.

Parameters

- **processing_step** (*str*) – Processing step ID string. Used to infer the S3 archive bucket name.
- **account_suffix** (*Union[str, constants.LiberaAccountSuffix]*, *optional*) – Account suffix for the bucket name, by default `constants.LiberaAccountSuffix.STAGE`
- **print_out** (*bool*, *optional*) – Print the list of files to the console, by default `False`

Returns

bucket_objects – S3Path objects for each file in the bucket

Return type

`list`

libera_utils.aws.s3_utilities.s3_list_cli_handler

`libera_utils.aws.s3_utilities.s3_list_cli_handler(parsed_args: Namespace) → None`

CLI handler function for s3-utils list CLI subcommand.

libera_utils.aws.s3_utilities.s3_put_cli_handler

`libera_utils.aws.s3_utilities.s3_put_cli_handler(parsed_args: Namespace) → None`

CLI handler function for s3-utils put CLI subcommand.

libera_utils.aws.s3_utilities.s3_put_in_archive_for_processing_step

`libera_utils.aws.s3_utilities.s3_put_in_archive_for_processing_step(path_to_file: Path | S3Path, processing_step: str | ProcessingStepIdentifier, *, account_suffix: str | LiberaAccountSuffix | None = LiberaAccountSuffix.STAGE)`

Upload a file to the archive S3 bucket associated with a given processing step.

Parameters

- **path_to_file** (*Path*) – Local path to the file to upload
- **processing_step** (*str*) – `processing_step: Union[str, constants.ProcessingStepIdentifier]` Processing step ID string or object. Used to infer the S3 archive bucket name.
- **account_suffix** (*Union[str, constants.LiberaAccountSuffix]*, *optional*) – Account suffix for the bucket name, by default `constants.LiberaAccountSuffix.STAGE`

`libera_utils.aws.s3_utilities.s3_copy_cli_handler(parsed_args: Namespace) → None`

CLI handler function for s3-utils cp CLI subcommand.

`libera_utils.aws.s3_utilities.s3_list_archive_files(processing_step: str | ProcessingStepIdentifier, *, account_suffix: str | LiberaAccountSuffix | None = LiberaAccountSuffix.STAGE, print_out: bool | None = False) → list`

List all files in an archive S3 bucket for a given processing step.

Parameters

- **processing_step** (*str*) – Processing step ID string. Used to infer the S3 archive bucket name.

- **account_suffix** (*Union[str, constants.LiberaAccountSuffix]*, *optional*) – Account suffix for the bucket name, by default constants.LiberaAccountSuffix.STAGE
- **print_out** (*bool*, *optional*) – Print the list of files to the console, by default False

Returns

bucket_objects – S3Path objects for each file in the bucket

Return type

list

`libera_utils.aws.s3_utilities.s3_list_cli_handler(parsed_args: Namespace) → None`

CLI handler function for s3-utils list CLI subcommand.

`libera_utils.aws.s3_utilities.s3_put_cli_handler(parsed_args: Namespace) → None`

CLI handler function for s3-utils put CLI subcommand.

`libera_utils.aws.s3_utilities.s3_put_in_archive_for_processing_step(path_to_file: Path | S3Path, processing_step: str | ProcessingStepIdentifier, *, account_suffix: str | LiberaAccountSuffix | None = LiberaAccountSuffix.STAGE)`

Upload a file to the archive S3 bucket associated with a given processing step.

Parameters

- **path_to_file** (*Path*) – Local path to the file to upload
- **processing_step** (*str*) – *processing_step*: Union[str, constants.ProcessingStepIdentifier] Processing step ID string or object. Used to infer the S3 archive bucket name.
- **account_suffix** (*Union[str, constants.LiberaAccountSuffix]*, *optional*) – Account suffix for the bucket name, by default constants.LiberaAccountSuffix.STAGE

libera_utils.aws.utils

Helper functions for AWS access

Functions

<code>get_aws_account_number([region_name])</code>	Get a users AWS account ID number :param region_name: Region that the users AWS account is on :type region_name: string
--	---

libera_utils.aws.utils.get_aws_account_number

`libera_utils.aws.utils.get_aws_account_number(region_name='us-west-2')`

Get a users AWS account ID number :param region_name: Region that the users AWS account is on :type region_name: string

Returns

account_id – users account_id number

Return type

int

`libera_utils.aws.utils.get_aws_account_number(region_name='us-west-2')`

Get a users AWS account ID number :param region_name: Region that the users AWS account is on :type region_name: string

Returns

account_id – users account_id number

Return type

int

3.1.2 libera_utils.backports

Module for python code that is included in later python releases but not in earlier ones

Modules

`enum_3_11`

Backported enum module from Python 3.11.1 This code is directly pulled from the Python 3.11.1 source code, located on GitHub at <https://github.com/python/cpython/tree/v3.11.1> Once 3.11 is the oldest supported Python version (~Oct 2026), this can probably be removed.

libera_utils.backports.enum_3_11

Backported enum module from Python 3.11.1 This code is directly pulled from the Python 3.11.1 source code, located on GitHub at <https://github.com/python/cpython/tree/v3.11.1> Once 3.11 is the oldest supported Python version (~Oct 2026), this can probably be removed.

Supports advanced quality flagging functionality from recent Python development.

3.1.3 libera_utils.cli

Module for the Libera SDC utilities CLI

Functions

<code>main([cli_args])</code>	Main CLI entrypoint that runs the function inferred from the specified subcommand
<code>parse_cli_args(cli_args)</code>	Parse CLI arguments
<code>print_version_info(*args)</code>	Print CLI version information

libera_utils.cli.main

`libera_utils.cli.main(cli_args: list = None)`

Main CLI entrypoint that runs the function inferred from the specified subcommand

libera_utils.cli.parse_cli_args

`libera_utils.cli.parse_cli_args(cli_args: list)`

Parse CLI arguments

Parameters

cli_args (*list*) – List of CLI arguments to parse

Returns

Parsed arguments in a Namespace object

Return type

`argparse.Namespace`

libera_utils.cli.print_version_info

`libera_utils.cli.print_version_info(*args)`

Print CLI version information

`libera_utils.cli.main(cli_args: list = None)`

Main CLI entrypoint that runs the function inferred from the specified subcommand

`libera_utils.cli.parse_cli_args(cli_args: list)`

Parse CLI arguments

Parameters

cli_args (*list*) – List of CLI arguments to parse

Returns

Parsed arguments in a Namespace object

Return type

`argparse.Namespace`

`libera_utils.cli.print_version_info(*args)`

Print CLI version information

3.1.4 libera_utils.config

Configuration reader. To modify the configuration, see file: `config.json`

Module Attributes

<code>config</code>	Singleton (one per process) accessor for <code>libera_utils.config._ConfigurationCache()</code>
---------------------	---

libera_utils.config.config

`libera_utils.config.config = <libera_utils.config._ConfigurationCache object>`

Singleton (one per process) accessor for `libera_utils.config._ConfigurationCache()`

Classes

<code>ConfigurationFormatter()</code>	Customize the string formatter to replace fields in a config string with values from the configuration dictionary.
---------------------------------------	--

libera_utils.config.ConfigurationFormatter

class `libera_utils.config.ConfigurationFormatter`

Bases: `Formatter`

Customize the string formatter to replace fields in a config string with values from the configuration dictionary. This will allow configuration parameters in the `emus_config.json` file to be based off of other configuration parameters by wrapping the configuration key in curly braces.

Methods

<code>get_field(field_name, args, kwargs)</code>	
<code>get_value(key, *args, **kwargs)</code>	Overrides the default <code>get_value</code> method in the python formatter.
<code>parse(format_string)</code>	

<code>check_unused_args</code>
<code>convert_field</code>
<code>format</code>
<code>format_field</code>
<code>vformat</code>

`__init__(*args, **kwargs)`

Methods

<code>get_value(key, *args, **kwargs)</code>	Overrides the default <code>get_value</code> method in the python formatter.
--	--

get_value(*key*: *str*, *args, **kwargs)

Overrides the default `get_value` method in the python formatter. This will return the value from the emus configuration with the specified key.

class libera_utils.config.ConfigurationFormatter

Customize the string formatter to replace fields in a config string with values from the configuration dictionary. This will allow configuration parameters in the `emus_config.json` file to be based off of other configuration parameters by wrapping the configuration key in curly braces.

Methods

<code>get_field(field_name, args, kwargs)</code>	
<code>get_value(key, *args, **kwargs)</code>	Overrides the default <code>get_value</code> method in the python formatter.
<code>parse(format_string)</code>	

<code>check_unused_args</code>
<code>convert_field</code>
<code>format</code>
<code>format_field</code>
<code>vformat</code>

get_value(*key: str, *args, **kwargs*)

Overrides the default get_value method in the python formatter. This will return the value from the emus configuration with the specified key.

class libera_utils.config._ConfigurationCache

Class that stores the JSON configuration and provides methods for accessing configuration information

Methods

<code>force_reload()</code>	Force reloading of the JSON config
<code>get(key)</code>	Retrieves a configuration value from either the cached JSON or from the environment

_format_return_value(*value: str*)

Recursively formats the returned value, looking for config keys to substitute.

Parameters

value (*str*) – String to format

Return type

str

_parse_numeric_types(*value: str*)

Checks the final result of a config retrieval. If it is a string that can be interpreted as a float or int, parse it and return that.

Parameters

value (*any*) – Final formatted value.

Return type

str or *float* or *int*

force_reload()

Force reloading of the JSON config

get(*key*)

Retrieves a configuration value from either the cached JSON or from the environment

Parameters

key (*str*) – Key for which to retrieve the configured value.

Returns

Resulting value

Return type

any

`libera_utils.config.config = <libera_utils.config._ConfigurationCache object>`

Singleton (one per process) accessor for `libera_utils.config._ConfigurationCache()`

3.1.5 libera_utils.db

db module for dyanmodb operations.

Modules

<i>dynamodb_utils</i>	Module for database utilities
-----------------------	-------------------------------

libera_utils.db.dynamodb_utils

Module for database utilities

Functions

<i>add_archive_time_to_ddb_item</i> (ddb_item)	Add archive time to DynamoDB item
<i>create_ddb_metadata_applicable_date_item</i> (*, ...)	Write metadata record to DynamoDB for a single file
<i>create_ddb_metadata_file_item</i> (filename, ...)	Write metadata record to DynamoDB for a single file
<i>get_dynamodb_table</i> (dynamo_table_name)	Get the DynamoDB table

libera_utils.db.dynamodb_utils.add_archive_time_to_ddb_item

`libera_utils.db.dynamodb_utils.add_archive_time_to_ddb_item(ddb_item: dict)`

Add archive time to DynamoDB item

libera_utils.db.dynamodb_utils.create_ddb_metadata_applicable_date_item

`libera_utils.db.dynamodb_utils.create_ddb_metadata_applicable_date_item(*, filename: str, data_level: str, data_type: str, applicable_date: str, data_subtype: str = None, additional_metadata: dict = None)`

Write metadata record to DynamoDB for a single file

libera_utils.db.dynamodb_utils.create_ddb_metadata_file_item

`libera_utils.db.dynamodb_utils.create_ddb_metadata_file_item(filename: str, algorithm_version: str, include_archive_time: bool = False, additional_metadata: dict = None)`

Write metadata record to DynamoDB for a single file

libera_utils.db.dynamodb_utils.get_dynamodb_table

`libera_utils.db.dynamodb_utils.get_dynamodb_table(dynamo_table_name: str)`

Get the DynamoDB table

`libera_utils.db.dynamodb_utils.add_archive_time_to_ddb_item(ddb_item: dict)`

Add archive time to DynamoDB item

```
libera_utils.db.dynamodb_utils.create_ddb_metadata_applicable_date_item(*, filename: str,
                                                                    data_level: str,
                                                                    data_type: str,
                                                                    applicable_date: str,
                                                                    data_subtype: str =
                                                                    None,
                                                                    additional_metadata:
                                                                    dict = None)
```

Write metadata record to DynamoDB for a single file

```
libera_utils.db.dynamodb_utils.create_ddb_metadata_file_item(filename: str, algorithm_version: str,
                                                            include_archive_time: bool = False,
                                                            additional_metadata: dict = None)
```

Write metadata record to DynamoDB for a single file

```
libera_utils.db.dynamodb_utils.get_dynamodb_table(dynamo_table_name: str)
```

Get the DynamoDB table

3.1.6 libera_utils.geolocation

Module for performing geolocation tasks

Functions

<code>angle_between(v1, v2[, degrees])</code>	Returns angle between vectors <i>v1</i> and <i>v2</i> , in units of radians (default) or degrees.
<code>cartesian_to_planetographic(cartesian_coords)</code>	Convert cartesian coordinates in the ITRF93 frame to planetographic latitude and longitude.
<code>frame_transform(from_frame, to_frame, et, ...)</code>	Transform a position <x, y, z> vector between reference frames, optionally normalizing the result.
<code>get_earth_radii()</code>	Retrieve Earth radii values from SPICE
<code>sub_observer_point(target, et, frame, ...[, ...])</code>	Computes the cartesian coordinates of the sub-observer point at time <i>et</i> and the observer altitude above the point.
<code>sub_solar_point(target, et, frame, observer, *)</code>	Computes the cartesian coordinates of the subsolar point at ephemeris time <i>et</i> .
<code>surface_intercept_point(sc_location, ...[, et])</code>	Returns rectangular coordinates of the point of interception of a look direction from the spacecraft onto the Earth ellipsoid.
<code>target_position(target, et, frame, observer, *)</code>	Calculates the position and velocity of the <i>target</i> at ephemeris time <i>et</i> relative to <i>observer</i> in reference frame <i>frame</i> .

libera_utils.geolocation.angle_between

```
libera_utils.geolocation.angle_between(v1: ndarray, v2: ndarray, degrees: bool = False)
```

Returns angle between vectors *v1* and *v2*, in units of radians (default) or degrees. *N* is the number of vectors *D* is the dimension of the space

Parameters

- **v1** (*numpy.ndarray*) – Vector(s) 1. May be shape (D,) or (N, D).
- **v2** (*numpy.ndarray*) – Vector(s) 2. May be shape (D,) or (N, D).
- **degrees** (*bool*) – Specify True to return result in degrees. Default is False (returns radians).

Returns

Angle between v1 and v2 in radians (optionally in degrees)

Return type

float or `numpy.ndarray`

libera_utils.geolocation.cartesian_to_planetographic

`libera_utils.geolocation.cartesian_to_planetographic`(*cartesian_coords*: `ndarray`, *degrees*: `bool` = `True`)

Convert cartesian coordinates in the ITRF93 frame to planetographic latitude and longitude. Longitude runs 0-360 such that longitude appears to increase as the planet rotates when viewed by an observer and latitude is calculated from a surface normal vector rather than a line through the planet center. See https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/17_frames_and_coordinate_systems.pdf for reference.

Parameters

- **cartesian_coords** (`numpy.ndarray`) – Rectangular coordinates in ITRF93 frame.
- **degrees** (`bool`) – Default true. If False, returns angles in radians.

Returns

Each coordinate is returned as (longitude, latitude, altitude).

Return type

`numpy.ndarray`

libera_utils.geolocation.frame_transform

`libera_utils.geolocation.frame_transform`(*from_frame*: `SpiceFrame`, *to_frame*: `SpiceFrame`, *et*: `float`, *position*: `ndarray`, *normalize*: `bool` = `False`) → `ndarray`

Transform a position <x, y, z> vector between reference frames, optionally normalizing the result.

Parameters

- **from_frame** (`spice_utils.SpiceFrame`) – Reference frame of position
- **to_frame** (`spice_utils.SpiceFrame`) – Reference frame of output
- **et** (`numpy.float64` or `numpy.ndarray`) – Ephemeris time(s) corresponding to position(s). For time-independent transformations, this can be any valid ephemeris time.
- **position** (`numpy.ndarray`) – <x, y, z> vector or array of vectors in reference frame *from_frame*
- **normalize** (`bool`, *Optional*) – Optionally normalize the output vector

Returns

3d position vector(s) in reference frame *to_frame*

Return type

`numpy.ndarray`

libera_utils.geolocation.get_earth_radii

`libera_utils.geolocation.get_earth_radii`()

Retrieve Earth radii values from SPICE

Returns

(re, rp, flat) tuple of equatorial and polar ellipsoid radii and a flattening coefficient

Return type

tuple

libera_utils.geolocation.sub_observer_point

`libera_utils.geolocation.sub_observer_point`(*target*: `SpiceBody`, *et*: `float`, *frame*: `SpiceFrame`, *observer*: `SpiceBody`, *, *abcorr*: `str = 'NONE'`, *method*: `str = 'NEAR POINT/ELLIPSOID'`)

Computes the cartesian coordinates of the sub-observer point at time *et* and the observer altitude above the point. Units in km.

Parameters

- **target** (`spice_utils.SpiceBody`) – Body on which the sub point will be calculated (usually a planetary body).
- **et** (`float` or `numpy.ndarray`) – Ephemeris time of observation.
- **frame** (`spice_utils.SpiceFrame`) – Reference frame for returned vectors.
- **observer** (`spice_utils.SpiceBody`) – The object from which to calculate the sub point (e.g. a spacecraft). A-B correction is applied based on the distance between observer and sub observer point.
- **abcorr** (`str`, *Optional*) – String delineating what kind of A-B light time correction to perform. Default is 'NONE'.
- **method** (`str`, *Optional*) – String specifying what kind of method to use to find the vector between the observer and the target. Default is `NEAR POINT/ELLIPSOID`, which uses the nearest point on the ellipsoid rather than drawing a line through the center of the ellipsoid.

Returns

Cartesian point on the target body surface in the specified reference frame and also the euclidean distance between the observer and the sub point in order: [x, y, z], `obs_alt`

Return type`numpy.ndarray`, `float`**libera_utils.geolocation.sub_solar_point**

`libera_utils.geolocation.sub_solar_point`(*target*: `SpiceBody`, *et*: `float`, *frame*: `SpiceFrame`, *observer*: `SpiceBody`, *, *abcorr*=`'LT+S'`, *method*=`'NEAR POINT/ELLIPSOID'`)

Computes the cartesian coordinates of the subsolar point at ephemeris time *et*.

Parameters

- **target** (`spice_utils.SpiceBody`) – Body on which the sub point will be calculated (usually a planetary body).
- **et** (`float` or `numpy.ndarray`) – Ephemeris time of observation.
- **frame** (`spice_utils.SpiceFrame`) – Reference frame for returned vectors.
- **observer** (`spice_utils.SpiceBody`) – The object from which to calculate the subsolar point (e.g. a spacecraft). A-B correction is applied based on the distance between observer and subsolar point.
- **abcorr** (`str`) – String delineating what kind of A-B lighttime correction to perform. Default is 'LT+S'.

- **method** (*str*) – String specifying what kind of method to use to find the vector between the observer and the target. Default is *NEAR POINT/ELLIPSOID*, which uses the nearest point on the ellipsoid rather than drawing a line through the center of the ellipsoid.

Returns

Subsolar point on the ellipsoid surface in the specified reference frame, apparent epoch at that point (depending on specified light time correction), and vector from observer to subsolar point.

Return type

`numpy.ndarray, numpy.ndarray, numpy.ndarray`

libera_utils.geolocation.surface_intercept_point

`libera_utils.geolocation.surface_intercept_point`(*sc_location: ndarray, look_vector: ndarray, look_frame: SpiceFrame, et: float = None*)

Returns rectangular coordinates of the point of interception of a look direction from the spacecraft onto the Earth ellipsoid. If the look vector misses the planet, then the distance returned will be non-zero and the point returned is the point on the look_vector ray that is closest to the ellipsoid.

This routine assumes that the location of the spacecraft and the location of the instrument are the same because we don't have ephemeris data for the instrument but we *_do_* have ephemeris for the spacecraft. Over the scale of distances involved, the offset between spacecraft and instrument (meters) should be negligible in affecting the near-point calculation.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/npedln_c.html

Parameters

- **sc_location** (*numpy.ndarray*) – The location of the observing body (i.e. the spacecraft body) with respect to Earth
- **look_vector** (*numpy.ndarray*) – Look direction unit vector (e.g. an instrument look direction)
- **look_frame** (*spice_utils.SpiceFrame*) – Reference frame of *look_vector*
- **et** (*float or numpy.ndarray or None, Optional*) – Ephemeris time (at spacecraft at photon detection time). Only required if *look_frame* is not ITRF93.

Returns

(*pnear, alt*) Rectangular coordinates of nearest point to reference surface ellipsoid and distance between the line and the near point.

Return type

`tuple`

libera_utils.geolocation.target_position

`libera_utils.geolocation.target_position`(*target: SpiceBody, et: float, frame: SpiceFrame, observer: SpiceBody, *, abcorr: str = 'NONE', normalize: bool = False*)

Calculates the position and velocity of the *target* at ephemeris time *et* relative to *observer* in reference frame *frame*. Also calculates the light travel time between *target* and *observer* at time *et*.

Parameters

- **target** (*spice_utils.SpiceBody*) – Target body for which to calculate position and velocity relative to observer
- **et** (*float or numpy.ndarray*) – Ephemeris time(s)
- **frame** (*spice_utils.SpiceFrame*) – Reference frame (unit vectors)

- **observer** (`spice_utils.SpiceBody`) – The observer of the target. Resulting coordinates point from observer to target.
- **abcorr** (`str`) – A scalar string that indicates the aberration corrections to apply to the database of the target body to account for one-way light time and stellar aberration. Default is 'NONE'.
- **normalize** (`bool`, *Optional*) – Return unit vectors for position and velocity (light time output is unchanged)

Returns

(`x`: `numpy.ndarray`, `v`: `numpy.ndarray`, `lt`: `numpy.ndarray`) or (`x`: `float`, `v`: `float`, `lt`: `float`) Rectangular position and velocity vectors (`x`, `y`, `z`), (`v_x`, `v_y`, `v_z`) where position points from the planet center of mass location at `et` to the aberration-corrected location of the target. Light time (`lt`) between planetary body and target.

Return type

`tuple`

`libera_utils.geolocation.angle_between`(`v1`: `ndarray`, `v2`: `ndarray`, `degrees`: `bool` = `False`)

Returns angle between vectors `v1` and `v2`, in units of radians (default) or degrees. `N` is the number of vectors `D` is the dimension of the space

Parameters

- **v1** (`numpy.ndarray`) – Vector(s) 1. May be shape (`D`,) or (`N`, `D`).
- **v2** (`numpy.ndarray`) – Vector(s) 2. May be shape (`D`,) or (`N`, `D`).
- **degrees** (`bool`) – Specify `True` to return result in degrees. Default is `False` (returns radians).

Returns

Angle between `v1` and `v2` in radians (optionally in degrees)

Return type

`float` or `numpy.ndarray`

`libera_utils.geolocation.cartesian_to_planetographic`(`cartesian_coords`: `ndarray`, `degrees`: `bool` = `True`)

Convert cartesian coordinates in the ITRF93 frame to planetographic latitude and longitude. Longitude runs 0-360 such that longitude appears to increase as the planet rotates when viewed by an observer and latitude is calculated from a surface normal vector rather than a line through the planet center. See https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/Tutorials/pdf/individual_docs/17_frames_and_coordinate_systems.pdf for reference.

Parameters

- **cartesian_coords** (`numpy.ndarray`) – Rectangular coordinates in ITRF93 frame.
- **degrees** (`bool`) – Default `true`. If `False`, returns angles in radians.

Returns

Each coordinate is returned as (longitude, latitude, altitude).

Return type

`numpy.ndarray`

`libera_utils.geolocation.frame_transform`(`from_frame`: `SpiceFrame`, `to_frame`: `SpiceFrame`, `et`: `float`, `position`: `ndarray`, `normalize`: `bool` = `False`) → `ndarray`

Transform a position `<x, y, z>` vector between reference frames, optionally normalizing the result.

Parameters

- **from_frame** (`spice_utils.SpiceFrame`) – Reference frame of position

- **to_frame** (`spice_utils.SpiceFrame`) – Reference frame of output
- **et** (`numpy.float64` or `numpy.ndarray`) – Ephemeris time(s) corresponding to position(s). For time-independent transformations, this can be any valid ephemeris time.
- **position** (`numpy.ndarray`) – $\langle x, y, z \rangle$ vector or array of vectors in reference frame *from_frame*
- **normalize** (`bool`, *Optional*) – Optionally normalize the output vector

Returns

3d position vector(s) in reference frame *to_frame*

Return type

`numpy.ndarray`

`libera_utils.geolocation.get_earth_radii()`

Retrieve Earth radii values from SPICE

Returns

(*re*, *rp*, *flat*) tuple of equatorial and polar ellipsoid radii and a flattening coefficient

Return type

tuple

`libera_utils.geolocation.sub_observer_point(target: SpiceBody, et: float, frame: SpiceFrame, observer: SpiceBody, *, abcorr: str = 'NONE', method: str = 'NEAR POINT/ELLIPSOID')`

Computes the cartesian coordinates of the sub-observer point at time *et* and the observer altitude above the point. Units in km.

Parameters

- **target** (`spice_utils.SpiceBody`) – Body on which the sub point will be calculated (usually a planetary body).
- **et** (`float` or `numpy.ndarray`) – Ephemeris time of observation.
- **frame** (`spice_utils.SpiceFrame`) – Reference frame for returned vectors.
- **observer** (`spice_utils.SpiceBody`) – The object from which to calculate the sub point (e.g. a spacecraft). A-B correction is applied based on the distance between observer and sub observer point.
- **abcorr** (`str`, *Optional*) – String delineating what kind of A-B light time correction to perform. Default is 'NONE'.
- **method** (`str`, *Optional*) – String specifying what kind of method to use to find the vector between the observer and the target. Default is *NEAR POINT/ELLIPSOID*, which uses the nearest point on the ellipsoid rather than drawing a line through the center of the ellipsoid.

Returns

Cartesian point on the target body surface in the specified reference frame and also the euclidean distance between the observer and the sub point in order: [x, y, z], *obs_alt*

Return type

`numpy.ndarray`, `float`

`libera_utils.geolocation.sub_solar_point(target: SpiceBody, et: float, frame: SpiceFrame, observer: SpiceBody, *, abcorr='LT+S', method='NEAR POINT/ELLIPSOID')`

Computes the cartesian coordinates of the subsolar point at ephemeris time *et*.

Parameters

- **target** (`spice_utils.SpiceBody`) – Body on which the sub point will be calculated (usually a planetary body).
- **et** (`float` or `numpy.ndarray`) – Ephemeris time of observation.
- **frame** (`spice_utils.SpiceFrame`) – Reference frame for returned vectors.
- **observer** (`spice_utils.SpiceBody`) – The object from which to calculate the subsolar point (e.g. a spacecraft). A-B correction is applied based on the distance between observer and subsolar point.
- **abcorr** (`str`) – String delineating what kind of A-B lighttime correction to perform. Default is 'LT+S'.
- **method** (`str`) – String specifying what kind of method to use to find the vector between the observer and the target. Default is *NEAR POINT/ELLIPSOID*, which uses the nearest point on the ellipsoid rather than drawing a line through the center of the ellipsoid.

Returns

Subsolar point on the ellipsoid surface in the specified reference frame, apparent epoch at that point (depending on specified light time correction), and vector from observer to subsolar point.

Return type

`numpy.ndarray`, `numpy.ndarray`, `numpy.ndarray`

`libera_utils.geolocation.surface_intercept_point`(*sc_location: ndarray, look_vector: ndarray, look_frame: SpiceFrame, et: float = None*)

Returns rectangular coordinates of the point of interception of a look direction from the spacecraft onto the Earth ellipsoid. If the look vector misses the planet, then the distance returned will be non-zero and the point returned is the point on the look_vector ray that is closest to the ellipsoid.

This routine assumes that the location of the spacecraft and the location of the instrument are the same because we don't have ephemeris data for the instrument but we *do* have ephemeris for the spacecraft. Over the scale of distances involved, the offset between spacecraft and instrument (meters) should be negligible in affecting the near-point calculation.

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/npedln_c.html

Parameters

- **sc_location** (`numpy.ndarray`) – The location of the observing body (i.e. the spacecraft body) with respect to Earth
- **look_vector** (`numpy.ndarray`) – Look direction unit vector (e.g. an instrument look direction)
- **look_frame** (`spice_utils.SpiceFrame`) – Reference frame of *look_vector*
- **et** (`float` or `numpy.ndarray` or `None`, *Optional*) – Ephemeris time (at spacecraft at photon detection time). Only required if *look_frame* is not ITRF93.

Returns

(*pnear*, *alt*) Rectangular coordinates of nearest point to reference surface ellipsoid and distance between the line and the near point.

Return type

`tuple`

`libera_utils.geolocation.target_position`(*target: SpiceBody, et: float, frame: SpiceFrame, observer: SpiceBody, *, abcorr: str = 'NONE', normalize: bool = False*)

Calculates the position and velocity of the *target* at ephemeris time *et* relative to *observer* in reference frame *frame*. Also calculates the light travel time between *target* and *observer* at time *et*.

Parameters

- **target** (`spice_utils.SpiceBody`) – Target body for which to calculate position and velocity relative to observer
- **et** (`float` or `numpy.ndarray`) – Ephemeris time(s)
- **frame** (`spice_utils.SpiceFrame`) – Reference frame (unit vectors)
- **observer** (`spice_utils.SpiceBody`) – The observer of the target. Resulting coordinates point from observer to target.
- **abcorr** (`str`) – A scalar string that indicates the aberration corrections to apply to the database of the target body to account for one-way light time and stellar aberration. Default is 'NONE'.
- **normalize** (`bool`, *Optional*) – Return unit vectors for position and velocity (light time output is unchanged)

Returns

(`x`: `numpy.ndarray`, `v`: `numpy.ndarray`, `lt`: `numpy.ndarray`) or (`x`: `float`, `v`: `float`, `lt`: `float`) Rectangular position and velocity vectors (`x`, `y`, `z`), (`v_x`, `v_y`, `v_z`) where position points from the planet center of mass location at `et` to the aberration-corrected location of the target. Light time (`lt`) between planetary body and target.

Return type

`tuple`

3.1.7 libera_utils.io

Modules

<code> caching </code>	Module containing code to manage local file caching
<code> filenaming </code>	Module for file naming utilities
<code> hdf </code>	Utils for HDF5 file handling
<code> manifest </code>	Module for manifest file handling
<code> smart_open </code>	Module for smart_open

libera_utils.io.caching

Module containing code to manage local file caching

Functions

<code> empty_local_cache_dir() </code>	Remove all cached files in the local cache.
<code> get_local_cache_dir() </code>	Determine where to cache files based on the system and installed package version.

libera_utils.io.caching.empty_local_cache_dir`libera_utils.io.caching.empty_local_cache_dir()`

Remove all cached files in the local cache.

Returns

List of removed files

Return type

list

libera_utils.io.caching.get_local_cache_dir`libera_utils.io.caching.get_local_cache_dir()`

Determine where to cache files based on the system and installed package version.

Returns

Path to the cache directory for this version of this package on the current system

Return type`pathlib.Path``libera_utils.io.caching.empty_local_cache_dir()`

Remove all cached files in the local cache.

Returns

List of removed files

Return type

list

`libera_utils.io.caching.get_local_cache_dir()`

Determine where to cache files based on the system and installed package version.

Returns

Path to the cache directory for this version of this package on the current system

Return type`pathlib.Path`**libera_utils.io.filenaming**

Module for file naming utilities

Functions

<code>format_semantic_version(semantic_version)</code>	Formats a semantic version string X.Y.Z into a filename-compatible string like VX-Y-Z, for X = major version, Y = minor version, Z = patch.
<code>get_current_revision_str()</code>	Get the current <code>r%y%j%H%M%S</code> string for filename revisions.
<code>get_current_version_str(package_name)</code>	Retrieve the current version of a (algorithm) package and format it for inclusion in a filename

libera_utils.io.filenaming.format_semantic_version

`libera_utils.io.filenaming.format_semantic_version(semantic_version: str) → str`

Formats a semantic version string X.Y.Z into a filename-compatible string like VX-Y-Z, for X = major version, Y = minor version, Z = patch.

Result is uppercase. Release candidate suffixes are allowed as no strict checking is done on the contents of X, Y, or Z. e.g. 1.2.3rc1 becomes V1-2-3RC1

Parameters

semantic_version (*str*) – String matching X.Y.Z where X, Y and Z are integers of any length

Return type

str

libera_utils.io.filenaming.get_current_revision_str

`libera_utils.io.filenaming.get_current_revision_str() → str`

Get the current `r%y%j%H%M%S` string for filename revisions.

Returns

Current (now) revision string.

Return type

str

libera_utils.io.filenaming.get_current_version_str

`libera_utils.io.filenaming.get_current_version_str(package_name: str) → str`

Retrieve the current version of a (algorithm) package and format it for inclusion in a filename

Parameters

package_name (*str*) – Package for which to retrieve a version string. This should be your algorithm package and it must use a semantic versioning scheme, configured in project metadata.

Returns

Version string in format vM1m2p3

Return type

str

Classes

<code>AbstractValidFilename(*args, **kwargs)</code>	Composition of a CloudPath/Path instance with some methods to perform regex validation on filenames
<code>AttitudeKernelFilename(*args, **kwargs)</code>	Class to construct, store, and manipulate an SPK filename
<code>EphemerisKernelFilename(*args, **kwargs)</code>	Class to construct, store, and manipulate an SPK filename
<code>L0Filename(*args, **kwargs)</code>	Filename validation class for L0 files from EDOS.
<code>LiberaDataProductFilename(*args, **kwargs)</code>	Filename validation class for L1B and L2 science products
<code>ManifestFilename(*args, **kwargs)</code>	Class for naming manifest files
<code>ProductName(value)</code>	Enum of valid product names as used in filenames, defined and sourced from the LASP-ASDC ICD

libera_utils.io.filenaming.AbstractValidFilename

class libera_utils.io.filenaming.**AbstractValidFilename**(*args, **kwargs)

Bases: ABC

Composition of a CloudPath/Path instance with some methods to perform regex validation on filenames

Attributes*archive_prefix*

Property that contains the generated prefix used for archiving, when applicable

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*args[, basepath])	Abstract method that must be implemented to provide hinting for required parts
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

__init__(*args, **kwargs)

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*args[, basepath])	Abstract method that must be implemented to provide hinting for required parts
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

Attributes

<code>archive_prefix</code>	Property that contains the generated prefix used for archiving, when applicable
<code>data_product_id</code>	Property that contains the DataProductIdentifier for this file type
<code>filename_parts</code>	Property that contains a namespace of filename parts
<code>path</code>	Property containing the file path
<code>processing_step_id</code>	Property that contains the ProcessingStepIdentifier that generates this file

static `_calculate_applicable_time`(*start: datetime, end: datetime*) → date

Based on the start time and end time of a file, returns the applicable time (date)

Parameters

- **start** (*datetime.datetime*) – Start of the applicable time range
- **end** (*datetime.datetime*) – End of the applicable time range

Returns

The date of the mean time between start and end

Return type

`datetime.date`

abstractmethod classmethod `_format_filename_parts`(***parts*)

Format parts into a filename

Note: When this is implemented by concrete classes, ****parts** becomes a set of explicitly named arguments

classmethod `_from_filename_parts`(**, basepath: str | Path | S3Path = None, **parts: Any*)

Create instance from filename parts.

The part kwarg names are named according to the regex for the file type.

Parameters

- **basepath** (*Union[str, Path, S3Path], Optional*) – Allows prepending a basepath or prefix.
- **parts** (*Any*) – Passed directly to `_format_filename_parts`. This is a dict of variable kwargs that will differ in each filename class based on the required parts for that particular filename type.

Return type

`AbstractValidFilename`

abstractmethod `_parse_filename_parts`()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

`types.SimpleNamespace`

abstract property `archive_prefix`: `str`

Property that contains the generated prefix used for archiving, when applicable

abstract property data_product_id: *DataProductIdentifier*

Property that contains the DataProductIdentifier for this file type

property filename_parts

Property that contains a namespace of filename parts

classmethod from_file_path(*args, **kwargs) → AVF

Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)

abstractmethod classmethod from_filename_parts(*args: Any, basepath: str | Path | S3Path = None, **kwargs: Any)

Abstract method that must be implemented to provide hinting for required parts

generate_prefixed_path(parent_path: str | Path | S3Path) → Path | S3Path

Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)

Parameters

parent_path (Union[str, Path, S3Path]) – Absolute path to the parent directory or S3 bucket prefix. The generated path prefix is appended to the parent path and followed by the file basename.

Return type

pathlib.Path or cloudpathlib.s3.s3path.S3Path

property path: Path | S3Path

Property containing the file path

abstract property processing_step_id: *ProcessingStepIdentifier*

Property that contains the ProcessingStepIdentifier that generates this file

regex_match(path: str | Path | S3Path)

Parse and validate a given path against class-attribute defined regex

Returns

Match group dict of filename parts

Return type

dict

libera_utils.io.filenaming.AttitudeKernelFilename

class libera_utils.io.filenaming.AttitudeKernelFilename(*args, **kwargs)

Bases: *AbstractValidFilename*

Class to construct, store, and manipulate an SPK filename

Attributes

archive_prefix

Property that contains the generated prefix for SPICE archiving

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*, ck_object, version, ...)	Create instance from filename parts.
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

__init__(*args, **kwargs)

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*, ck_object, version, ...)	Create instance from filename parts.
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

Attributes

<i>archive_prefix</i>	Property that contains the generated prefix for SPICE archiving
<i>data_product_id</i>	Property that contains the DataProductIdentifier for this file type
<i>filename_parts</i>	Property that contains a namespace of filename parts
<i>path</i>	Property containing the file path
<i>processing_step_id</i>	Property that contains the ProcessingStepIdentifier that generates this file

static *_calculate_applicable_time*(start: *datetime*, end: *datetime*) → date

Based on the start time and end time of a file, returns the applicable time (date)

Parameters

- **start** (*datetime.datetime*) – Start of the applicable time range
- **end** (*datetime.datetime*) – End of the applicable time range

Returns

The date of the mean time between start and end

Return type

datetime.date

classmethod `_format_filename_parts`(**ck_object*: str, *version*: str, *utc_start*: datetime, *utc_end*: datetime, *revision*: datetime)

Format filename parts as a string

Parameters

- **ck_object** (str) – Name of object whose attitude is represented in this CK.
- **utc_start** (datetime.datetime) – Start time of data.
- **utc_end** (datetime.datetime) – End time of data.
- **version** (str) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **revision** (datetime.datetime) – When the file was last revised.

Return type

str

classmethod `_from_filename_parts`(**basepath*: str | Path | S3Path = None, ***parts*: Any)

Create instance from filename parts.

The part kwarg names are named according to the regex for the file type.

Parameters

- **basepath** (Union[str, Path, S3Path], Optional) – Allows prepending a basepath or prefix.
- **parts** (Any) – Passed directly to `_format_filename_parts`. This is a dict of variable kwargs that will differ in each filename class based on the required parts for that particular filename type.

Return type

AbstractValidFilename

method `_parse_filename_parts`()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

types.SimpleNamespace

property `archive_prefix`: str

Property that contains the generated prefix for SPICE archiving

property `data_product_id`: *DataProductIdentifier*

Property that contains the DataProductIdentifier for this file type

property `filename_parts`

Property that contains a namespace of filename parts

classmethod `from_file_path`(*args, ***kwargs*) → AVF

Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)

classmethod `from_filename_parts`(**ck_object*: str, *version*: str, *utc_start*: datetime, *utc_end*: datetime, *revision*: datetime, *basepath*: str | Path | S3Path | None = None)

Create instance from filename parts.

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part arg names are named according to the regex for the file type.

Parameters

- **ck_object** (str) – Name of object whose attitude is represented in this CK.
- **version** (str) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (datetime.datetime) – Start time of data.
- **utc_end** (datetime.datetime) – End time of data.
- **revision** (datetime.datetime) – When the file was last revised.
- **basepath** (Optional[Union[str, Path, S3Path]]) – Allows prepending a basepath or prefix.

Return type

AttitudeKernelFilename

generate_prefixed_path(*parent_path*: str | Path | S3Path) → Path | S3Path

Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)

Parameters

parent_path (Union[str, Path, S3Path]) – Absolute path to the parent directory or S3 bucket prefix. The generated path prefix is appended to the parent path and followed by the file basename.

Return type

pathlib.Path or cloudpathlib.s3.s3path.S3Path

property path: Path | S3Path

Property containing the file path

property processing_step_id: *ProcessingStepIdentifier*

Property that contains the ProcessingStepIdentifier that generates this file

regex_match(*path*: str | Path | S3Path)

Parse and validate a given path against class-attribute defined regex

Returns

Match group dict of filename parts

Return type

dict

libera_utils.io.filenaming.EphemerisKernelFilename

class `libera_utils.io.filenaming.EphemerisKernelFilename`(*args, **kwargs)

Bases: *AbstractValidFilename*

Class to construct, store, and manipulate an SPK filename

Attributes

archive_prefix

Property that contains the generated prefix for SPICE archiving

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*args, spk_object, version, ...)	Create instance from filename parts.
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

__init__(*args, **kwargs)

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*args, spk_object, version, ...)	Create instance from filename parts.
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

Attributes

<i>archive_prefix</i>	Property that contains the generated prefix for SPICE archiving
<i>data_product_id</i>	Property that contains the DataProductIdentifier for this file type
<i>filename_parts</i>	Property that contains a namespace of filename parts
<i>path</i>	Property containing the file path
<i>processing_step_id</i>	Property that contains the ProcessingStepIdentifier that generates this file

static `_calculate_applicable_time`(*start: datetime, end: datetime*) → date

Based on the start time and end time of a file, returns the applicable time (date)

Parameters

- **start** (*datetime.datetime*) – Start of the applicable time range
- **end** (*datetime.datetime*) – End of the applicable time range

Returns

The date of the mean time between start and end

Return type

datetime.date

classmethod `_format_filename_parts`(**, spk_object: str, version: str, utc_start: datetime, utc_end: datetime, revision: datetime*)

Format filename parts as a string

Parameters

- **spk_object** (*str*) – Name of object whose ephemeris is represented in this SPK.
- **version** (*str*) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (*datetime.datetime*) – Start time of data.
- **utc_end** (*datetime.datetime*) – End time of data.
- **revision** (*datetime.datetime*) – Time when the file was last revised

Return type

str

classmethod `_from_filename_parts`(**, basepath: str | Path | S3Path = None, **parts: Any*)

Create instance from filename parts.

The part kwarg names are named according to the regex for the file type.

Parameters

- **basepath** (*Union[str, Path, S3Path], Optional*) – Allows prepending a basepath or prefix.
- **parts** (*Any*) – Passed directly to `_format_filename_parts`. This is a dict of variable kwargs that will differ in each filename class based on the required parts for that particular filename type.

Return type

AbstractValidFilename

`_parse_filename_parts`()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

types.SimpleNamespace

property `archive_prefix`: *str*

Property that contains the generated prefix for SPICE archiving

property data_product_id: *DataProductIdentifier*

Property that contains the DataProductIdentifier for this file type

property filename_parts

Property that contains a namespace of filename parts

classmethod from_file_path(*args, **kwargs) → AVF

Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)

classmethod from_filename_parts(**spk_object*: str, *version*: str, *utc_start*: datetime, *utc_end*: datetime, *revision*: datetime, *basepath*: str | Path | S3Path | None = None)

Create instance from filename parts.

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part arg names are named according to the regex for the file type.

Parameters

- **spk_object** (str) – Name of object whose attitude is represented in this SPK.
- **version** (str) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (datetime.datetime) – Start time of data.
- **utc_end** (datetime.datetime) – End time of data.
- **revision** (datetime.datetime) – When the file was last revised.
- **basepath** (Optional[Union[str, Path, S3Path]]) – Allows prepending a basepath or prefix.

Return type

EphemerisKernelFilename

generate_prefixed_path(parent_path: str | Path | S3Path) → Path | S3Path

Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename}. The parent_path can be an S3 bucket or an absolute local filepath (must start with /)

Parameters

parent_path (Union[str, Path, S3Path]) – Absolute path to the parent directory or S3 bucket prefix. The generated path prefix is appended to the parent path and followed by the file basename.

Return type

pathlib.Path or cloudpathlib.s3.s3path.S3Path

property path: Path | S3Path

Property containing the file path

property processing_step_id: *ProcessingStepIdentifier*

Property that contains the ProcessingStepIdentifier that generates this file

regex_match(path: str | Path | S3Path)

Parse and validate a given path against class-attribute defined regex

Returns

Match group dict of filename parts

Return type

dict

libera_utils.io.filenaming.L0Filename

class libera_utils.io.filenaming.L0Filename(*args, **kwargs)

Bases: *AbstractValidFilename*

Filename validation class for L0 files from EDOS.

Attributes*archive_prefix*

Property that contains the generated prefix for L0 archiving

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*id_char, scid, ...[, ...])	Create instance from filename parts
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

__init__(*args, **kwargs)

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*id_char, scid, ...[, ...])	Create instance from filename parts
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

Attributes

<i>archive_prefix</i>	Property that contains the generated prefix for L0 archiving
<i>data_product_id</i>	Property that contains the DataProductIdentifier for this file type
<i>filename_parts</i>	Property that contains a namespace of filename parts
<i>path</i>	Property containing the file path
<i>processing_step_id</i>	Property that contains the ProcessingStepIdentifier that generates this file

static `_calculate_applicable_time(start: datetime.datetime, end: datetime.datetime)` → `date`

Based on the start time and end time of a file, returns the applicable time (date)

Parameters

- **start** (*datetime.datetime*) – Start of the applicable time range
- **end** (*datetime.datetime*) – End of the applicable time range

Returns

The date of the mean time between start and end

Return type

`datetime.date`

classmethod `_format_filename_parts(*, id_char: str, scid: int, first_apid: int, fill: str, created_time: datetime.datetime, numeric_id: int, file_number: int, extension: str, signal: str | None = None)`

Construct a path from filename parts

Parameters

- **id_char** (*str*) – Either P (for PDS files, Construction Records) or X (for Delivery Records)
- **scid** (*int*) – Spacecraft ID
- **first_apid** (*int*) – First APID in the file
- **fill** (*str*) – Custom string up to 14 characters long
- **created_time** (*datetime.datetime*) – Creation time of the file
- **numeric_id** (*int*) – Data set ID, 0-9, one digit
- **file_number** (*str*) – File number within the data set. Construction records are always file number zero.
- **extension** (*str*) – File name extension. Either PDR or PDS
- **signal** (*Optional[str]*, *Optional*) – Optional signal suffix. Always ‘.XFR’

Returns

Formatted filename

Return type

`str`

classmethod `_from_filename_parts`(**basepath*: *str* | *Path* | *S3Path* = *None*, ***parts*: *Any*)

Create instance from filename parts.

The part kwarg names are named according to the regex for the file type.

Parameters

- **basepath** (*Union[str, Path, S3Path]*, *Optional*) – Allows prepending a basepath or prefix.
- **parts** (*Any*) – Passed directly to `_format_filename_parts`. This is a dict of variable kwargs that will differ in each filename class based on the required parts for that particular filename type.

Return type

AbstractValidFilename

classmethod `_parse_filename_parts`()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

types.SimpleNamespace

property `archive_prefix`: *str*

Property that contains the generated prefix for L0 archiving

property `data_product_id`: *DataProductIdentifier*

Property that contains the *DataProductIdentifier* for this file type

property `filename_parts`

Property that contains a namespace of filename parts

classmethod `from_file_path`(**args*, ***kwargs*) → AVF

Factory method to produce an *AbstractValidFilename* from a valid Libera file path (*str* or *Path*)

classmethod `from_filename_parts`(**id_char*: *str*, *scid*: *int*, *first_apid*: *int*, *fill*: *str*, *created_time*: *datetime.datetime*, *numeric_id*: *int*, *file_number*: *int*, *extension*: *str*, *signal*: *str* | *None* = *None*, *basepath*: *str* | *Path* | *S3Path* | *None* = *None*)

Create instance from filename parts

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part names are named according to the regex for the file type.

Parameters

- **id_char** (*str*) – Either P (for PDS files, Construction Records) or X (for Delivery Records)
- **scid** (*int*) – Spacecraft ID
- **first_apid** (*int*) – First APID in the file
- **fill** (*str*) – Custom string up to 14 characters long
- **created_time** (*datetime.datetime*) – Creation time of the file
- **numeric_id** (*int*) – Data set ID, 0-9, one digit
- **file_number** (*str*) – File number within the data set. Construction records are always file number zero.

- **extension** (*str*) – File name extension. Either PDR or PDS
- **signal** (*Optional[str]*) – Optional signal suffix. Always ‘.XFR’
- **basepath** (*Optional[Union[str, Path, S3Path]]*) – Allows prepending a basepath or prefix.

Return type*LOFilename***generate_prefixed_path**(*parent_path: str | Path | S3Path*) → *Path | S3Path*

Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)

Parameters

parent_path (*Union[str, Path, S3Path]*) – Absolute path to the parent directory or S3 bucket prefix. The generated path prefix is appended to the parent path and followed by the file basename.

Return type*pathlib.Path* or *cloudpathlib.s3.s3path.S3Path***property path:** *Path | S3Path*

Property containing the file path

property processing_step_id: *ProcessingStepIdentifier*

Property that contains the ProcessingStepIdentifier that generates this file

regex_match(*path: str | Path | S3Path*)

Parse and validate a given path against class-attribute defined regex

Returns

Match group dict of filename parts

Return type*dict***libera_utils.io.filenaming.LiberaDataProductFilename****class** *libera_utils.io.filenaming.LiberaDataProductFilename*(*args, **kwargs)Bases: *AbstractValidFilename*

Filename validation class for L1B and L2 science products

Attributes*archive_prefix*

Property that contains the generated prefix for L1B and L2 archiving

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<code>from_file_path(*args, **kwargs)</code>	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<code>from_filename_parts(*, data_level, ..., ...)</code>	Create instance from filename parts.
<code>generate_prefixed_path(parent_path)</code>	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<code>regex_match(path)</code>	Parse and validate a given path against class-attribute defined regex

`__init__(*args, **kwargs)`

Methods

<code>from_file_path(*args, **kwargs)</code>	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<code>from_filename_parts(*, data_level, ..., ...)</code>	Create instance from filename parts.
<code>generate_prefixed_path(parent_path)</code>	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<code>regex_match(path)</code>	Parse and validate a given path against class-attribute defined regex

Attributes

<code>archive_prefix</code>	Property that contains the generated prefix for L1B and L2 archiving
<code>data_product_id</code>	Property that contains the DataProductIdentifier for this file type
<code>filename_parts</code>	Property that contains a namespace of filename parts
<code>path</code>	Property containing the file path
<code>processing_step_id</code>	Property that contains the ProcessingStepIdentifier that generates this file

`static _calculate_applicable_time(start: datetime, end: datetime) → date`

Based on the start time and end time of a file, returns the applicable time (date)

Parameters

- **start** (`datetime.datetime`) – Start of the applicable time range
- **end** (`datetime.datetime`) – End of the applicable time range

Returns

The date of the mean time between start and end

Return type

`datetime.date`

classmethod `_format_filename_parts`(*, *data_level: str, product_name: str, version: str, utc_start: datetime, utc_end: datetime, revision: datetime, extension: str*)

Construct a path from filename parts

Parameters

- **data_level** (*str*) – L1B or L2
- **product_name** (*str*) – Libera instrument, cam or rad for L1B and cloud-fraction etc. for L2. May contain anything except for underscores.
- **version** (*str*) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (*datetime.datetime*) – First timestamp in the SPK
- **utc_end** (*datetime.datetime*) – Last timestamp in the SPK
- **revision** (*datetime.datetime*) – Time when the file was created.
- **extension** (*str*) – File extension (.nc or .h5)

Returns

Formatted filename

Return type

str

classmethod `_from_filename_parts`(*, *basepath: str | Path | S3Path = None, **parts: Any*)

Create instance from filename parts.

The part kwarg names are named according to the regex for the file type.

Parameters

- **basepath** (*Union[str, Path, S3Path], Optional*) – Allows prepending a basepath or prefix.
- **parts** (*Any*) – Passed directly to `_format_filename_parts`. This is a dict of variable kwargs that will differ in each filename class based on the required parts for that particular filename type.

Return type

AbstractValidFilename

method `_parse_filename_parts`()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

types.SimpleNamespace

property `archive_prefix`: *str*

Property that contains the generated prefix for L1B and L2 archiving

property `data_product_id`: *DataProductIdentifier*

Property that contains the DataProductIdentifier for this file type

property `filename_parts`

Property that contains a namespace of filename parts

classmethod `from_file_path(*args, **kwargs)` → AVF

Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)

classmethod `from_filename_parts(*, data_level: str, product_name: str, version: str, utc_start: datetime, utc_end: datetime, revision: datetime, extension: str = 'nc', basepath: str | Path | S3Path | None = None)`

Create instance from filename parts. All keyword arguments other than basepath are required!

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part names are named according to the regex for the file type.

Parameters

- **data_level** (*str*) – L1B or L2 identifying the level of the data product
- **product_name** (*str*) – Product type. e.g. cloud-fraction for L2 or cam for L1B. May contain anything except for underscores.
- **version** (*str*) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (*datetime.datetime*) – First timestamp in the SPK
- **utc_end** (*datetime.datetime*) – Last timestamp in the SPK
- **revision** (*datetime.datetime*) – Time when the file was created.
- **extension** (*str*) – File extension (.nc or .h5)
- **basepath** (*Optional[Union[str, Path, S3Path]]*) – Allows prepending a basepath or prefix.

Return type

LiberaDataProductFilename

generate_prefixed_path(*parent_path: str | Path | S3Path*) → *Path | S3Path*

Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)

Parameters

parent_path (*Union[str, Path, S3Path]*) – Absolute path to the parent directory or S3 bucket prefix. The generated path prefix is appended to the parent path and followed by the file basename.

Return type

pathlib.Path or *cloudpathlib.s3.s3path.S3Path*

property path: *Path | S3Path*

Property containing the file path

property processing_step_id: *ProcessingStepIdentifier*

Property that contains the ProcessingStepIdentifier that generates this file

regex_match(*path: str | Path | S3Path*)

Parse and validate a given path against class-attribute defined regex

Returns

Match group dict of filename parts

Return type

dict

libera_utils.io.filenaming.ManifestFilename

class libera_utils.io.filenaming.**ManifestFilename**(*args, **kwargs)

Bases: *AbstractValidFilename*

Class for naming manifest files

Attributes*archive_prefix*

Manifests are not archived like data products, but for convenience and ease of debugging they will be kept in the dropbox bucket by input/output and day they were made.

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (manifest_type, ulid_code)	Create instance from filename parts.
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

__init__(*args, **kwargs)

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (manifest_type, ulid_code)	Create instance from filename parts.
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

Attributes

<code>archive_prefix</code>	Manifests are not archived like data products, but for convenience and ease of debugging they will be kept in the dropbox bucket by input/output and day they were made.
<code>data_product_id</code>	Property that contains the DataProductIdentifier for this file type
<code>filename_parts</code>	Property that contains a namespace of filename parts
<code>path</code>	Property containing the file path
<code>processing_step_id</code>	Property that contains the ProcessingStepIdentifier that generates this file

static `_calculate_applicable_time(start: datetime, end: datetime) → date`

Based on the start time and end time of a file, returns the applicable time (date)

Parameters

- **start** (`datetime.datetime`) – Start of the applicable time range
- **end** (`datetime.datetime`) – End of the applicable time range

Returns

The date of the mean time between start and end

Return type

`datetime.date`

classmethod `_format_filename_parts(manifest_type: ManifestType, ulid_code: ULID)`

Construct a path from filename parts

Parameters

- **manifest_type** (`ManifestType`) – Input or output
- **ulid_code** (`ulid.ULID`) – ULID code for use in filename parts

Returns

Formatted filename

Return type

`str`

classmethod `_from_filename_parts(*, basepath: str | Path | S3Path = None, **parts: Any)`

Create instance from filename parts.

The part kwarg names are named according to the regex for the file type.

Parameters

- **basepath** (`Union[str, Path, S3Path]`, *Optional*) – Allows prepending a basepath or prefix.
- **parts** (*Any*) – Passed directly to `_format_filename_parts`. This is a dict of variable kwargs that will differ in each filename class based on the required parts for that particular filename type.

Return type

`AbstractValidFilename`

_parse_filename_parts()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

`types.SimpleNamespace`

property archive_prefix: str

Manifests are not archived like data products, but for convenience and ease of debugging they will be kept in the dropbox bucket by input/output and day they were made. This is used by the step function clean up function in the CDK. # Generate prefix structure # <manifest_type>/<year>/<month>/<day>

property data_product_id: DataProductIdentifier

Property that contains the DataProductIdentifier for this file type

property filename_parts

Property that contains a namespace of filename parts

classmethod from_file_path(*args, **kwargs) → AVF

Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)

classmethod from_filename_parts(manifest_type: ManifestType, ulid_code: ULID, basepath: str | Path | S3Path = None)

Create instance from filename parts.

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part names are named according to the regex for the file type.

Parameters

- **manifest_type** (`ManifestType`) – Input or output
- **ulid_code** (`ulid.ULID`) – ULID code for use in filename parts
- **basepath** (`Optional[Union[str, Path, S3Path]]`) – Allows prepending a basepath or prefix.

Return type

`ManifestFilename`

generate_prefixed_path(parent_path: str | Path | S3Path) → Path | S3Path

Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)

Parameters

parent_path (`Union[str, Path, S3Path]`) – Absolute path to the parent directory or S3 bucket prefix. The generated path prefix is appended to the parent path and followed by the file basename.

Return type

`pathlib.Path` or `cloudpathlib.s3.s3path.S3Path`

property path: Path | S3Path

Property containing the file path

property processing_step_id: ProcessingStepIdentifier

Property that contains the ProcessingStepIdentifier that generates this file

regex_match(*path*: *str* | *Path* | *S3Path*)

Parse and validate a given path against class-attribute defined regex

Returns

Match group dict of filename parts

Return type

dict

libera_utils.io.filenaming.ProductName

class libera_utils.io.filenaming.**ProductName**(*value*)

Bases: `StrEnum`

Enum of valid product names as used in filenames, defined and sourced from the LASP-ASDC ICD

Methods

<code>capitalize()</code>	Return a capitalized version of the string.
<code>casefold()</code>	Return a version of the string suitable for caseless comparisons.
<code>center(width[, fillchar])</code>	Return a centered string of length width.
<code>count(sub[, start[, end]])</code>	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<code>encode([, encoding, errors])</code>	Encode the string using the codec registered for encoding.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>expandtabs([, tabsize])</code>	Return a copy where all tab characters are expanded using spaces.
<code>find(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>index(sub[, start[, end]])</code>	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<code>isalnum()</code>	Return True if the string is an alpha-numeric string, False otherwise.
<code>isalpha()</code>	Return True if the string is an alphabetic string, False otherwise.
<code>isascii()</code>	Return True if all characters in the string are ASCII, False otherwise.
<code>isdecimal()</code>	Return True if the string is a decimal string, False otherwise.
<code>isdigit()</code>	Return True if the string is a digit string, False otherwise.
<code>isidentifier()</code>	Return True if the string is a valid Python identifier, False otherwise.
<code>islower()</code>	Return True if the string is a lowercase string, False otherwise.

continues on next page

Table 70 – continued from previous page

<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.
<i>isprintable()</i>	Return True if the string is printable, False otherwise.
<i>isspace()</i>	Return True if the string is a whitespace string, False otherwise.
<i>istitle()</i>	Return True if the string is a title-cased string, False otherwise.
<i>isupper()</i>	Return True if the string is an uppercase string, False otherwise.
<i>join(iterable, /)</i>	Concatenate any number of strings.
<i>ljust(width[, fillchar])</i>	Return a left-justified string of length width.
<i>lower()</i>	Return a copy of the string converted to lowercase.
<i>lstrip([chars])</i>	Return a copy of the string with leading whitespace removed.
<i>maketrans(x[, y, z])</i>	Return a translation table usable for str.translate().
<i>partition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>removeprefix(prefix, /)</i>	Return a str with the given prefix string removed if present.
<i>removesuffix(suffix, /)</i>	Return a str with the given suffix string removed if present.
<i>replace(old, new[, count])</i>	Return a copy with all occurrences of substring old replaced by new.
<i>rfind(sub[, start[, end]])</i>	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex(sub[, start[, end]])</i>	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust(width[, fillchar])</i>	Return a right-justified string of length width.
<i>rpartition(sep, /)</i>	Partition the string into three parts using the given separator.
<i>rsplit(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using sep as the separator string.
<i>rstrip([chars])</i>	Return a copy of the string with trailing whitespace removed.
<i>split(/[, sep, maxsplit])</i>	Return a list of the substrings in the string, using sep as the separator string.
<i>splitlines(/[, keepends])</i>	Return a list of the lines in the string, breaking at line boundaries.
<i>startswith(prefix[, start[, end]])</i>	Return True if S starts with the specified prefix, False otherwise.
<i>strip([chars])</i>	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase()</i>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>title()</i>	Return a version of the string where each word is titlecased.
<i>translate(table, /)</i>	Replace each character in the string using the given translation table.
<i>upper()</i>	Return a copy of the string converted to uppercase.
<i>zfill(width, /)</i>	Pad a numeric string with zeros on the left, to fill a field of the given width.

`__init__(*args, **kwargs)`

Methods

<i>encode</i> ([, encoding, errors])	Encode the string using the codec registered for encoding.
<i>replace</i> (old, new[, count])	Return a copy with all occurrences of substring old replaced by new.
<i>split</i> ([, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>rsplit</i> ([, sep, maxsplit])	Return a list of the substrings in the string, using sep as the separator string.
<i>join</i> (iterable, /)	Concatenate any number of strings.
<i>capitalize</i> (/)	Return a capitalized version of the string.
<i>casefold</i> (/)	Return a version of the string suitable for caseless comparisons.
<i>title</i> (/)	Return a version of the string where each word is titlecased.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>expandtabs</i> ([, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>partition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>ljust</i> (width[, fillchar])	Return a left-justified string of length width.
<i>lower</i> (/)	Return a copy of the string converted to lowercase.
<i>lstrip</i> ([chars])	Return a copy of the string with leading whitespace removed.
<i>rfind</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rindex</i> (sub[, start[, end]])	Return the highest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>rjust</i> (width[, fillchar])	Return a right-justified string of length width.
<i>rstrip</i> ([chars])	Return a copy of the string with trailing whitespace removed.
<i>rpartition</i> (sep, /)	Partition the string into three parts using the given separator.
<i>splitlines</i> ([, keepends])	Return a list of the lines in the string, breaking at line boundaries.
<i>strip</i> ([chars])	Return a copy of the string with leading and trailing whitespace removed.
<i>swapcase</i> (/)	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<i>translate</i> (table, /)	Replace each character in the string using the given translation table.
<i>upper</i> (/)	Return a copy of the string converted to uppercase.

continues on next page

Table 71 – continued from previous page

<code>startswith(prefix[, start[, end]])</code>	Return True if S starts with the specified prefix, False otherwise.
<code>endswith(suffix[, start[, end]])</code>	Return True if S ends with the specified suffix, False otherwise.
<code>removeprefix(prefix, /)</code>	Return a str with the given prefix string removed if present.
<code>removesuffix(suffix, /)</code>	Return a str with the given suffix string removed if present.
<code>isascii(/)</code>	Return True if all characters in the string are ASCII, False otherwise.
<code>islower(/)</code>	Return True if the string is a lowercase string, False otherwise.
<code>isupper(/)</code>	Return True if the string is an uppercase string, False otherwise.
<code>istitle(/)</code>	Return True if the string is a title-cased string, False otherwise.
<code>isspace(/)</code>	Return True if the string is a whitespace string, False otherwise.
<code>isdecimal(/)</code>	Return True if the string is a decimal string, False otherwise.
<code>isdigit(/)</code>	Return True if the string is a digit string, False otherwise.
<code>isnumeric(/)</code>	Return True if the string is a numeric string, False otherwise.
<code>isalpha(/)</code>	Return True if the string is an alphabetic string, False otherwise.
<code>isalnum(/)</code>	Return True if the string is an alpha-numeric string, False otherwise.
<code>isidentifier(/)</code>	Return True if the string is a valid Python identifier, False otherwise.
<code>isprintable(/)</code>	Return True if the string is printable, False otherwise.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.
<code>format(*args, **kwargs)</code>	Return a formatted version of S, using substitutions from args and kwargs.
<code>format_map(mapping)</code>	Return a formatted version of S, using substitutions from mapping.
<code>maketrans(x[, y, z])</code>	Return a translation table usable for <code>str.translate()</code> .

Attributes

<code>processing_step_id</code>	ProcessingStepIdentifier for this product name
<code>data_product_id</code>	DataProductIdentifier for this product name
RAD	
CAM	

`capitalize(/)`

Return a capitalized version of the string.

More specifically, make the first character have upper case and the rest lower case.

casefold(/)

Return a version of the string suitable for caseless comparisons.

center(width, fillchar=' ', /)

Return a centered string of length width.

Padding is done using the specified fill character (default is a space).

count(sub[, start[, end]]) → int

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

property data_product_id: *DataProductIdentifier*

DataProductIdentifier for this product name

encode(/, encoding='utf-8', errors='strict')

Encode the string using the codec registered for encoding.

encoding

The encoding in which to encode the string.

errors

The error handling scheme to use for encoding errors. The default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith(suffix[, start[, end]]) → bool

Return True if S ends with the specified suffix, False otherwise. With optional start, test S beginning at that position. With optional end, stop comparing S at that position. suffix can also be a tuple of strings to try.

expandtabs(/, tabsize=8)

Return a copy where all tab characters are expanded using spaces.

If tabsize is not given, a tab size of 8 characters is assumed.

find(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Return -1 on failure.

format(*args, **kwargs) → str

Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(mapping) → str

Return a formatted version of S, using substitutions from mapping. The substitutions are identified by braces ('{' and '}').

index(sub[, start[, end]]) → int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

isalnum()

Return True if the string is an alpha-numeric string, False otherwise.

A string is alpha-numeric if all characters in the string are alpha-numeric and there is at least one character in the string.

isalpha()

Return True if the string is an alphabetic string, False otherwise.

A string is alphabetic if all characters in the string are alphabetic and there is at least one character in the string.

isascii()

Return True if all characters in the string are ASCII, False otherwise.

ASCII characters have code points in the range U+0000-U+007F. Empty string is ASCII too.

isdecimal()

Return True if the string is a decimal string, False otherwise.

A string is a decimal string if all characters in the string are decimal and there is at least one character in the string.

isdigit()

Return True if the string is a digit string, False otherwise.

A string is a digit string if all characters in the string are digits and there is at least one character in the string.

isidentifier()

Return True if the string is a valid Python identifier, False otherwise.

Call `keyword.iskeyword(s)` to test whether string `s` is a reserved identifier, such as “def” or “class”.

islower()

Return True if the string is a lowercase string, False otherwise.

A string is lowercase if all cased characters in the string are lowercase and there is at least one cased character in the string.

isnumeric()

Return True if the string is a numeric string, False otherwise.

A string is numeric if all characters in the string are numeric and there is at least one character in the string.

isprintable()

Return True if the string is printable, False otherwise.

A string is printable if all of its characters are considered printable in `repr()` or if it is empty.

isspace()

Return True if the string is a whitespace string, False otherwise.

A string is whitespace if all characters in the string are whitespace and there is at least one character in the string.

istitle()

Return True if the string is a title-cased string, False otherwise.

In a title-cased string, upper- and title-case characters may only follow uncased characters and lowercase characters only cased ones.

isupper(/)

Return True if the string is an uppercase string, False otherwise.

A string is uppercase if all cased characters in the string are uppercase and there is at least one cased character in the string.

join(*iterable*, /)

Concatenate any number of strings.

The string whose method is called is inserted in between each given string. The result is returned as a new string.

Example: `'.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'`

ljust(*width*, *fillchar*=' ', /)

Return a left-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

lower(/)

Return a copy of the string converted to lowercase.

lstrip(*chars*=None, /)

Return a copy of the string with leading whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

static maketrans(*x*, *y*=<unrepresentable>, *z*=<unrepresentable>, /)

Return a translation table usable for `str.translate()`.

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two arguments, they must be strings of equal length, and in the resulting dictionary, each character in *x* will be mapped to the character at the same position in *y*. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

partition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing the original string and two empty strings.

property processing_step_id: *ProcessingStepIdentifier*

ProcessingStepIdentifier for this product name

removeprefix(*prefix*, /)

Return a str with the given prefix string removed if present.

If the string starts with the prefix string, return `string[len(prefix):]`. Otherwise, return a copy of the original string.

removesuffix(*suffix*, /)

Return a str with the given suffix string removed if present.

If the string ends with the suffix string and that suffix is not empty, return `string[:-len(suffix)]`. Otherwise, return a copy of the original string.

replace(*old*, *new*, *count*=-1, /)

Return a copy with all occurrences of substring *old* replaced by *new*.

count

Maximum number of occurrences to replace. -1 (the default value) means replace all occurrences.

If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex(*sub*[, *start*[, *end*]]) → int

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Raises `ValueError` when the substring is not found.

rjust(*width*, *fillchar*=' ', /)

Return a right-justified string of length *width*.

Padding is done using the specified fill character (default is a space).

rpartition(*sep*, /)

Partition the string into three parts using the given separator.

This will search for the separator in the string, starting at the end. If the separator is found, returns a 3-tuple containing the part before the separator, the separator itself, and the part after it.

If the separator is not found, returns a 3-tuple containing two empty strings and the original string.

rsplit(/, *sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to `None` (the default value), will split on any whitespace character (including `n r t f` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the end of the string and works to the front.

rstrip(*chars*=None, /)

Return a copy of the string with trailing whitespace removed.

If *chars* is given and not `None`, remove characters in *chars* instead.

split(/, *sep*=None, *maxsplit*=-1)

Return a list of the substrings in the string, using *sep* as the separator string.

sep

The separator used to split the string.

When set to `None` (the default value), will split on any whitespace character (including `n r t f` and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits. -1 (the default value) means no limit.

Splitting starts at the front of the string and works to the end.

Note, `str.split()` is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

splitlines(/, *keepends=False*)

Return a list of the lines in the string, breaking at line boundaries.

Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith(*prefix*[, *start*[, *end*]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional *start*, test S beginning at that position. With optional *end*, stop comparing S at that position. *prefix* can also be a tuple of strings to try.

strip(*chars=None*, /)

Return a copy of the string with leading and trailing whitespace removed.

If *chars* is given and not None, remove characters in *chars* instead.

swapcase(/)

Convert uppercase characters to lowercase and lowercase characters to uppercase.

title(/)

Return a version of the string where each word is titlecased.

More specifically, words start with uppercased characters and all remaining cased characters have lower case.

translate(*table*, /)

Replace each character in the string using the given translation table.

table

Translation table, which must be a mapping of Unicode ordinals to Unicode ordinals, strings, or None.

The table must implement lookup/indexing via `__getitem__`, for instance a dictionary or list. If this operation raises `LookupError`, the character is left untouched. Characters mapped to None are deleted.

upper(/)

Return a copy of the string converted to uppercase.

zfill(*width*, /)

Pad a numeric string with zeros on the left, to fill a field of the given width.

The string is never truncated.

class libera_utils.io.filenameing.**AbstractValidFilename**(*args, **kwargs)

Composition of a `CloudPath/Path` instance with some methods to perform regex validation on filenames

Attributes***archive_prefix***

Property that contains the generated prefix used for archiving, when applicable

data_product_id

Property that contains the `DataProductIdentifier` for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<code>from_file_path(*args, **kwargs)</code>	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<code>from_filename_parts(*args[, basepath])</code>	Abstract method that must be implemented to provide hinting for required parts
<code>generate_prefixed_path(parent_path)</code>	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<code>regex_match(path)</code>	Parse and validate a given path against class-attribute defined regex

static `_calculate_applicable_time(start: datetime, end: datetime) → date`

Based on the start time and end time of a file, returns the applicable time (date)

Parameters

- **start** (`datetime.datetime`) – Start of the applicable time range
- **end** (`datetime.datetime`) – End of the applicable time range

Returns

The date of the mean time between start and end

Return type

`datetime.date`

abstractmethod classmethod `_format_filename_parts(**parts)`

Format parts into a filename

Note: When this is implemented by concrete classes, ****parts** becomes a set of explicitly named arguments

classmethod `_from_filename_parts(*, basepath: str | Path | S3Path = None, **parts: Any)`

Create instance from filename parts.

The part kwarg names are named according to the regex for the file type.

Parameters

- **basepath** (`Union[str, Path, S3Path]`, *Optional*) – Allows prepending a basepath or prefix.
- **parts** (*Any*) – Passed directly to `_format_filename_parts`. This is a dict of variable kwargs that will differ in each filename class based on the required parts for that particular filename type.

Return type

`AbstractValidFilename`

abstractmethod `_parse_filename_parts()`

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

`types.SimpleNamespace`

abstract property archive_prefix: `str`

Property that contains the generated prefix used for archiving, when applicable

abstract property data_product_id: `DataProductIdentifier`

Property that contains the DataProductIdentifier for this file type

property filename_parts

Property that contains a namespace of filename parts

classmethod from_file_path(*args, **kwargs) → AVF

Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)

abstractmethod classmethod from_filename_parts(*args: Any, basepath: str | Path | S3Path = None, **kwargs: Any)

Abstract method that must be implemented to provide hinting for required parts

generate_prefixed_path(parent_path: str | Path | S3Path) → Path | S3Path

Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)

Parameters

parent_path (`Union[str, Path, S3Path]`) – Absolute path to the parent directory or S3 bucket prefix. The generated path prefix is appended to the parent path and followed by the file basename.

Return type

`pathlib.Path` or `cloudpathlib.s3.s3path.S3Path`

property path: `Path | S3Path`

Property containing the file path

abstract property processing_step_id: `ProcessingStepIdentifier`

Property that contains the ProcessingStepIdentifier that generates this file

regex_match(path: str | Path | S3Path)

Parse and validate a given path against class-attribute defined regex

Returns

Match group dict of filename parts

Return type

`dict`

class libera_utils.io.filenaming.AttitudeKernelFilename(*args, **kwargs)

Class to construct, store, and manipulate an SPK filename

Attributes

archive_prefix

Property that contains the generated prefix for SPICE archiving

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*, ck_object, version, ...)	Create instance from filename parts.
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

classmethod *_format_filename_parts*(*, *ck_object*: str, *version*: str, *utc_start*: datetime, *utc_end*: datetime, *revision*: datetime)

Format filename parts as a string

Parameters

- **ck_object** (str) – Name of object whose attitude is represented in this CK.
- **utc_start** (datetime.datetime) – Start time of data.
- **utc_end** (datetime.datetime) – End time of data.
- **version** (str) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **revision** (datetime.datetime) – When the file was last revised.

Return type

str

_parse_filename_parts()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

types.SimpleNamespace

property *archive_prefix*: str

Property that contains the generated prefix for SPICE archiving

property *data_product_id*: *DataProductIdentifier*

Property that contains the DataProductIdentifier for this file type

classmethod *from_filename_parts*(*, *ck_object*: str, *version*: str, *utc_start*: datetime, *utc_end*: datetime, *revision*: datetime, *basepath*: str | Path | S3Path | None = None)

Create instance from filename parts.

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part arg names are named according to the regex for the file type.

Parameters

- **ck_object** (*str*) – Name of object whose attitude is represented in this CK.
- **version** (*str*) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (*datetime.datetime*) – Start time of data.
- **utc_end** (*datetime.datetime*) – End time of data.
- **revision** (*datetime.datetime*) – When the file was last revised.
- **basepath** (*Optional[Union[str, Path, S3Path]]*) – Allows prepending a basepath or prefix.

Return type

AttitudeKernelFilename

property processing_step_id: *ProcessingStepIdentifier*

Property that contains the ProcessingStepIdentifier that generates this file

class libera_utils.io.filenaming.EphemerisKernelFilename(*args, **kwargs)

Class to construct, store, and manipulate an SPK filename

Attributes

archive_prefix

Property that contains the generated prefix for SPICE archiving

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*, spk_object, version, ...)	Create instance from filename parts.
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

classmethod `_format_filename_parts`(*, *spk_object*: *str*, *version*: *str*, *utc_start*: *datetime*, *utc_end*: *datetime*, *revision*: *datetime*)

Format filename parts as a string

Parameters

- **spk_object** (*str*) – Name of object whose ephemeris is represented in this SPK.
- **version** (*str*) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (*datetime.datetime*) – Start time of data.
- **utc_end** (*datetime.datetime*) – End time of data.
- **revision** (*datetime.datetime*) – Time when the file was last revised

Return type

str

_parse_filename_parts()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

types.SimpleNamespace

property `archive_prefix`: *str*

Property that contains the generated prefix for SPICE archiving

property `data_product_id`: *DataProductIdentifier*

Property that contains the DataProductIdentifier for this file type

classmethod `from_filename_parts`(*, *spk_object*: *str*, *version*: *str*, *utc_start*: *datetime*, *utc_end*: *datetime*, *revision*: *datetime*, *basepath*: *str* | *Path* | *S3Path* | *None* = *None*)

Create instance from filename parts.

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part arg names are named according to the regex for the file type.

Parameters

- **spk_object** (*str*) – Name of object whose attitude is represented in this SPK.
- **version** (*str*) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (*datetime.datetime*) – Start time of data.
- **utc_end** (*datetime.datetime*) – End time of data.
- **revision** (*datetime.datetime*) – When the file was last revised.
- **basepath** (*Optional[Union[str, Path, S3Path]]*) – Allows prepending a basepath or prefix.

Return type

EphemerisKernelFilename

property processing_step_id: *ProcessingStepIdentifier*

Property that contains the ProcessingStepIdentifier that generates this file

class libera_utils.io.filenaming.L0Filename(*args, **kwargs)

Filename validation class for L0 files from EDOS.

Attributes

archive_prefix

Property that contains the generated prefix for L0 archiving

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*id_char, scid, ...[, ...])	Create instance from filename parts
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

classmethod *_format_filename_parts*(*id_char: str, scid: int, first_apid: int, fill: str, created_time: datetime, numeric_id: int, file_number: int, extension: str, signal: str | None = None)

Construct a path from filename parts

Parameters

- **id_char** (*str*) – Either P (for PDS files, Construction Records) or X (for Delivery Records)
- **scid** (*int*) – Spacecraft ID
- **first_apid** (*int*) – First APID in the file
- **fill** (*str*) – Custom string up to 14 characters long
- **created_time** (*datetime.datetime*) – Creation time of the file
- **numeric_id** (*int*) – Data set ID, 0-9, one digit
- **file_number** (*str*) – File number within the data set. Construction records are always file number zero.
- **extension** (*str*) – File name extension. Either PDR or PDS

- **signal** (*Optional[str]*, *Optional*) – Optional signal suffix. Always ‘.XFR’

Returns

Formatted filename

Return type

str

_parse_filename_parts()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

types.SimpleNamespace

property archive_prefix: str

Property that contains the generated prefix for LO archiving

property data_product_id: DataProductIdentifier

Property that contains the DataProductIdentifier for this file type

classmethod from_filename_parts(**, id_char: str, scid: int, first_apid: int, fill: str, created_time: datetime, numeric_id: int, file_number: int, extension: str, signal: str | None = None, basepath: str | Path | S3Path | None = None*)

Create instance from filename parts

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part names are named according to the regex for the file type.

Parameters

- **id_char** (*str*) – Either P (for PDS files, Construction Records) or X (for Delivery Records)
- **scid** (*int*) – Spacecraft ID
- **first_apid** (*int*) – First APID in the file
- **fill** (*str*) – Custom string up to 14 characters long
- **created_time** (*datetime.datetime*) – Creation time of the file
- **numeric_id** (*int*) – Data set ID, 0-9, one digit
- **file_number** (*str*) – File number within the data set. Construction records are always file number zero.
- **extension** (*str*) – File name extension. Either PDR or PDS
- **signal** (*Optional[str]*) – Optional signal suffix. Always ‘.XFR’
- **basepath** (*Optional[Union[str, Path, S3Path]]*) – Allows prepending a basepath or prefix.

Return type

LOFilename

property processing_step_id: ProcessingStepIdentifier

Property that contains the ProcessingStepIdentifier that generates this file

class libera_utils.io.naming.LiberaDataProductFilename(*args, **kwargs)

Filename validation class for L1B and L2 science products

Attributes

archive_prefix

Property that contains the generated prefix for L1B and L2 archiving

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<i>from_file_path</i> (*args, **kwargs)	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<i>from_filename_parts</i> (*args, data_level, ..., ...)	Create instance from filename parts.
<i>generate_prefixed_path</i> (parent_path)	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<i>regex_match</i> (path)	Parse and validate a given path against class-attribute defined regex

classmethod *_format_filename_parts*(*args, data_level: str, product_name: str, version: str, utc_start: datetime, utc_end: datetime, revision: datetime, extension: str)

Construct a path from filename parts

Parameters

- **data_level** (*str*) – L1B or L2
- **product_name** (*str*) – Libera instrument, cam or rad for L1B and cloud-fraction etc. for L2. May contain anything except for underscores.
- **version** (*str*) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (*datetime.datetime*) – First timestamp in the SPK
- **utc_end** (*datetime.datetime*) – Last timestamp in the SPK
- **revision** (*datetime.datetime*) – Time when the file was created.
- **extension** (*str*) – File extension (.nc or .h5)

Returns

Formatted filename

Return type

str

_parse_filename_parts()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

`types.SimpleNamespace`

property archive_prefix: str

Property that contains the generated prefix for L1B and L2 archiving

property data_product_id: *DataProductIdentifier*

Property that contains the DataProductIdentifier for this file type

classmethod from_filename_parts(**, data_level: str, product_name: str, version: str, utc_start: datetime, utc_end: datetime, revision: datetime, extension: str = 'nc', basepath: str | Path | S3Path | None = None*)

Create instance from filename parts. All keyword arguments other than basepath are required!

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part names are named according to the regex for the file type.

Parameters

- **data_level** (*str*) – L1B or L2 identifying the level of the data product
- **product_name** (*str*) – Product type. e.g. cloud-fraction for L2 or cam for L1B. May contain anything except for underscores.
- **version** (*str*) – Software version that the file was created with. Corresponds to the algorithm version as determined by the algorithm software.
- **utc_start** (*datetime.datetime*) – First timestamp in the SPK
- **utc_end** (*datetime.datetime*) – Last timestamp in the SPK
- **revision** (*datetime.datetime*) – Time when the file was created.
- **extension** (*str*) – File extension (.nc or .h5)
- **basepath** (*Optional[Union[str, Path, S3Path]]*) – Allows prepending a basepath or prefix.

Return type

LiberaDataProductFilename

property processing_step_id: *ProcessingStepIdentifier*

Property that contains the ProcessingStepIdentifier that generates this file

class `libera_utils.io.filenameing.ManifestFilename`(**args, **kwargs*)

Class for naming manifest files

Attributes***archive_prefix***

Manifests are not archived like data products, but for convenience and ease of debugging they will be kept in the dropbox bucket by input/output and day they were made.

data_product_id

Property that contains the DataProductIdentifier for this file type

filename_parts

Property that contains a namespace of filename parts

path

Property containing the file path

processing_step_id

Property that contains the ProcessingStepIdentifier that generates this file

Methods

<code>from_file_path(*args, **kwargs)</code>	Factory method to produce an AbstractValidFilename from a valid Libera file path (str or Path)
<code>from_filename_parts(manifest_type, ulid_code)</code>	Create instance from filename parts.
<code>generate_prefixed_path(parent_path)</code>	Generates an absolute path of the form {parent_path}/{prefix_structure}/{file_basename} The parent_path can be an S3 bucket or an absolute local filepath (must start with /)
<code>regex_match(path)</code>	Parse and validate a given path against class-attribute defined regex

classmethod `_format_filename_parts`(*manifest_type*: ManifestType, *ulid_code*: ULID)

Construct a path from filename parts

Parameters

- **manifest_type** (ManifestType) – Input or output
- **ulid_code** (ulid.ULID) – ULID code for use in filename parts

Returns

Formatted filename

Return type

str

method `_parse_filename_parts`()

Parse the filename parts into objects from regex matched strings

Returns

namespace object containing filename parts as parsed objects

Return type

types.SimpleNamespace

property `archive_prefix`: str

Manifests are not archived like data products, but for convenience and ease of debugging they will be kept in the dropbox bucket by input/output and day they were made. This is used by the step function clean up function in the CDK. # Generate prefix structure # <manifest_type>/<year>/<month>/<day>

property `data_product_id`: DataProductIdentifier

Property that contains the DataProductIdentifier for this file type

classmethod `from_filename_parts`(*manifest_type*: ManifestType, *ulid_code*: ULID, *basepath*: str | Path | S3Path = None)

Create instance from filename parts.

This method exists primarily to expose typehinting to the user for use with the generic `_from_filename_parts`. The part names are named according to the regex for the file type.

Parameters

- **manifest_type** (*ManifestType*) – Input or output
- **ulid_code** (*ulid.ULID*) – ULID code for use in filename parts
- **basepath** (*Optional[Union[str, Path, S3Path]]*) – Allows prepending a basepath or prefix.

Return type*ManifestFilename***property processing_step_id:** *ProcessingStepIdentifier*

Property that contains the ProcessingStepIdentifier that generates this file

class `libera_utils.io.filenaming.ProductName`(*value*)

Enum of valid product names as used in filenames, defined and sourced from the LASP-ASDC ICD

Methods

<i>capitalize()</i>	Return a capitalized version of the string.
<i>casefold()</i>	Return a version of the string suitable for caseless comparisons.
<i>center</i> (width[, fillchar])	Return a centered string of length width.
<i>count</i> (sub[, start[, end]])	Return the number of non-overlapping occurrences of substring sub in string S[start:end].
<i>encode</i> ([, encoding, errors])	Encode the string using the codec registered for encoding.
<i>endswith</i> (suffix[, start[, end]])	Return True if S ends with the specified suffix, False otherwise.
<i>expandtabs</i> ([, tabsize])	Return a copy where all tab characters are expanded using spaces.
<i>find</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>format</i> (*args, **kwargs)	Return a formatted version of S, using substitutions from args and kwargs.
<i>format_map</i> (mapping)	Return a formatted version of S, using substitutions from mapping.
<i>index</i> (sub[, start[, end]])	Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end].
<i>isalnum()</i>	Return True if the string is an alpha-numeric string, False otherwise.
<i>isalpha()</i>	Return True if the string is an alphabetic string, False otherwise.
<i>isascii()</i>	Return True if all characters in the string are ASCII, False otherwise.
<i>isdecimal()</i>	Return True if the string is a decimal string, False otherwise.
<i>isdigit()</i>	Return True if the string is a digit string, False otherwise.
<i>isidentifier()</i>	Return True if the string is a valid Python identifier, False otherwise.
<i>islower()</i>	Return True if the string is a lowercase string, False otherwise.
<i>isnumeric()</i>	Return True if the string is a numeric string, False otherwise.

continues on next page

Table 79 – continued from previous page

<code>isprintable()</code>	Return True if the string is printable, False otherwise.
<code>isspace()</code>	Return True if the string is a whitespace string, False otherwise.
<code>istitle()</code>	Return True if the string is a title-cased string, False otherwise.
<code>isupper()</code>	Return True if the string is an uppercase string, False otherwise.
<code>join(iterable, /)</code>	Concatenate any number of strings.
<code>ljust(width[, fillchar])</code>	Return a left-justified string of length width.
<code>lower()</code>	Return a copy of the string converted to lowercase.
<code>lstrip([chars])</code>	Return a copy of the string with leading whitespace removed.
<code>maketrans(x[, y, z])</code>	Return a translation table usable for <code>str.translate()</code> .
<code>partition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>removeprefix(prefix, /)</code>	Return a str with the given prefix string removed if present.
<code>removesuffix(suffix, /)</code>	Return a str with the given suffix string removed if present.
<code>replace(old, new[, count])</code>	Return a copy with all occurrences of substring <code>old</code> replaced by <code>new</code> .
<code>rfind(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rindex(sub[, start[, end]])</code>	Return the highest index in <code>S</code> where substring <code>sub</code> is found, such that <code>sub</code> is contained within <code>S[start:end]</code> .
<code>rjust(width[, fillchar])</code>	Return a right-justified string of length width.
<code>rpartition(sep, /)</code>	Partition the string into three parts using the given separator.
<code>rsplit(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>rstrip([chars])</code>	Return a copy of the string with trailing whitespace removed.
<code>split(/[, sep, maxsplit])</code>	Return a list of the substrings in the string, using <code>sep</code> as the separator string.
<code>splitlines(/[, keepends])</code>	Return a list of the lines in the string, breaking at line boundaries.
<code>startswith(prefix[, start[, end]])</code>	Return True if <code>S</code> starts with the specified prefix, False otherwise.
<code>strip([chars])</code>	Return a copy of the string with leading and trailing whitespace removed.
<code>swapcase()</code>	Convert uppercase characters to lowercase and lowercase characters to uppercase.
<code>title()</code>	Return a version of the string where each word is titlecased.
<code>translate(table, /)</code>	Replace each character in the string using the given translation table.
<code>upper()</code>	Return a copy of the string converted to uppercase.
<code>zfill(width, /)</code>	Pad a numeric string with zeros on the left, to fill a field of the given width.

property `data_product_id`: [DataProductIdentifier](#)

DataProductIdentifier for this product name

property processing_step_id: *ProcessingStepIdentifier*

ProcessingStepIdentifier for this product name

`libera_utils.io.filenaming.format_semantic_version(semantic_version: str) → str`

Formats a semantic version string X.Y.Z into a filename-compatible string like VX-Y-Z, for X = major version, Y = minor version, Z = patch.

Result is uppercase. Release candidate suffixes are allowed as no strict checking is done on the contents of X, Y, or Z. e.g. 1.2.3rc1 becomes V1-2-3RC1

Parameters

semantic_version (*str*) – String matching X.Y.Z where X, Y and Z are integers of any length

Return type

str

`libera_utils.io.filenaming.get_current_revision_str() → str`

Get the current *r%y%j%H%M%S* string for filename revisions.

Returns

Current (now) revision string.

Return type

str

`libera_utils.io.filenaming.get_current_version_str(package_name: str) → str`

Retrieve the current version of a (algorithm) package and format it for inclusion in a filename

Parameters

package_name (*str*) – Package for which to retrieve a version string. This should be your algorithm package and it must use a semantic versioning scheme, configured in project metadata.

Returns

Version string in format vM1m2p3

Return type

str

libera_utils.io.hdf

Utils for HDF5 file handling

Functions

<code>h5dump(f[, include_attrs, stdout])</code>	Prints the contents of an HDF5 object.
---	--

libera_utils.io.hdf.h5dump

`libera_utils.io.hdf.h5dump(f: File, include_attrs: bool = True, stdout: bool = False)`

Prints the contents of an HDF5 object.

Parameters

- **f** (*h5py.File* or *h5py.Group*) – File, Group object from which to start inspecting.
- **include_attrs** (*bool*, *Optional*) – Default True.
- **stdout** (*bool*, *Optional*) – Default False. If True, prints to stdout as the object tree is traversed.

Returns

Concatenated string of HDF5 contents

Return type

str

`libera_utils.io.hdf.h5dump(f: File, include_attrs: bool = True, stdout: bool = False)`

Prints the contents of an HDF5 object.

Parameters

- **f** (*h5py.File* or *h5py.Group*) – File, Group object from which to start inspecting.
- **include_attrs** (*bool*, *Optional*) – Default True.
- **stdout** (*bool*, *Optional*) – Default False. If True, prints to stdout as the object tree is traversed.

Returns

Concatenated string of HDF5 contents

Return type

str

libera_utils.io.manifest

Module for manifest file handling

Functions

<code>calculate_checksum(file)</code>	Compute the checksum of the given file.
<code>get_ulid_code(filename)</code>	Get ULID code from filename.

libera_utils.io.manifest.calculate_checksum

`libera_utils.io.manifest.calculate_checksum(file: str | Path | S3Path) → str`

Compute the checksum of the given file.

libera_utils.io.manifest.get_ulid_code

`libera_utils.io.manifest.get_ulid_code(filename: str | Path | S3Path | ManifestFilename | None) → ULID | None`

Get ULID code from filename.

Classes

<code>Manifest(*, manifest_type, files, ...)</code>	Pydantic model for a manifest file.
<code>ManifestFileRecord(*, filename, checksum)</code>	Pydantic model for an individual data product file recorded within a manifest file.

libera_utils.io.manifest.Manifest

```
class libera_utils.io.manifest.Manifest(* , manifest_type: ~libera_utils.aws.constants.ManifestType,
                                         files: ~typing.List[~libera_utils.io.manifest.ManifestFileRecord]
                                         = <factory>, configuration: ~typing.Dict[str, ~typing.Any] =
                                         <factory>, filename:
                                         ~libera_utils.io.filenaming.ManifestFilename | None = None,
                                         ulid_code: ~ulid.ULID | None = <factory>)
```

Bases: BaseModel

Pydantic model for a manifest file.

Attributes

model_extra

Get extra fields set during validation.

model_fields_set

Returns the set of fields that have been explicitly set on this model instance.

Methods

<i>add_desired_time_range</i> (start_datetime, ...)	Add a time range to the configuration section of the manifest.
<i>add_files</i> (*files)	Add files to the manifest from filename
<i>check_file_structure</i> (file_structure, ...)	Check file structure, returning True if it is good.
<i>copy</i> (*[, include, exclude, update, deep])	Returns a copy of the model.
<i>from_file</i> (filepath)	Read a manifest file and return a Manifest object (factory method).
<i>model_construct</i> ([_fields_set])	Creates a new instance of the <i>Model</i> class with validated data.
<i>model_copy</i> (*[, update, deep])	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#model_copy
<i>model_dump</i> (*[, mode, include, exclude, ...])	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#modelmodel_dump
<i>model_dump_json</i> (*[, indent, include, ...])	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#modelmodel_dump_json
<i>model_json_schema</i> ([by_alias, ref_template, ...])	Generates a JSON schema for a model class.
<i>model_parametrized_name</i> (params)	Compute the class name for parametrizations of generic classes.
<i>model_post_init</i> (_BaseModel__context)	Override this method to perform additional initialization after <i>__init__</i> and <i>model_construct</i> .
<i>model_rebuild</i> (*[, force, raise_errors, ...])	Try to rebuild the pydantic-core schema for the model.
<i>model_validate</i> (obj, *[, strict, ...])	Validate a pydantic model instance.
<i>model_validate_json</i> (json_data, *[, strict, ...])	Usage docs: https://docs.pydantic.dev/2.10/concepts/json/#json-parsing
<i>model_validate_strings</i> (obj, *[, strict, context])	Validate the given object with string data against the Pydantic model.
<i>output_manifest_from_input_manifest</i> (...)	Create Output manifest from input manifest file path, adds input files to output manifest configuration
<i>serialize_filename</i> (filename, _info)	Custom serializer for the manifest filename.
<i>transform_filename</i> (raw_filename)	Convert raw filename to ManifestFilename class if necessary.
<i>transform_files</i> (raw_list)	Allow for the incoming files list to have varying types.
<i>validate_checksums</i> ()	Validate checksums of listed files

continues on next page

Table 83 – continued from previous page

<code>write(out_path[, filename])</code>	Write a manifest file from a Manifest object (self).
--	--

<code>construct</code>
<code>dict</code>
<code>from_orm</code>
<code>json</code>
<code>parse_file</code>
<code>parse_obj</code>
<code>parse_raw</code>
<code>schema</code>
<code>schema_json</code>
<code>update_forward_refs</code>
<code>validate</code>

`__init__(**data: Any) → None`

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Methods

<code>add_desired_time_range(start_datetime, ...)</code>	Add a time range to the configuration section of the manifest.
<code>add_files(*files)</code>	Add files to the manifest from filename
<code>check_file_structure(file_structure, ...)</code>	Check file structure, returning True if it is good.
<code>from_file(filepath)</code>	Read a manifest file and return a Manifest object (factory method).
<code>output_manifest_from_input_manifest(...)</code>	Create Output manifest from input manifest file path, adds input files to output manifest configuration
<code>serialize_filename(filename, _info)</code>	Custom serializer for the manifest filename.
<code>transform_filename(raw_filename)</code>	Convert raw filename to ManifestFilename class if necessary.
<code>transform_files(raw_list)</code>	Allow for the incoming files list to have varying types.
<code>validate_checksums()</code>	Validate checksums of listed files
<code>write(out_path[, filename])</code>	Write a manifest file from a Manifest object (self).

Attributes

<code>model_computed_fields</code>	
<code>model_config</code>	Configuration for the model, should be a dictionary conforming to <code>[ConfigDict][pydantic.config.ConfigDict]</code> .
<code>model_extra</code>	Get extra fields set during validation.

continues on next page

Table 85 – continued from previous page

model_fields	
<i>model_fields_set</i>	Returns the set of fields that have been explicitly set on this model instance.
manifest_type	
files	
configuration	
filename	
ulid_code	

_generate_filename() → *ManifestFilename*

Generate a valid manifest filename

add_desired_time_range(*start_datetime: datetime, end_datetime: datetime*)

Add a time range to the configuration section of the manifest.

Parameters

- **start_datetime** (*datetime.datetime*) – The desired start time for the range of data in this manifest
- **end_datetime** (*datetime.datetime*) – The desired end time for the range of data in this manifest

Return type

None

add_files(**files: str | Path | S3Path*)

Add files to the manifest from filename

Parameters

files (*Union[str, Path, S3Path]*) – Path to the file to add to the manifest.

Return type

None

classmethod check_file_structure(*file_structure: ManifestFileRecord, existing_names: set[str], existing_checksums: set[str]*) → bool

Check file structure, returning True if it is good.

classmethod from_file(*filepath: str | Path | S3Path*)

Read a manifest file and return a Manifest object (factory method).

Parameters

filepath (*Union[str, Path, S3Path]*) – Location of manifest file to read.

Returns

Pydantic model built from the json of the given manifest file.

Return type

Manifest

model_config: `ClassVar[ConfigDict] = {'arbitrary_types_allowed': True}`

Configuration for the model, should be a dictionary conforming to `[ConfigDict][pydantic.config.ConfigDict]`.

property model_extra: `dict[str, Any] | None`

Get extra fields set during validation.

Returns

A dictionary of extra fields, or *None* if `config.extra` is not set to “allow”.

property model_fields_set: `set[str]`

Returns the set of fields that have been explicitly set on this model instance.

Returns

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

classmethod output_manifest_from_input_manifest(`input_manifest: Path | S3Path | Manifest`) → `Manifest`

Create Output manifest from input manifest file path, adds input files to output manifest configuration

Parameters

input_manifest (`Union[Path, S3Path, 'Manifest']`) – An S3 or regular path to an input_manifest object, or the input manifest object itself

Returns

output_manifest – The newly created output manifest

Return type

`Manifest`

serialize_filename(`filename: str | Path | S3Path | ManifestFilename | None, _info`) → `str`

Custom serializer for the manifest filename.

classmethod transform_filename(`raw_filename: str | ManifestFilename | None`) → `ManifestFilename | None`

Convert raw filename to ManifestFilename class if necessary.

classmethod transform_files(`raw_list: list[dict | str | Path | S3Path | ManifestFileRecord] | None`) → `list[ManifestFileRecord]`

Allow for the incoming files list to have varying types. Convert to a standardized list of ManifestFileStructure.

validate_checksums() → `None`

Validate checksums of listed files

write(`out_path: str | Path | S3Path, filename: str = None`) → `Path | S3Path`

Write a manifest file from a Manifest object (self).

Parameters

- **out_path** (`Union[str, Path, S3Path]`) – Directory path to write to (directory being used loosely to refer also to an S3 bucket path).
- **filename** (`str, Optional`) – must be a valid manifest filename. If not provided, the method uses the objects internal filename attribute. If that is not set, then a filename is automatically generated.

Returns

The path where the manifest file is written.

Return type

Union[Path, S3Path]

libera_utils.io.manifest.ManifestFileRecord**class** libera_utils.io.manifest.**ManifestFileRecord**(**, filename: str, checksum: str*)

Bases: BaseModel

Pydantic model for an individual data product file recorded within a manifest file.

Attributes***model_extra***

Get extra fields set during validation.

model_fields_set

Returns the set of fields that have been explicitly set on this model instance.

Methods

<code>copy(*[, include, exclude, update, deep])</code>	Returns a copy of the model.
<code>model_construct([_fields_set])</code>	Creates a new instance of the <i>Model</i> class with validated data.
<code>model_copy(*[, update, deep])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#model_copy
<code>model_dump(*[, mode, include, exclude, ...])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#modelmodel_dump
<code>model_dump_json(*[, indent, include, ...])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#modelmodel_dump_json
<code>model_json_schema([by_alias, ref_template, ...])</code>	Generates a JSON schema for a model class.
<code>model_parametrized_name(params)</code>	Compute the class name for parametrizations of generic classes.
<code>model_post_init(_BaseModel__context)</code>	Override this method to perform additional initialization after <code>__init__</code> and <code>model_construct</code> .
<code>model_rebuild(*[, force, raise_errors, ...])</code>	Try to rebuild the pydantic-core schema for the model.
<code>model_validate(obj, *[, strict, ...])</code>	Validate a pydantic model instance.
<code>model_validate_json(json_data, *[, strict, ...])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/json/#json-parsing
<code>model_validate_strings(obj, *[, strict, context])</code>	Validate the given object with string data against the Pydantic model.

construct
dict
from_orm
json
parse_file
parse_obj
parse_raw
schema
schema_json
update_forward_refs
validate

`__init__(**data: Any) → None`

Create a new model by parsing and validating input data from keyword arguments.

Raises `[ValidationError][pydantic_core.ValidationError]` if the input data cannot be validated to form a valid model.

`self` is explicitly positional-only to allow `self` as a field name.

Attributes

<code>model_computed_fields</code>	
<code>model_config</code>	Configuration for the model, should be a dictionary conforming to <code>[ConfigDict][pydantic.config.ConfigDict]</code> .
<code>model_extra</code>	Get extra fields set during validation.
<code>model_fields</code>	
<code>model_fields_set</code>	Returns the set of fields that have been explicitly set on this model instance.
<code>filename</code>	
<code>checksum</code>	

`model_config: ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to `[ConfigDict][pydantic.config.ConfigDict]`.

`property model_extra: dict[str, Any] | None`

Get extra fields set during validation.

Returns

A dictionary of extra fields, or `None` if `config.extra` is not set to “`allow`”.

`property model_fields_set: set[str]`

Returns the set of fields that have been explicitly set on this model instance.

Returns

A set of strings representing the fields that have been set, i.e. that were not filled from defaults.

Exceptions

<code>ManifestError</code>	Generic exception related to manifest file handling
----------------------------	---

`libera_utils.io.manifest.ManifestError`

exception `libera_utils.io.manifest.ManifestError`

Generic exception related to manifest file handling

```
class libera_utils.io.manifest.Manifest(*, manifest_type: ~libera_utils.aws.constants.ManifestType,
    files: ~typing.List[~libera_utils.io.manifest.ManifestFileRecord]
    = <factory>, configuration: ~typing.Dict[str, ~typing.Any] =
    <factory>, filename:
    ~libera_utils.io.naming.ManifestFilename | None = None,
    ulid_code: ~ulid.ULID | None = <factory>)
```

Pydantic model for a manifest file.

Attributes

`model_extra`

Get extra fields set during validation.

`model_fields_set`

Returns the set of fields that have been explicitly set on this model instance.

Methods

<code>add_desired_time_range(start_datetime, ...)</code>	Add a time range to the configuration section of the manifest.
<code>add_files(*files)</code>	Add files to the manifest from filename
<code>check_file_structure(file_structure, ...)</code>	Check file structure, returning True if it is good.
<code>copy(*[, include, exclude, update, deep])</code>	Returns a copy of the model.
<code>from_file(filepath)</code>	Read a manifest file and return a Manifest object (factory method).
<code>model_construct([_fields_set])</code>	Creates a new instance of the <i>Model</i> class with validated data.
<code>model_copy(*[, update, deep])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#model_copy
<code>model_dump(*[, mode, include, exclude, ...])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#modelmodel_dump
<code>model_dump_json(*[, indent, include, ...])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#modelmodel_dump_json
<code>model_json_schema([by_alias, ref_template, ...])</code>	Generates a JSON schema for a model class.
<code>model_parametrized_name(params)</code>	Compute the class name for parametrizations of generic classes.
<code>model_post_init(_BaseModel__context)</code>	Override this method to perform additional initialization after <code>__init__</code> and <code>model_construct</code> .
<code>model_rebuild(*[, force, raise_errors, ...])</code>	Try to rebuild the pydantic-core schema for the model.
<code>model_validate(obj, *[, strict, ...])</code>	Validate a pydantic model instance.
<code>model_validate_json(json_data, *[, strict, ...])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/json/#json-parsing
<code>model_validate_strings(obj, *[, strict, context])</code>	Validate the given object with string data against the Pydantic model.
<code>output_manifest_from_input_manifest(...)</code>	Create Output manifest from input manifest file path, adds input files to output manifest configuration
<code>serialize_filename(filename, _info)</code>	Custom serializer for the manifest filename.
<code>transform_filename(raw_filename)</code>	Convert raw filename to ManifestFilename class if necessary.
<code>transform_files(raw_list)</code>	Allow for the incoming files list to have varying types.
<code>validate_checksums()</code>	Validate checksums of listed files
<code>write(out_path[, filename])</code>	Write a manifest file from a Manifest object (self).

construct
dict
from_orm
json
parse_file
parse_obj
parse_raw
schema
schema_json
update_forward_refs
validate

_generate_filename() → *ManifestFilename*

Generate a valid manifest filename

add_desired_time_range(*start_datetime: datetime, end_datetime: datetime*)

Add a time range to the configuration section of the manifest.

Parameters

- **start_datetime** (*datetime.datetime*) – The desired start time for the range of data in this manifest
- **end_datetime** (*datetime.datetime*) – The desired end time for the range of data in this manifest

Return type

None

add_files(**files: str | Path | S3Path*)

Add files to the manifest from filename

Parameters

files (*Union[str, Path, S3Path]*) – Path to the file to add to the manifest.

Return type

None

classmethod check_file_structure(*file_structure: ManifestFileRecord, existing_names: set[str], existing_checksums: set[str]*) → bool

Check file structure, returning True if it is good.

classmethod from_file(*filepath: str | Path | S3Path*)

Read a manifest file and return a Manifest object (factory method).

Parameters

filepath (*Union[str, Path, S3Path]*) – Location of manifest file to read.

Returns

Pydantic model built from the json of the given manifest file.

Return type

Manifest

model_config: `ClassVar[ConfigDict] = {'arbitrary_types_allowed': True}`

Configuration for the model, should be a dictionary conforming to `[ConfigDict][pydantic.config.ConfigDict]`.

classmethod `output_manifest_from_input_manifest(input_manifest: Path | S3Path | Manifest) → Manifest`

Create Output manifest from input manifest file path, adds input files to output manifest configuration

Parameters

input_manifest (`Union[Path, S3Path, 'Manifest']`) – An S3 or regular path to an input_manifest object, or the input manifest object itself

Returns

output_manifest – The newly created output manifest

Return type

`Manifest`

serialize_filename(`filename: str | Path | S3Path | ManifestFilename | None, _info`) → `str`

Custom serializer for the manifest filename.

classmethod `transform_filename(raw_filename: str | ManifestFilename | None) → ManifestFilename | None`

Convert raw filename to ManifestFilename class if necessary.

classmethod `transform_files(raw_list: list[dict | str | Path | S3Path | ManifestFileRecord] | None) → list[ManifestFileRecord]`

Allow for the incoming files list to have varying types. Convert to a standardized list of ManifestFileStructure.

validate_checksums() → `None`

Validate checksums of listed files

write(`out_path: str | Path | S3Path, filename: str = None`) → `Path | S3Path`

Write a manifest file from a Manifest object (self).

Parameters

- **out_path** (`Union[str, Path, S3Path]`) – Directory path to write to (directory being used loosely to refer also to an S3 bucket path).
- **filename** (`str, Optional`) – must be a valid manifest filename. If not provided, the method uses the objects internal filename attribute. If that is not set, then a filename is automatically generated.

Returns

The path where the manifest file is written.

Return type

`Union[Path, S3Path]`

exception `libera_utils.io.manifest.ManifestError`

Generic exception related to manifest file handling

class `libera_utils.io.manifest.ManifestFileRecord(*, filename: str, checksum: str)`

Pydantic model for an individual data product file recorded within a manifest file.

Attributes

`model_extra`

Get extra fields set during validation.

model_fields_set

Returns the set of fields that have been explicitly set on this model instance.

Methods

<code>copy(*[, include, exclude, update, deep])</code>	Returns a copy of the model.
<code>model_construct([_fields_set])</code>	Creates a new instance of the <i>Model</i> class with validated data.
<code>model_copy(*[, update, deep])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#model_copy
<code>model_dump(*[, mode, include, exclude, ...])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#modelmodel_dump
<code>model_dump_json(*[, indent, include, ...])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/serialization/#modelmodel_dump_json
<code>model_json_schema([by_alias, ref_template, ...])</code>	Generates a JSON schema for a model class.
<code>model_parametrized_name(params)</code>	Compute the class name for parametrizations of generic classes.
<code>model_post_init(_BaseModel__context)</code>	Override this method to perform additional initialization after <code>__init__</code> and <code>model_construct</code> .
<code>model_rebuild(*[, force, raise_errors, ...])</code>	Try to rebuild the pydantic-core schema for the model.
<code>model_validate(obj, *[, strict, ...])</code>	Validate a pydantic model instance.
<code>model_validate_json(json_data, *[, strict, ...])</code>	Usage docs: https://docs.pydantic.dev/2.10/concepts/json/#json-parsing
<code>model_validate_strings(obj, *[, strict, context])</code>	Validate the given object with string data against the Pydantic model.

construct
dict
from_orm
json
parse_file
parse_obj
parse_raw
schema
schema_json
update_forward_refs
validate

model_config: `ClassVar[ConfigDict] = {}`

Configuration for the model, should be a dictionary conforming to `[ConfigDict][pydantic.config.ConfigDict]`.

`libera_utils.io.manifest.calculate_checksum(file: str | Path | S3Path) → str`

Compute the checksum of the given file.

`libera_utils.io.manifest.get_ulid_code(filename: str | Path | S3Path | ManifestFilename | None) → ULID | None`

Get ULID code from filename.

libera_utils.io.smart_open

Module for smart_open

Functions

<code>is_gzip(path)</code>	Determine if a string points to a gzip file.
<code>is_s3(path)</code>	Determine if a string points to an s3 location or not.
<code>smart_copy_file(source_path, dest_path[, delete])</code>	Copy function that can handle local files or files in an S3 bucket.
<code>smart_open(path[, mode, enable_gzip])</code>	Open function that can handle local files or files in an S3 bucket.

libera_utils.io.smart_open.is_gzip

`libera_utils.io.smart_open.is_gzip(path: str | Path | S3Path)`

Determine if a string points to a gzip file.

Parameters

path (`Union[str, Path, S3Path]`) – Path to check.

Return type

`bool`

libera_utils.io.smart_open.is_s3

`libera_utils.io.smart_open.is_s3(path: str | Path | S3Path)`

Determine if a string points to an s3 location or not.

Parameters

path (`Union[str, Path, S3Path]`) – Path to determine if it is and s3 location or not.

Return type

`bool`

libera_utils.io.smart_open.smart_copy_file

`libera_utils.io.smart_open.smart_copy_file(source_path: str | Path | S3Path, dest_path: str | Path | S3Path, delete: bool | None = False)`

Copy function that can handle local files or files in an S3 bucket. Returns the path to the newly created file as a Path or an S3Path, depending on the destination.

Parameters

- **source_path** (`Union[str, Path, S3Path]`) – Path to the source file to be copied. Files residing in an s3 bucket must begin with “s3://”.
- **dest_path** (`Union[str, Path, S3Path]`) – Path to the Destination file to be copied to. Files residing in an s3 bucket must begin with “s3://”.
- **delete** (`bool`, *optional*) – If true, deletes files copied from source (default = False)

Returns

The path to the newly created file

Return type

Path or S3Path

libera_utils.io.smart_open.smart_open

`libera_utils.io.smart_open.smart_open(path: str | Path | S3Path, mode: str | None = 'rb', enable_gzip: bool | None = True)`

Open function that can handle local files or files in an S3 bucket. It also correctly handles gzip files determined by a *.gz extension.

Parameters

- **path** (*Union[str, Path, S3Path]*) – Path to the file to be opened. Files residing in an s3 bucket must begin with “s3://”.
- **mode** (*str, Optional*) – Optional string specifying the mode in which the file is opened. Defaults to ‘rb’.
- **enable_gzip** (*bool, Optional*) – Flag to specify that *.gz files should be opened as a *GzipFile* object. Setting this to False is useful when creating the md5sum of a *.gz file. Defaults to True.

Return type

IO or *gzip.GzipFile*

`libera_utils.io.smart_open._copy_local_to_local(source_path: str | Path, dest_path: str | Path, delete: bool | None = False)`

Copy a local source file to a local destination.

Parameters

- **source_path** (*Union[str, Path]*) – Path to the source file to be copied.
- **dest_path** (*Union[str, Path]*) – Path to the destination for the copied file.
- **delete** (*bool, Optional*) – If true, deletes files copied from source (default = False)

Returns

The path to the newly created file

Return type

Path

`libera_utils.io.smart_open._copy_local_to_s3(source_path: str | Path, dest_path: str | S3Path, delete: bool | None = False)`

Copy a local file to an S3 object.

Parameters

- **source_path** (*Union[str, Path]*) – Path to the source file to be copied.
- **dest_path** (*Union[str, S3Path]*) – Path to the destination for the copied file. Files residing in an s3 bucket must begin with “s3://”.
- **delete** (*bool, optional*) – If true, deletes files copied from source (default = False)

Returns

The path to the newly created file

Return type

S3Path

`libera_utils.io.smart_open._copy_s3_to_local(source_path: str | S3Path, dest_path: str | Path, delete: bool | None = False)`

Copy an S3 object to a local file.

Parameters

- **source_path** (*Union[str, S3Path]*) – Path to the source file to be copied. Files residing in an s3 bucket must begin with “s3://”.
- **dest_path** (*Union[str, Path]*) – Path to the destination for the copied file.
- **delete** (*bool, optional*) – If true, deletes files copied from source (default = False)

Returns

The path to the newly created file

Return type

Path

```
libera_utils.io.smart_open._copy_s3_to_s3(source_path: str | S3Path, dest_path: str | S3Path, delete:
                                         bool | None = False)
```

Copy an S3 object to a different S3 object.

Parameters

- **source_path** (*Union[str, S3Path]*) – Path to the source file to be copied. Files residing in an s3 bucket must begin with “s3://”.
- **dest_path** (*Union[str, S3Path]*) – Path to the Destination file to be copied to. Files residing in an s3 bucket must begin with “s3://”.
- **delete** (*bool, optional*) – If true, deletes files copied from source (default = False)

Returns

The path to the newly created file

Return type

S3Path

```
libera_utils.io.smart_open.is_gzip(path: str | Path | S3Path)
```

Determine if a string points to a gzip file.

Parameters

path (*Union[str, Path, S3Path]*) – Path to check.

Return type

bool

```
libera_utils.io.smart_open.is_s3(path: str | Path | S3Path)
```

Determine if a string points to an s3 location or not.

Parameters

path (*Union[str, Path, S3Path]*) – Path to determine if it is and s3 location or not.

Return type

bool

```
libera_utils.io.smart_open.smart_copy_file(source_path: str | Path | S3Path, dest_path: str | Path |
                                           S3Path, delete: bool | None = False)
```

Copy function that can handle local files or files in an S3 bucket. Returns the path to the newly created file as a Path or an S3Path, depending on the destination.

Parameters

- **source_path** (*Union[str, Path, S3Path]*) – Path to the source file to be copied. Files residing in an s3 bucket must begin with “s3://”.

- **dest_path** (*Union[str, Path, S3Path]*) – Path to the Destination file to be copied to. Files residing in an s3 bucket must begin with “s3://”.
- **delete** (*bool, optional*) – If true, deletes files copied from source (default = False)

Returns

The path to the newly created file

Return type

Path or S3Path

`libera_utils.io.smart_open.smart_open(path: str | Path | S3Path, mode: str | None = 'rb', enable_gzip: bool | None = True)`

Open function that can handle local files or files in an S3 bucket. It also correctly handles gzip files determined by a *.gz extension.

Parameters

- **path** (*Union[str, Path, S3Path]*) – Path to the file to be opened. Files residing in an s3 bucket must begin with “s3://”.
- **mode** (*str, Optional*) – Optional string specifying the mode in which the file is opened. Defaults to ‘rb’.
- **enable_gzip** (*bool, Optional*) – Flag to specify that *.gz files should be opened as a *GzipFile* object. Setting this to False is useful when creating the md5sum of a *.gz file. Defaults to True.

Return type

IO or *gzip.GzipFile*

3.1.8 libera_utils.kernel_maker

Module containing CLI tool for creating SPICE kernels from packets

Functions

<code>get_spice_packet_data_from_filepaths(...)</code>	Utility function to return an array of packet data from a list of file paths of raw JPSS APID 11 geolocation packet data files.
<code>make_azel_ck(parsed_args)</code>	Create a Libera Az-El CK from CCSDS packets or ASCII input files The C-kernel (CK) is the component of SPICE concerned with attitude of spacecraft structures or instruments.
<code>make_jpss_ck(parsed_args)</code>	Create a JPSS CK from APID 11 CCSDS packets.
<code>make_jpss_kernels_from_manifest(...)</code>	Alpha function triggering kernel generation from manifest file.
<code>make_jpss_spk(parsed_args)</code>	Create a JPSS SPK from APID 11 CCSDS packets.
<code>write_kernel_input_file(data, filepath[, ...])</code>	Write ephemeris and attitude data to MKSPK and MSOPCK input data files, respectively.
<code>write_kernel_setup_file(data, filepath)</code>	Write an MSOPCK or MKSPK compatible setup file of key-value pairs.

libera_utils.kernel_maker.get_spice_packet_data_from_filepaths

`libera_utils.kernel_maker.get_spice_packet_data_from_filepaths(packet_data_filepaths)`

Utility function to return an array of packet data from a list of file paths of raw JPSS APID 11 geolocation packet data files.

Parameters**packet_data_filepaths**

[list] The list of file paths to the raw packet data

:returns: **packet_data – The configured packet data. See `packets.py` for more details on structure**

:rtype: numpy.ndarray

libera_utils.kernel_maker.make_azel_ck

`libera_utils.kernel_maker.make_azel_ck(parsed_args: Namespace)`

Create a Libera Az-El CK from CCSDS packets or ASCII input files. The C-kernel (CK) is the component of SPICE concerned with attitude of spacecraft structures or instruments.

Parameters

parsed_args (*argparse.Namespace*) – Namespace of parsed CLI arguments

Return type

None

libera_utils.kernel_maker.make_jpss_ck

`libera_utils.kernel_maker.make_jpss_ck(parsed_args: Namespace)`

Create a JPSS CK from APID 11 CCSDS packets. The C-kernel (CK) is the component of SPICE concerned with attitude of spacecraft structures or instruments.

Parameters

parsed_args (*argparse.Namespace*) – Namespace of parsed CLI arguments

Return type

None

libera_utils.kernel_maker.make_jpss_kernels_from_manifest

`libera_utils.kernel_maker.make_jpss_kernels_from_manifest(manifest_file_path: str, output_directory: str)`

Alpha function triggering kernel generation from manifest file.

If the manifest configuration field contains “start_time” and “end_time” fields then this function will select only packet data that falls in that range. If these are not given, then all packet data will be used.

Parameters

- **manifest_file_path** (*str* or *cloudpathlib.any_path.AnyPath*) – Path to the manifest file that includes end_time and start_time in the configuration section
- **output_directory** (*str* or *cloudpathlib.any_path.AnyPath*) – Path to save the completed kernels

Returns

output_directory – Path to the directory containing the completed kernels

Return type

str or cloudpathlib.anypath.AnyPath

libera_utils.kernel_maker.make_jpss_spk

libera_utils.kernel_maker.**make_jpss_spk**(*parsed_args: Namespace*)

Create a JPSS SPK from APID 11 CCSDS packets. The SPK system is the component of SPICE concerned with ephemeris data (position/velocity).

Parameters

parsed_args (*argparse.Namespace*) – Namespace of parsed CLI arguments

Return type

None

libera_utils.kernel_maker.write_kernel_input_file

libera_utils.kernel_maker.**write_kernel_input_file**(*data: ndarray, filepath: str, fields: list = None, fmt: str = '%.16f'*)

Write ephemeris and attitude data to MKSPK and MSOPCK input data files, respectively.

See MSOPCK documentation here:

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/ug/msopck.html

See MKSPK documentation here:

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/ug/mkspk.html

Parameters

- **data** (*numpy.ndarray*) – Structured array (named, with data types) of attitude or ephemeris data.
- **filepath** (*str* or *pathlib.Path*) – Filepath to write to.
- **fields** (*list*) – Optional. List of field names to write out to the data file. If not specified, assume fields are already in the proper order.
- **fmt** (*str* or *list*) – Format specifier(s) to pass to np.savetxt. Default is to assume everything should be floats with 16 decimal places of precision (%.16f). If a list is passed, it must contain a format specifier for each column in data.

Returns

Absolute path to written file.

Return type

pathlib.Path

libera_utils.kernel_maker.write_kernel_setup_file

libera_utils.kernel_maker.**write_kernel_setup_file**(*data: dict, filepath: Path*)

Write an MSOPCK or MKSPK compatible setup file of key-value pairs. See documentation here: https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/ug/msopck.html#Input%20Data%20Format

Parameters

- **data** (*dict*) – Dictionary of key-value pairs to write to the setup file.
- **filepath** (*pathlib.Path*) – Filepath to write to.

Returns

Absolute path to written file.

Return type`pathlib.Path``libera_utils.kernel_maker.get_spice_packet_data_from_filepaths(packet_data_filepaths)`

Utility function to return an array of packet data from a list of file paths of raw JPSS APID 11 geolocation packet data files.

Parameters**packet_data_filepaths**

[list] The list of file paths to the raw packet data

:returns: **packet_data – The configured packet data. See packets.py for more details on structure**

:rtype: numpy.ndarray

`libera_utils.kernel_maker.make_azel_ck(parsed_args: Namespace)`

Create a Libera Az-El CK from CCSDS packets or ASCII input files The C-kernel (CK) is the component of SPICE concerned with attitude of spacecraft structures or instruments.

Parameters

parsed_args (`argparse.Namespace`) – Namespace of parsed CLI arguments

Return type

None

`libera_utils.kernel_maker.make_jpss_ck(parsed_args: Namespace)`

Create a JPSS CK from APID 11 CCSDS packets. The C-kernel (CK) is the component of SPICE concerned with attitude of spacecraft structures or instruments.

Parameters

parsed_args (`argparse.Namespace`) – Namespace of parsed CLI arguments

Return type

None

`libera_utils.kernel_maker.make_jpss_kernels_from_manifest(manifest_file_path: str, output_directory: str)`

Alpha function triggering kernel generation from manifest file.

If the manifest configuration field contains “start_time” and “end_time” fields then this function will select only packet data that falls in that range. If these are not given, then all packet data will be used.

Parameters

- **manifest_file_path** (`str` or `cloudpathlib.anypath.AnyPath`) – Path to the manifest file that includes end_time and start_time in the configuration section
- **output_directory** (`str` or `cloudpathlib.anypath.AnyPath`) – Path to save the completed kernels

Returns

output_directory – Path to the directory containing the completed kernels

Return type`str` or `cloudpathlib.anypath.AnyPath``libera_utils.kernel_maker.make_jpss_spk(parsed_args: Namespace)`

Create a JPSS SPK from APID 11 CCSDS packets. The SPK system is the component of SPICE concerned with ephemeris data (position/velocity).

Parameters

parsed_args (*argparse.Namespace*) – Namespace of parsed CLI arguments

Return type

None

`libera_utils.kernel_maker.write_kernel_input_file(data: ndarray, filepath: str, fields: list = None, fmt: str = '%.16f')`

Write ephemeris and attitude data to MKSPK and MSOPCK input data files, respectively.

See MSOPCK documentation here:

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/ug/msopck.html

See MKSPK documentation here:

https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/ug/mkspk.html

Parameters

- **data** (*numpy.ndarray*) – Structured array (named, with data types) of attitude or ephemeris data.
- **filepath** (*str* or *pathlib.Path*) – Filepath to write to.
- **fields** (*list*) – Optional. List of field names to write out to the data file. If not specified, assume fields are already in the proper order.
- **fmt** (*str* or *list*) – Format specifier(s) to pass to `np.savetxt`. Default is to assume everything should be floats with 16 decimal places of precision (`%.16f`). If a list is passed, it must contain a format specifier for each column in data.

Returns

Absolute path to written file.

Return type

pathlib.Path

`libera_utils.kernel_maker.write_kernel_setup_file(data: dict, filepath: Path)`

Write an MSOPCK or MKSPK compatible setup file of key-value pairs. See documentation here: https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/ug/msopck.html#Input%20Data%20Format

Parameters

- **data** (*dict*) – Dictionary of key-value pairs to write to the setup file.
- **filepath** (*pathlib.Path*) – Filepath to write to.

Returns

Absolute path to written file.

Return type

pathlib.Path

3.1.9 libera_utils.logutil

Logging utilities

Functions

<code>configure_static_logging(config_file)</code>	Configure logging based on a static logging configuration yaml file.
<code>configure_task_logging(task_id, *, ...)</code>	Configure logging for a specific task (e.g. a processing algorithm).
<code>flush_cloudwatch_logs()</code>	Force flush of all cloudwatch logging handlers.

libera_utils.logutil.configure_static_logging

`libera_utils.logutil.configure_static_logging(config_file: str | Path | S3Path)`

Configure logging based on a static logging configuration yaml file.

The yaml is interpreted as a dict configuration. There is no ability to customize this logging configuration at runtime.

Parameters

config_file (*cloudpathlib.anypath.AnyPath* or *str*) – Location of config file.

➔ See also

`configure_task_logging`

Runtime modifiable logging configuration.

libera_utils.logutil.configure_task_logging

`libera_utils.logutil.configure_task_logging(task_id: str, *, limit_debug_loggers: Iterable[str] | str | None = None, console_log_level: str | int = 20, console_log_json: bool = False, log_dir: str | Path | S3Path | None = None, cloudwatch_log_group: str | None = None)`

Configure logging for a specific task (e.g. a processing algorithm).

File-based logging is always done at the DEBUG level. Watchtower-based cloudwatch logging is always done at the DEBUG level. Console logging level defaults to INFO but can be set with `console_log_level`.

Examples

Example 1: The following will configure DEBUG console-only logging for anything in your script but all other loggers will be limited to INFO level.

```
`python configure_task_logging("my-script", limit_debug_loggers="__main__",
console_log_level=logging.DEBUG) `
```

Example 2: This will allow all debug messages through from all loggers and sets up file-based logging and a custom cloudwatch log group. Also console messages will be logged in serialized JSON.

```
```python configure_task_logging("my-script",
 console_log_level=logging.DEBUG, log_dir=Path("/tmp/my-script"), console_log_json=True,
 cloudwatch_log_group="custom-log-group")
```
```

Parameters

- **task_id** (*str*) – Unique identifier by which to name the log file and cloudwatch log stream.

- **limit_debug_loggers** (*Optional[Union[Iterable[str] | str]]*) – A list of logger name prefixes from which you want to allow debug messages (blocks debug from all others). For example, if you are working on a package called *my_app* and using module level logging, all your loggers will be named like *my_app.module_name.submodule_name*. By setting this to (*my_app*), all loggers that are named *my_app.** will propagate debug messages while preventing spammy debug messages from installed libraries like boto3. If this is empty or None, all debug messages will propagate. To use this in scripts, either leave it unset or use *limit_debug_loggers=(“__main__”)*.
- **console_log_level** (*str or int, Optional*) – Log level for console logging. If not specified, defaults to INFO
- **console_log_json** (*bool, Optional*) – If True, console logs will be JSON formatted. This is suitable for setting up loggers in AWS services that are automatically monitored by cloudwatch on stdout and stderr (e.g. Lambda or Batch)
- **log_dir** (*str or Path or S3Path, Optional*) – Log directory, which may be a local or S3Path. Default is None and results in no file-based logging.
- **cloudwatch_log_group** (*str, Optional*) – Override optional environment variable log group name. Default is None and will result in falling back to the LIBERA_LOG_GROUP environment variable. If that is not set, no cloudwatch JSON logging will be configured.

Notes

Even in the absence of cloudwatch JSON logging, all stdout/stderr messages generated by a Lambda will be logged to CloudWatch as string messages. Embedded JSON strings in log message text can still be queried in CloudWatch.

See also

[*configure_static_logging*](#)

Static logging configuration based on yaml file.

libera_utils.logutil.flush_cloudwatch_logs

`libera_utils.logutil.flush_cloudwatch_logs()`

Force flush of all cloudwatch logging handlers.

If you are missing the last few log messages in a log stream, this may help get those logs ingested before the process shuts down the logging system.

Return type

None

Classes

`JsonLogFormatter(*args[, ...])`

Altered version of the CloudWatchLogFormatter provided in the watchtower library

libera_utils.logutil.JsonLogFormatter

class libera_utils.logutil.JsonLogFormatter(*args, add_log_record_attrs: Tuple[str, ...] | None = None, add_asctime: bool = True, **kwargs)

Bases: `Formatter`

Altered version of the CloudWatchLogFormatter provided in the watchtower library

Methods

| | |
|--|--|
| <code>converter([seconds])</code> | Convert seconds since the Epoch to a time tuple expressing local time. |
| <code>format(record)</code> | Format log message to a string |
| <code>formatException(ei)</code> | Format and return the specified exception information as a string. |
| <code>formatStack(stack_info)</code> | This method is provided as an extension point for specialized formatting of stack information. |
| <code>formatTime(record[, datefmt])</code> | Return the creation time of the specified LogRecord as formatted text. |
| <code>usesTime()</code> | Check if the format uses the creation time of the record. |

formatMessage

`__init__`(*args, add_log_record_attrs: Tuple[str, ...] | None = None, add_asctime: bool = True, **kwargs)

Parameters

- **add_log_record_attrs** (*Optional*, *tuple*) – Tuple of log record attributes to add to the resulting structured JSON structure that comes out of the logging formatter.
- **add_asctime** (*bool*) – If True, adds an ASCII (ISO 8601-like) timestamp to the log record. Default True.

Methods

| | |
|-----------------------------|--------------------------------|
| <code>format(record)</code> | Format log message to a string |
|-----------------------------|--------------------------------|

Attributes

| |
|----------------------------------|
| <code>default_msec_format</code> |
| <code>default_time_format</code> |

format(record: *LogRecord*) → str
Format log message to a string

Parameters

record (*logging.LogRecord*) – Log record object containing the logged message, which may be a dict (Mapping) or a string

```
class libera_utils.logutil.JsonLogFormatter(*args, add_log_record_attrs: Tuple[str, ...] | None = None,
                                           add_asctime: bool = True, **kwargs)
```

Altered version of the CloudWatchLogFormatter provided in the watchtower library

Methods

| | |
|--|--|
| <code>converter([seconds])</code> | Convert seconds since the Epoch to a time tuple expressing local time. |
| <code>format(record)</code> | Format log message to a string |
| <code>formatException(ei)</code> | Format and return the specified exception information as a string. |
| <code>formatStack(stack_info)</code> | This method is provided as an extension point for specialized formatting of stack information. |
| <code>formatTime(record[, datefmt])</code> | Return the creation time of the specified LogRecord as formatted text. |
| <code>usesTime()</code> | Check if the format uses the creation time of the record. |

formatMessage

format (*record: LogRecord*) → *str*

Format log message to a string

Parameters

record (*logging.LogRecord*) – Log record object containing the logged message, which may be a dict (Mapping) or a string

`libera_utils.logutil._json_serialize_default(o: Any) → str`

A standard ‘default’ json serializer function.

- Serializes datetime objects using their `.isoformat()` method.
- Serializes all other objects using `repr()`.

`libera_utils.logutil.configure_static_logging(config_file: str | Path | S3Path)`

Configure logging based on a static logging configuration yaml file.

The yaml is interpreted as a dict configuration. There is no ability to customize this logging configuration at runtime.

Parameters

config_file (*cloudpathlib.anypath.AnyPath* or *str*) – Location of config file.

↪ See also***configure_task_logging***

Runtime modifiable logging configuration.

```
libera_utils.logutil.configure_task_logging(task_id: str, *, limit_debug_loggers: Iterable[str] | str |
None = None, console_log_level: str | int = 20,
console_log_json: bool = False, log_dir: str | Path | S3Path
| None = None, cloudwatch_log_group: str | None = None)
```

Configure logging for a specific task (e.g. a processing algorithm).

File-based logging is always done at the DEBUG level. Watchtower-based cloudwatch logging is always done at the DEBUG level. Console logging level defaults to INFO but can be set with `console_log_level`.

Examples

Example 1: The following will configure DEBUG console-only logging for anything in your script but all other loggers will be limited to INFO level.

```
`python configure_task_logging("my-script", limit_debug_loggers=("__main__",),
console_log_level=logging.DEBUG) `
```

Example 2: This will allow all debug messages through from all loggers and sets up file-based logging and a custom cloudwatch log group. Also console messages will be logged in serialized JSON.

```
```python configure_task_logging("my-script",
 console_log_level=logging.DEBUG, log_dir=Path("/tmp/my-script"), console_log_json=True,
 cloudwatch_log_group="custom-log-group")
```
```

Parameters

- **task_id** (*str*) – Unique identifier by which to name the log file and cloudwatch log stream.
- **limit_debug_loggers** (*Optional[Union[Iterable[str] | str]]*) – A list of logger name prefixes from which you want to allow debug messages (blocks debug from all others). For example, if you are working on a package called *my_app* and using module level logging, all your loggers will be named like *my_app.module_name.submodule_name*. By setting this to (*my_app*), all loggers that are named *my_app.** will propagate debug messages while preventing spammy debug messages from installed libraries like boto3. If this is empty or None, all debug messages will propagate. To use this in scripts, either leave it unset or use `limit_debug_loggers=("__main__")`.
- **console_log_level** (*str or int, Optional*) – Log level for console logging. If not specified, defaults to INFO
- **console_log_json** (*bool, Optional*) – If True, console logs will be JSON formatted. This is suitable for setting up loggers in AWS services that are automatically monitored by cloudwatch on stdout and stderr (e.g. Lambda or Batch)
- **log_dir** (*str or Path or S3Path, Optional*) – Log directory, which may be a local or S3Path. Default is None and results in no file-based logging.
- **cloudwatch_log_group** (*str, Optional*) – Override optional environment variable log group name. Default is None and will result in falling back to the LIBERA_LOG_GROUP environment variable. If that is not set, no cloudwatch JSON logging will be configured.

Notes

Even in the absence of cloudwatch JSON logging, all stdout/stderr messages generated by a Lambda will be logged to CloudWatch as string messages. Embedded JSON strings in log message text can still be queried in CloudWatch.

 See also***configure_static_logging***

Static logging configuration based on yaml file.

libera_utils.logutil.flush_cloudwatch_logs()

Force flush of all cloudwatch logging handlers.

If you are missing the last few log messages in a log stream, this may help get those logs ingested before the process shuts down the logging system.

Return type

None

3.1.10 libera_utils.packets

Module for reading packet data

Functions***array_from_packets***(packets[, apid])

Create an array from a list of packets.

parse_packets(packet_parser, ...[, apid])

Parse a recarray from a list of packet filepaths, assuming the same parser for all

libera_utils.packets.array_from_packets**libera_utils.packets.array_from_packets**(packets: list, apid: int = None)

Create an array from a list of packets. This function assumes that the fields and format for every packet is identical for a given APID.

Parameters

- **packets** (*list*) – List of `lasp_packets.parser.Packet` objects.
- **apid** (*int*) – Application Packet ID to create an array from. We can only create an array for a single APID because we need to assume the same fields in every packet. If not specified, every packet must be of the same APID.

Returns

Record array with one column per field name in the packet type. Values are derived if a derived value exists, otherwise, the values are the raw values.

Return type`numpy.recarray`**libera_utils.packets.parse_packets****libera_utils.packets.parse_packets**(packet_parser: PacketParser, packet_data_filepaths: list, apid: int = None)

Parse a recarray from a list of packet filepaths, assuming the same parser for all

Parameters

- **packet_parser** (*space_packet_parser.parser.PacketParser*) – Parser, already initialized with the anticipated definition.

- **packet_data_filepaths** (*list*) – List of filepaths to packets files.
- **apid** (*int*) – Filter on APID so we don't get mismatches in case the parser finds multiple parsable packet definitions in the files. This can happen if the XTCE document contains definitions for multiple packet types and >1 of those packet types is present in the packet data files.

Returns

Concatenated arrays of packet data.

Return type

`numpy.recarray`

`libera_utils.packets.array_from_packets`(*packets: list, apid: int = None*)

Create an array from a list of packets. This function assumes that the fields and format for every packet is identical for a given APID.

Parameters

- **packets** (*list*) – List of `lasp_packets.parser.Packet` objects.
- **apid** (*int*) – Application Packet ID to create an array from. We can only create an array for a single APID because we need to assume the same fields in every packet. If not specified, every packet must be of the same APID.

Returns

Record array with one column per field name in the packet type. Values are derived if a derived value exists, otherwise, the values are the raw values.

Return type

`numpy.recarray`

`libera_utils.packets.parse_packets`(*packet_parser: PacketParser, packet_data_filepaths: list, apid: int = None*)

Parse a recarray from a list of packet filepaths, assuming the same parser for all

Parameters

- **packet_parser** (*space_packet_parser.parser.PacketParser*) – Parser, already initialized with the anticipated definition.
- **packet_data_filepaths** (*list*) – List of filepaths to packets files.
- **apid** (*int*) – Filter on APID so we don't get mismatches in case the parser finds multiple parsable packet definitions in the files. This can happen if the XTCE document contains definitions for multiple packet types and >1 of those packet types is present in the packet data files.

Returns

Concatenated arrays of packet data.

Return type

`numpy.recarray`

3.1.11 libera_utils.quality_flags

Quality flag definitions

Functions

| | |
|-------------------------------|---|
| <code>with_all_none(f)</code> | Decorator that adds <i>NONE</i> and <i>ALL</i> psuedo-members to a QualityFlag <i>f</i> |
|-------------------------------|---|

libera_utils.quality_flags.with_all_none

`libera_utils.quality_flags.with_all_none(f)`

Decorator that adds *NONE* and *ALL* psuedo-members to a QualityFlag *f*

For example:

```
@with_all_none
class MyQualityFlag(QualityFlag, metaclass=FrozenFlagMeta):
    MISSING_DATA = FlagBit(0b1, message="Data is missing!")
    VOLTAGE_TOO_HIGH = FlagBit(0b10, message="Voltage is too high!")
qf = MyQualityFlag.ALL # Equivalent to MyQualityFlag(0b11)
qf.summary
```

Classes

| | |
|---|---|
| <code>FlagBit(*args[, message])</code> | Subclass of int to capture both an integer value and an accompanying message |
| <code>FrozenFlagMeta(name, bases, classdict)</code> | Metaclass that freezes an enum entirely, preventing values from being updated, added, or deleted. |
| <code>QualityFlag(value[, names, module, ...])</code> | Subclass of Flag that add a method for decomposing a flag into its individual components and a property to return a list of all messages associated with a quality flag |

libera_utils.quality_flags.FlagBit

`class libera_utils.quality_flags.FlagBit(*args, message=None, **kwargs)`

Bases: `int`

Subclass of int to capture both an integer value and an accompanying message

Attributes

denominator

the denominator of a rational number in lowest terms

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

Methods

| | |
|---------------------------------|-----------------------|
| <code>as_integer_ratio()</code> | Return integer ratio. |
|---------------------------------|-----------------------|

continues on next page

Table 102 – continued from previous page

| | |
|--|--|
| <code>bit_count()</code> | Number of ones in the binary representation of the absolute value of self. |
| <code>bit_length()</code> | Number of bits necessary to represent self in binary. |
| <code>conjugate</code> | Returns self, the complex conjugate of any int. |
| <code>from_bytes(/, bytes[, byteorder, signed])</code> | Return the integer represented by the given array of bytes. |
| <code>to_bytes(/[, length, byteorder, signed])</code> | Return an array of bytes representing an integer. |

`__init__(*args, **kwargs)`

Methods

| | |
|--|--|
| <code>as_integer_ratio()</code> | Return integer ratio. |
| <code>bit_count()</code> | Number of ones in the binary representation of the absolute value of self. |
| <code>bit_length()</code> | Number of bits necessary to represent self in binary. |
| <code>conjugate</code> | Returns self, the complex conjugate of any int. |
| <code>from_bytes(/, bytes[, byteorder, signed])</code> | Return the integer represented by the given array of bytes. |
| <code>to_bytes(/[, length, byteorder, signed])</code> | Return an array of bytes representing an integer. |

Attributes

| | |
|--------------------------|--|
| <code>denominator</code> | the denominator of a rational number in lowest terms |
| <code>imag</code> | the imaginary part of a complex number |
| <code>numerator</code> | the numerator of a rational number in lowest terms |
| <code>real</code> | the real part of a complex number |

`as_integer_ratio()`

Return integer ratio.

Return a pair of integers, whose ratio is exactly equal to the original int and with a positive denominator.

```
>>> (10).as_integer_ratio()
(10, 1)
>>> (-10).as_integer_ratio()
(-10, 1)
>>> (0).as_integer_ratio()
(0, 1)
```

`bit_count()`

Number of ones in the binary representation of the absolute value of self.

Also known as the population count.

```
>>> bin(13)
'0b1101'
>>> (13).bit_count()
3
```

bit_length()

Number of bits necessary to represent self in binary.

```
>>> bin(37)
'0b100101'
>>> (37).bit_length()
6
```

conjugate()

Returns self, the complex conjugate of any int.

denominator

the denominator of a rational number in lowest terms

classmethod from_bytes(/, bytes, byteorder='big', *, signed=False)

Return the integer represented by the given array of bytes.

bytes

Holds the array of bytes to convert. The argument must either support the buffer protocol or be an iterable object producing bytes. Bytes and bytearray are examples of built-in objects that support the buffer protocol.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Indicates whether two's complement is used to represent the integer.

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

to_bytes(/, length=1, byteorder='big', *, signed=False)

Return an array of bytes representing an integer.

length

Length of bytes object to use. An `OverflowError` is raised if the integer is not representable with the given number of bytes. Default is length 1.

byteorder

The byte order used to represent the integer. If byteorder is 'big', the most significant byte is at the beginning of the byte array. If byteorder is 'little', the most significant byte is at the end of the byte array. To request the native byte order of the host system, use `sys.byteorder` as the byte order value. Default is to use 'big'.

signed

Determines whether two's complement is used to represent the integer. If signed is False and a negative integer is given, an `OverflowError` is raised.

libera_utils.quality_flags.FrozenFlagMeta

class libera_utils.quality_flags.**FrozenFlagMeta**(*name, bases, classdict*)

Bases: EnumType

Metaclass that freezes an enum entirely, preventing values from being updated, added, or deleted.

Methods

| | |
|---|---|
| <code>__call__(value[, names, module])</code> | Either returns an existing member, or creates a new enum class. |
| <code>mro()</code> | Return a type's method resolution order. |

`__init__(*args, **kwargs)`

Methods

| | |
|--------------------|--|
| <code>mro()</code> | Return a type's method resolution order. |
|--------------------|--|

`mro()`

Return a type's method resolution order.

libera_utils.quality_flags.QualityFlag

class libera_utils.quality_flags.**QualityFlag**(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: Flag

Subclass of Flag that add a method for decomposing a flag into its individual components and a property to return a list of all messages associated with a quality flag

`__init__(*args, **kws)`

class libera_utils.quality_flags.**FlagBit**(**args, message=None, **kwargs*)

Subclass of int to capture both an integer value and an accompanying message

Attributes

denominator

the denominator of a rational number in lowest terms

imag

the imaginary part of a complex number

numerator

the numerator of a rational number in lowest terms

real

the real part of a complex number

Methods

| | |
|---------------------------------|-----------------------|
| <code>as_integer_ratio()</code> | Return integer ratio. |
|---------------------------------|-----------------------|

continues on next page

Table 107 – continued from previous page

| | |
|--|--|
| <code>bit_count()</code> | Number of ones in the binary representation of the absolute value of self. |
| <code>bit_length()</code> | Number of bits necessary to represent self in binary. |
| <code>conjugate</code> | Returns self, the complex conjugate of any int. |
| <code>from_bytes(/, bytes[, byteorder, signed])</code> | Return the integer represented by the given array of bytes. |
| <code>to_bytes(/, length, byteorder, signed)</code> | Return an array of bytes representing an integer. |

class `libera_utils.quality_flags.FrozenFlagMeta`(*name, bases, classdict*)

Metaclass that freezes an enum entirely, preventing values from being updated, added, or deleted.

Methods

| | |
|---|---|
| <code>__call__(value[, names, module])</code> | Either returns an existing member, or creates a new enum class. |
| <code>mro()</code> | Return a type's method resolution order. |

class `libera_utils.quality_flags.QualityFlag`(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Subclass of `Flag` that add a method for decomposing a flag into its individual components and a property to return a list of all messages associated with a quality flag

`libera_utils.quality_flags.with_all_none`(*f*)

Decorator that adds *NONE* and *ALL* psuedo-members to a `QualityFlag` *f*

For example:

```
@with_all_none
class MyQualityFlag(QualityFlag, metaclass=FrozenFlagMeta):
    MISSING_DATA = FlagBit(0b1, message="Data is missing!")
    VOLTAGE_TOO_HIGH = FlagBit(0b10, message="Voltage is too high!")
qf = MyQualityFlag.ALL # Equivalent to MyQualityFlag(0b11)
qf.summary
```

3.1.12 libera_utils.spice_utils

Modules for SPICE kernel creation, management, and usage

Functions

| | |
|---|--|
| <code>ensure_spice</code> (<i>[f_py, time_kernels_only]</i>) | Before trying to understand this piece of code, read this: https://stackoverflow.com/questions/5929107/decorators-with-parameters/60832711#60832711 |
| <code>find_most_recent_naif_kernel</code> (<i>naif_base_url, ...</i>) | Retrieves the name of the most recent kernel at NAIF. |
| <code>ls_kernel_coverage</code> (<i>kernel_type[, verbose]</i>) | List time coverage of all furnished kernels of a given type |
| <code>ls_kernels</code> (<i>[verbose, log]</i>) | List all furnished spice kernels. |
| <code>ls_spice_constants</code> (<i>[verbose]</i>) | List all constants in the Spice constant pool |

libera_utils.spice_utils.ensure_spice

`libera_utils.spice_utils.ensure_spice(f_py: Callable = None, time_kernels_only: bool = False)`

Before trying to understand this piece of code, read this: <https://stackoverflow.com/questions/5929107/decorators-with-parameters/60832711#60832711>

Decorator/wrapper that tries to ensure that a metakernel is furnished in as complete a way as possible.

Control flow overview:

1. **Try simply calling the wrapped function naively.**
 - SUCCESS? Great! We're done.
 - SpicyError? Go to step 2.
2. **Furnish metakernel at SPICE_METAKERNEL**
 - SUCCESS? Great, return the original function again (so it can be re-run).
 - KeyError? Seems like SPICE_METAKERNEL isn't set, no problem. Go to step 3.

Usage:

Three ways to use this object

1. A decorator with no arguments

```
@ensure_spice
def my_spicy_func(a, b):
    pass
```

2. A decorator with parameters. This is useful if we only need the latest SCLK and LSK kernels for the function involved.

```
@ensure_spice(time_kernels_only=True)
def my_spicy_time_func(a, b):
    pass
```

3. An explicit wrapper function, providing a dynamically set value for parameters, e.g. `time_kernels_only`

```
wrapped = ensure_spice(spicy_func, time_kernels_only=True)
result = wrapped(*args, **kwargs)
```

Parameters

- **f_py** (*Callable*) – The function requiring SPICE that we are going to wrap if being used explicitly, Otherwise None, in which case `ensure_spice` is being used, not as a function wrapper (see `l2a_processing.py`) but as a true decorator without an explicit function argument.
- **time_kernels_only** (*bool, Optional*) – Specify that we only need to furnish time kernels (if `SPICE_METAKERNEL` is set, we still just furnish that metakernel and assume the time kernels are included).

Returns

Decorated function, with spice error handling

Return type

Callable

libera_utils.spice_utils.find_most_recent_naif_kernel

libera_utils.spice_utils.find_most_recent_naif_kernel(*naif_base_url*: str, *kernel_file_regex*: str, *allowed_attempts*: int = 3) → str

Retrieves the name of the most recent kernel at NAIF.

Parameters

- **naif_base_url** (str) – URL to search for filenames matching kernel_file_regex
- **kernel_file_regex** (str) – Regular expression to match filenames on the naif website
- **allowed_attempts** (int, Optional) – Number of allowed download times for naif page default = 3

Returns

Returns the file name of the latest kernel on the naif page (e.g., “naif0012.tls”)

Return type

str

libera_utils.spice_utils.ls_kernel_coverage

libera_utils.spice_utils.ls_kernel_coverage(*kernel_type*: str, *verbose*: bool = False) → dict

List time coverage of all furnished kernels of a given type

Parameters

- **kernel_type** (str) – Either ‘CK’ or ‘SPK’
- **verbose** (bool) – If True, print to stdout also

Returns

Key is filename, value is a list of tuples giving the start and end times in ET.

Return type

dict

libera_utils.spice_utils.ls_kernels

libera_utils.spice_utils.ls_kernels(*verbose*: bool = False, *log*: bool = False) → list

List all furnished spice kernels.

Parameters

- **verbose** (bool) – If True, print to stdout also
- **log** (bool) – Whether or not to log the current kernel pool (this gets called a lot)

Returns

A list of KernelFileRecord named tuples.

Return type

list

libera_utils.spice_utils.ls_spice_constants

libera_utils.spice_utils.ls_spice_constants(*verbose*: bool = False) → dict

List all constants in the Spice constant pool

Parameters

verbose – If true, print to stdout also

Returns

Dictionary of kernel constants

Return type

dict

Classes

| | |
|---|--|
| <i>KernelFileCache</i> (kernel_url[, max_cache_age, ...]) | Class for downloading, caching, and furnishing SPICE kernel files locally. |
| <i>KernelFileRecord</i> (kernel_type, file_name) | Tuple for keeping track of kernel files with default kernel_level |
| <i>SpiceBody</i> (value) | Enum containing SPICE IDs for ephemeris bodies that we use. |
| <i>SpiceFrame</i> (value) | Enum containing SPICE IDs for reference frames, possibly defined in the Frame Kernel (FK) |
| <i>SpiceId</i> (strid, numid) | Class that represents a unique identifier in the NAIF SPICE library |
| <i>SpiceInstrument</i> (value) | Enum containing SPICE IDs for instrument geometries configured in the Instrument Kernel (IK) |

libera_utils.spice_utils.KernelFileCache

```
class libera_utils.spice_utils.KernelFileCache(kernel_url: str, max_cache_age: timedelta =
datetime.timedelta(days=1), fallback_kernel: Path =
None)
```

Bases: `object`

Class for downloading, caching, and furnishing SPICE kernel files locally.

It attempts to find a cached kernel file in the user's cache directory (OS-specific location). If that file is not there or is old, it attempts to download it from the specified location. If it is unable to do that, it can optionally read a fallback file included in the libera_utils package but this is not recommended.

Attributes*cache_dir*

Property that calls out to get the proper local cache directory

kernel_basename

Base filename of the kernel.

kernel_path

Return the local path location of the kernel if it exists.

Methods

| | |
|---|---|
| <i>clear</i> () | Remove cached kernel file |
| <i>download_kernel</i> (kernel_url[, allowed_attempts]) | Downloads a kernel from a URL or an S3 location to the system cache location. |
| <i>furnsh</i> () | Furnish the cached kernel |
| <i>is_cached</i> ([include_stale]) | Check the cache directory for kernel file that is within cache age limit. |

`__init__(kernel_url: str, max_cache_age: timedelta = datetime.timedelta(days=1), fallback_kernel: Path = None)`

Create a new file cache. Downloading is done on first access of kernel_path if the file is not already cached. Fallback occurs only after failing to download. :param kernel_url: Location of kernel file as a URL or an S3Path :type kernel_url: str or cloudpathlib.S3Path :param max_cache_age: Length of time to tolerate stale kernels in the cache without forcing a redownload. :type max_cache_age: datetime.timedelta :param fallback_kernel: Path pointing to a fallback kernel location. May be None, which disallows a fallback. :type fallback_kernel: pathlib.Path

Methods

| | |
|--|---|
| <code>clear()</code> | Remove cached kernel file |
| <code>download_kernel(kernel_url[, allowed_attempts])</code> | Downloads a kernel from a URL or an S3 location to the system cache location. |
| <code>furnsh()</code> | Furnish the cached kernel |
| <code>is_cached([include_stale])</code> | Check the cache directory for kernel file that is within cache age limit. |

Attributes

| | |
|------------------------------|---|
| <code>cache_dir</code> | Property that calls out to get the proper local cache directory |
| <code>kernel_basename</code> | Base filename of the kernel. |
| <code>kernel_path</code> | Return the local path location of the kernel if it exists. |

property cache_dir

Property that calls out to get the proper local cache directory

Returns

Path to the proper local cache for the system.

Return type

`pathlib.Path`

clear()

Remove cached kernel file

`download_kernel(kernel_url: str, allowed_attempts: int = 3) → Path`

Downloads a kernel from a URL or an S3 location to the system cache location.

Parameters

- **kernel_url** (*str*) – Filename of kernel on NAIF site, as discovered by `find_most_recent_naif_kernel`
- **allowed_attempts** (*int*, *Optional*) – Number of allowed download times for naif kernel default = 3

Returns

Location of downloaded file

Return type

`pathlib.Path`

furnsh()

Furnish the cached kernel

is_cached(*include_stale: bool = False*) → bool

Check the cache directory for kernel file that is within cache age limit. If present, return True.

Parameters

include_stale (*bool*) – Default False. If True, results include kernel that are past the max age.

Returns

Returns True if kernel is present locally and within the age limit.

Return type

bool

property kernel_basename

Base filename of the kernel.

Return type

str

property kernel_path: Path

Return the local path location of the kernel if it exists. If not, try downloading it. If that fails, return the fallback kernel, if allowed.

libera_utils.spice_utils.KernelFileRecord

class libera_utils.spice_utils.**KernelFileRecord**(*kernel_type: str, file_name: str*)

Bases: `NamedTuple`

Tuple for keeping track of kernel files with default kernel_level

Methods

| | |
|---|--|
| <code>count</code> (value, /) | Return number of occurrences of value. |
| <code>index</code> (value[, start, stop]) | Return first index of value. |

`__init__`(*args, **kwargs)

Methods

| | |
|---|--|
| <code>count</code> (value, /) | Return number of occurrences of value. |
| <code>index</code> (value[, start, stop]) | Return first index of value. |

Attributes

| | |
|--------------------------|--------------------------|
| <code>file_name</code> | Alias for field number 1 |
| <code>kernel_type</code> | Alias for field number 0 |

count(*value, /*)

Return number of occurrences of value.

file_name: `str`

Alias for field number 1

index(*value*, *start=0*, *stop=sys.maxsize*, /)

Return first index of value.

Raises ValueError if the value is not present.

kernel_type: `str`

Alias for field number 0

libera_utils.spice_utils.SpiceBody

class libera_utils.spice_utils.SpiceBody(*value*)

Bases: `Enum`

Enum containing SPICE IDs for ephemeris bodies that we use.

__init__(*args, **kws)

Attributes

| |
|-----------------------|
| JPSS |
| SSB |
| SUN |
| EARTH |
| EARTH_MOON_BARYCENTER |

libera_utils.spice_utils.SpiceFrame

class libera_utils.spice_utils.SpiceFrame(*value*)

Bases: `Enum`

Enum containing SPICE IDs for reference frames, possibly defined in the Frame Kernel (FK)

__init__(*args, **kws)

Attributes

| |
|-------------|
| J2000 |
| ITRF93 |
| EARTH_FIXED |

libera_utils.spice_utils.SpiceId

class libera_utils.spice_utils.**SpiceId**(*strid: str, numid: int*)

Bases: `NamedTuple`

Class that represents a unique identifier in the NAIF SPICE library

Methods

| | |
|---|--|
| <code>count</code> (value, /) | Return number of occurrences of value. |
| <code>index</code> (value[, start, stop]) | Return first index of value. |

`__init__`(*args, **kwargs)

Methods

| | |
|---|--|
| <code>count</code> (value, /) | Return number of occurrences of value. |
| <code>index</code> (value[, start, stop]) | Return first index of value. |

Attributes

| | |
|--------------------|--------------------------|
| <code>numid</code> | Alias for field number 1 |
| <code>strid</code> | Alias for field number 0 |

count(value, /)

Return number of occurrences of value.

index(value, start=0, stop=sys.maxsize, /)

Return first index of value.

Raises ValueError if the value is not present.

numid: int

Alias for field number 1

strid: str

Alias for field number 0

libera_utils.spice_utils.SpiceInstrument

class libera_utils.spice_utils.**SpiceInstrument**(value)

Bases: `Enum`

Enum containing SPICE IDs for instrument geometries configured in the Instrument Kernel (IK)

`__init__`(*args, **kws)

class libera_utils.spice_utils.**KernelFileCache**(*kernel_url: str, max_cache_age: timedelta = datetime.timedelta(days=1), fallback_kernel: Path = None*)

Class for downloading, caching, and furnishing SPICE kernel files locally.

It attempts to find a cached kernel file in the user's cache directory (OS-specific location). If that file is not there or is old, it attempts to download it from the specified location. If it is unable to do that, it can optionally read a fallback file included in the libera_utils package but this is not recommended.

Attributes

cache_dir

Property that calls out to get the proper local cache directory

kernel_basename

Base filename of the kernel.

kernel_path

Return the local path location of the kernel if it exists.

Methods

| | |
|---|---|
| <i>clear()</i> | Remove cached kernel file |
| <i>download_kernel</i> (kernel_url[, allowed_attempts]) | Downloads a kernel from a URL or an S3 location to the system cache location. |
| <i>furnsh()</i> | Furnish the cached kernel |
| <i>is_cached</i> ([include_stale]) | Check the cache directory for kernel file that is within cache age limit. |

property *cache_dir*

Property that calls out to get the proper local cache directory

Returns

Path to the proper local cache for the system.

Return type

`pathlib.Path`

clear()

Remove cached kernel file

download_kernel(kernel_url: str, allowed_attempts: int = 3) → Path

Downloads a kernel from a URL or an S3 location to the system cache location.

Parameters

- **kernel_url** (*str*) – Filename of kernel on NAIF site, as discovered by `find_most_recent_naif_kernel`
- **allowed_attempts** (*int*, *Optional*) – Number of allowed download times for naif kernel default = 3

Returns

Location of downloaded file

Return type

`pathlib.Path`

furnsh()

Furnish the cached kernel

is_cached(include_stale: bool = False) → bool

Check the cache directory for kernel file that is within cache age limit. If present, return True.

Parameters

include_stale (*bool*) – Default False. If True, results include kernel that are past the max age.

Returns

Returns True if kernel is present locally and within the age limit.

Return type

bool

property kernel_basename

Base filename of the kernel.

Return type

str

property kernel_path: Path

Return the local path location of the kernel if it exists. If not, try downloading it. If that fails, return the fallback kernel, if allowed.

class libera_utils.spice_utils.**KernelFileRecord**(*kernel_type: str, file_name: str*)

Tuple for keeping track of kernel files with default kernel_level

Methods

| | |
|-------------------------------------|--|
| <i>count</i> (value, /) | Return number of occurrences of value. |
| <i>index</i> (value[, start, stop]) | Return first index of value. |

file_name: str

Alias for field number 1

kernel_type: str

Alias for field number 0

class libera_utils.spice_utils.**SpiceBody**(*value*)

Enum containing SPICE IDs for ephemeris bodies that we use.

class libera_utils.spice_utils.**SpiceFrame**(*value*)

Enum containing SPICE IDs for reference frames, possibly defined in the Frame Kernel (FK)

class libera_utils.spice_utils.**SpiceId**(*strid: str, numid: int*)

Class that represents a unique identifier in the NAIF SPICE library

Methods

| | |
|-------------------------------------|--|
| <i>count</i> (value, /) | Return number of occurrences of value. |
| <i>index</i> (value[, start, stop]) | Return first index of value. |

numid: int

Alias for field number 1

strid: str

Alias for field number 0

class libera_utils.spice_utils.SpiceInstrument(*value*)

Enum containing SPICE IDs for instrument geometries configured in the Instrument Kernel (IK)

libera_utils.spice_utils.ensure_spice(*f_py*: *Callable* = *None*, *time_kernels_only*: *bool* = *False*)

Before trying to understand this piece of code, read this: <https://stackoverflow.com/questions/5929107/decorators-with-parameters/60832711#60832711>

Decorator/wrapper that tries to ensure that a metakernel is furnished in as complete a way as possible.

Control flow overview:

1. **Try simply calling the wrapped function naively.**
 - SUCCESS? Great! We're done.
 - SpiceyError? Go to step 2.
2. **Furnish metakernel at SPICE_METAKERNEL**
 - SUCCESS? Great, return the original function again (so it can be re-run).
 - KeyError? Seems like SPICE_METAKERNEL isn't set, no problem. Go to step 3.

Usage:

Three ways to use this object

1. A decorator with no arguments

```
@ensure_spice
def my_spicey_func(a, b):
    pass
```

2. A decorator with parameters. This is useful if we only need the latest SCLK and LSK kernels for the function involved.

```
@ensure_spice(time_kernels_only=True)
def my_spicey_time_func(a, b):
    pass
```

3. An explicit wrapper function, providing a dynamically set value for parameters, e.g. time_kernels_only

```
wrapped = ensure_spice(spicey_func, time_kernels_only=True)
result = wrapped(*args, **kwargs)
```

Parameters

- **f_py** (*Callable*) – The function requiring SPICE that we are going to wrap if being used explicitly, Otherwise *None*, in which case ensure_spice is being used, not as a function wrapper (see l2a_processing.py) but as a true decorator without an explicit function argument.
- **time_kernels_only** (*bool*, *Optional*) – Specify that we only need to furnish time kernels (if SPICE_METAKERNEL is set, we still just furnish that metakernel and assume the time kernels are included).

Returns

Decorated function, with spice error handling

Return type

Callable

`libera_utils.spice_utils.find_most_recent_naif_kernel` (*naif_base_url: str, kernel_file_regex: str, allowed_attempts: int = 3*) → *str*

Retrieves the name of the most recent kernel at NAIF.

Parameters

- **naif_base_url** (*str*) – URL to search for filenames matching `kernel_file_regex`
- **kernel_file_regex** (*str*) – Regular expression to match filenames on the naif website
- **allowed_attempts** (*int, Optional*) – Number of allowed download times for naif page default = 3

Returns

Returns the file name of the latest kernel on the naif page (e.g., “naif0012.tls”)

Return type

str

`libera_utils.spice_utils.ls_kernel_coverage` (*kernel_type: str, verbose: bool = False*) → *dict*

List time coverage of all furnished kernels of a given type

Parameters

- **kernel_type** (*str*) – Either ‘CK’ or ‘SPK’
- **verbose** (*bool*) – If True, print to stdout also

Returns

Key is filename, value is a list of tuples giving the start and end times in ET.

Return type

dict

`libera_utils.spice_utils.ls_kernels` (*verbose: bool = False, log: bool = False*) → *list*

List all furnished spice kernels.

Parameters

- **verbose** (*bool*) – If True, print to stdout also
- **log** (*bool*) – Whether or not to log the current kernel pool (this gets called a lot)

Returns

A list of `KernelFileRecord` named tuples.

Return type

list

`libera_utils.spice_utils.ls_spice_constants` (*verbose: bool = False*) → *dict*

List all constants in the Spice constant pool

Parameters

verbose – If true, print to stdout also

Returns

Dictionary of kernel constants

Return type

dict

3.1.13 libera_utils.time

Module for dealing with time and time conventions

Some convention for this module

1. Only decorate direct spiceypy wrapper functions with the `ensure_spice` decorator. They should directly call a spiceypy function.
2. All spiceypy wrapper functions should read as `<spicepyfunc>_wrapper`. We really only use these to allow array inputs for spiceypy functions that aren't already vectorized in C and to wrap them in `ensure_spice`.
3. All functions should have robust type-hinting.

Functions

| | |
|--|--|
| <code>convert_cds_integer_to_datetime(satellite_time)</code> | Helper function to convert a satellite time given as an CCSDS Day Segmented Time Code (CDS) form as 8 byte integer to a timezone aware datetime object |
| <code>et2utc_wrapper(et, fmt, prec)</code> | Convert ephemeris times to UTC ISO strings. |
| <code>et_2_datetime(et)</code> | Convert ephemeris time to a python datetime object by first converting it to a UTC timestamp. |
| <code>et_2_timestamp(et[, fmt])</code> | Convert ephemeris time to a custom formatted timestamp (default is lowercase version of ISO). |
| <code>sce2s_wrapper(et)</code> | Convert ephemeris times to SCLK string https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sce2s_c.html
Decorated wrapper for <code>spiceypy.sce2s</code> that will automatically furnish the latest metakernel and retry if the first call raises an exception. |
| <code>scs2e_wrapper(sclk_str)</code> | Convert SCLK strings to ephemeris time. |
| <code>utc2et_wrapper(iso_str)</code> | Convert UTC ISO strings to ephemeris times. |

libera_utils.time.convert_cds_integer_to_datetime

`libera_utils.time.convert_cds_integer_to_datetime(satellite_time: int)`

Helper function to convert a satellite time given as an CCSDS Day Segmented Time Code (CDS) form as 8 byte integer to a timezone aware datetime object

Parameters

satellite_time (*int*) – A 64-bit unsigned integer that represents CDS time

Returns

cds_time

Return type

`datetime.datetime`

libera_utils.time.et2utc_wrapper

`libera_utils.time.et2utc_wrapper(et: float | Collection[float] | ndarray, fmt: str, prec: int) → str | Collection[str]`

Convert ephemeris times to UTC ISO strings. https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/et2utc_c.html Decorated wrapper for `spiceypy.et2utc` that will automatically furnish the latest metakernel and retry if the first call raises an exception.

Parameters

- **et** (*Union[float, Collection[float], numpy.ndarray*) – The ephemeris time value to be converted to UTC.
- **fmt** (*str*) – Format string defines the format of the output time string. See CSPICE docs.
- **prec** (*int*) – Number of digits of precision for fractional seconds.

Returns

UTC time string(s)

Return type

Union[numpy.ndarray, str]

libera_utils.time.et_2_datetime`libera_utils.time.et_2_datetime(et: float | Collection[float] | ndarray) → datetime | ndarray`

Convert ephemeris time to a python datetime object by first converting it to a UTC timestamp.

Parameters**et** (*float or Collection or numpy.ndarray*) – Ephemeris times to be converted.**Returns**

Object representation of ephemeris times.

Return type

datetime.datetime or numpy.ndarray

libera_utils.time.et_2_timestamp`libera_utils.time.et_2_timestamp(et: float | Collection[float] | ndarray, fmt: str = '%Y%m%dT%H%M%S.%f') → str | Collection[str]`

Convert ephemeris time to a custom formatted timestamp (default is lowercase version of ISO).

Parameters

- **et** (*Union[float, Collection[float], numpy.ndarray*) – Ephemeris Time to be converted.
- **fmt** (*str, Optional*) – Format string as defined by the datetime.strftime() function.

Returns

Formatted timestamps

Return type

Union[str, Collection[str]]

libera_utils.time.sce2s_wrapper`libera_utils.time.sce2s_wrapper(et: float | Collection[float] | ndarray) → str | ndarray`Convert ephemeris times to SCLK string https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sce2s_c.html
Decorated wrapper for spiceypy.sce2s that will automatically furnish the latest metakernel and retry if the first call raises an exception.**Parameters****et** (*Union[float, Collection[float], numpy.ndarray*) – Ephemeris time**Returns**

SCLK string

Return type

Union[str, Collection[str]]

libera_utils.time.scs2e_wrapper

`libera_utils.time.scs2e_wrapper(sclk_str: str | Collection[str]) → float | ndarray`

Convert SCLK strings to ephemeris time. https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/scs2e_c.html
Decorated wrapper for `spiceypy.scs2e` that will automatically furnish the latest metakernel and retry if the first call raises an exception.

Parameters

sclk_str (`Union[str, Collection[str]]`) – Spacecraft clock string

Returns

Ephemeris time

Return type

`Union[float, numpy.ndarray]`

libera_utils.time.utc2et_wrapper

`libera_utils.time.utc2et_wrapper(iso_str: str | Collection[str]) → float | ndarray`

Convert UTC ISO strings to ephemeris times. https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/utc2et_c.html
Decorated wrapper for `spiceypy.utc2et` that will automatically furnish the latest metakernel and retry if the first call raises an exception.

Parameters

iso_str (`Union[str, Collection[str]]`) – The UTC to convert to ephemeris time

Returns

Ephemeris time

Return type

`float` or `numpy.ndarray`

`libera_utils.time.convert_cds_integer_to_datetime(satellite_time: int)`

Helper function to convert a satellite time given as an CCSDS Day Segmented Time Code (CDS) form as 8 byte integer to a timezone aware datetime object

Parameters

satellite_time (`int`) – A 64-bit unsigned integer that represents CDS time

Returns

`cds_time`

Return type

`datetime.datetime`

`libera_utils.time.et2utc_wrapper(et: float | Collection[float] | ndarray, fmt: str, prec: int) → str | Collection[str]`

Convert ephemeris times to UTC ISO strings. https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/et2utc_c.html
Decorated wrapper for `spiceypy.et2utc` that will automatically furnish the latest metakernel and retry if the first call raises an exception.

Parameters

- **et** (`Union[float, Collection[float], numpy.ndarray]`) – The ephemeris time value to be converted to UTC.
- **fmt** (`str`) – Format string defines the format of the output time string. See CSPICE docs.
- **prec** (`int`) – Number of digits of precision for fractional seconds.

Returns

UTC time string(s)

Return type

Union[numpy.ndarray, str]

`libera_utils.time.et_2_datetime`(*et*: float | Collection[float] | ndarray) → datetime | ndarray

Convert ephemeris time to a python datetime object by first converting it to a UTC timestamp.

Parameters

et (float or Collection or numpy.ndarray) – Ephemeris times to be converted.

Returns

Object representation of ephemeris times.

Return type

datetime.datetime or numpy.ndarray

`libera_utils.time.et_2_timestamp`(*et*: float | Collection[float] | ndarray, *fmt*: str = '%Y%m%dT%H%M%S.%f') → str | Collection[str]

Convert ephemeris time to a custom formatted timestamp (default is lowercase version of ISO).

Parameters

- *et* (Union[float, Collection[float], numpy.ndarray]) – Ephemeris Time to be converted.
- *fmt* (str, Optional) – Format string as defined by the datetime.strftime() function.

Returns

Formatted timestamps

Return type

Union[str, Collection[str]]

`libera_utils.time.sce2s_wrapper`(*et*: float | Collection[float] | ndarray) → str | ndarray

Convert ephemeris times to SCLK string https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/sce2s_c.html
Decorated wrapper for spiceypy.sce2s that will automatically furnish the latest metakernel and retry if the first call raises an exception.

Parameters

et (Union[float, Collection[float], numpy.ndarray]) – Ephemeris time

Returns

SCLK string

Return type

Union[str, Collection[str]]

`libera_utils.time.scs2e_wrapper`(*sclk_str*: str | Collection[str]) → float | ndarray

Convert SCLK strings to ephemeris time. https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/scs2e_c.html
Decorated wrapper for spiceypy.scs2e that will automatically furnish the latest metakernel and retry if the first call raises an exception.

Parameters

sclk_str (Union[str, Collection[str]]) – Spacecraft clock string

Returns

Ephemeris time

Return type

Union[float, numpy.ndarray]

`libera_utils.time.utc2et_wrapper(iso_str: str | Collection[str]) → float | ndarray`

Convert UTC ISO strings to ephemeris times. https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/C/cspice/utc2et_c.html Decorated wrapper for `spiceypy.utc2et` that will automatically furnish the latest metakernel and retry if the first call raises an exception.

Parameters

`iso_str` (`Union[str, Collection[str]]`) – The UTC to convert to ephemeris time

Returns

Ephemeris time

Return type

float or `numpy.ndarray`

3.1.14 libera_utils.version

Module for anything related to package versioning

Functions

| | |
|------------------------|-----------------------------------|
| <code>version()</code> | Get package version from metadata |
|------------------------|-----------------------------------|

`libera_utils.version.version`

`libera_utils.version.version()`

Get package version from metadata

`libera_utils.version.version()`

Get package version from metadata

VERSION CHANGES

4.1 2.5.2 (released)

- Added the `s3-utils` cli interface with subcommands `put`, `list`, and `cp` for ease of use `s3` interactions
- The cli subcommand `libera-utils s3-utils put` will upload a file to the correct `S3` archive bucket given an algorithm
- The cli subcommand `libera-utils s3-utils list` will list all files in a given `S3` archive bucket for a given algorithm
- The cli subcommand `libera-utils s3-utils cp` will copy a file from one `S3` archive bucket to another location
- Added the Libera Archive bucket naming pattern to the `aws constants` module
- Improved type hinting for the `smart_open` module
- Improved readability of cli testing
- Removing testing against python 3.9 and 3.10

4.2 2.5.1 (released)

- BUGFIX: missing `pyyaml` dependency that prevented usage of `logutil` module

4.3 2.5.0 (released)

- Reimplement `Manifest` class as a `pydantic` model and integrate with dependent code
- BREAKING: Remove the deprecated `add_file_to_manifest` method from `Manifest` class
- BREAKING: Rename `outpath` to `out_path` on the `Manifest` class `write` method

4.4 2.4.5 (released)

- Add `--ecr-image-tags` option to `libera-utils ecr-upload` CLI for tagging remote algorithm images in an ECR
- Add `validation` and `serialization` methods to the `DataProductIdentifier` and `ProcessingStepProductIdentifier` enums

4.5 2.4.4 (released)

- Allow overriding the standard docker config.json file with a minimal file for ECR uploading to prevent ECR upload permission failures from cached docker login credentials from CDK deployments

4.6 2.4.3 (released)

- Changes to ecr_upload to support programmatic building and pushing of Docker images
- Remove DynamoDB docs (moved to libera_cdk)

4.7 2.4.2 (released)

- BREAKING: Remove the AnyFilename polymorphic class. Please use AbstractValidFilename.from_file_path()

4.8 2.4.1 (released)

- Updating requirements of methods to use keyword arguments rather than positional arguments
- Adding ProcessingStepIdentifier and DataProductIdentifier to the filenamings classes
- Updating ecr names to work with the completion checker testing in libera_cdk

4.9 2.4.0 (released)

- Add properties to filenamings classes to retrieve data_product_id and processing_step_id
- Add ProcessingStepIdentifier and DataProductIdentifier standardization to be used by downstream repos

4.10 2.3.1 (released)

- Fix os.path.join bug in filenamings module that broke mocked S3 paths and also fix typehinting

4.11 2.3.0 (released)

- Create CLI tools for AWS ECR image upload and Step Function triggering
- Update manifest filenames to use ULID instead of timestamp for unique identifiers
- Change logutil configure_task_logging to optionally log JSON to console
- Allow configure_task_logging to optionally propagate DEBUG messages from specific loggers
- Update documentation for how the database is used in the Libera project in DynamoDB
- Create tools for DynamoDB in AWS for .pds files (CONS and PDS)
- Replace the use of PostgreSQL with DynamoDB for the Libera project

4.12 2.2.0 (released)

- Add AnyFilename polymorphic class
- Change filename of all products to a LiberaDataProductFilename that inherits from AnyFilename
- Update filenaming convention to be all capital letters
- Improve API for manifest module
- Add prefixing to Filename classes for predictable archive paths
- Add prefixing for manifest files for predictable and navigable paths in s3 buckets
- Update git to include lfs and move test data to lfs
- Improve database manager including caching improvements
- Improve smart_copy_file and bug fixes to smart_open testing
- Refactoring and improving pds ingest for database entries and integration testing in CDK
- Added handling of construction records and pds files appropriately when ingesting
 - This includes reading a construction record and removing the pds file entry for the construction record itself
- Improved testing of pds ingest and pds file orm models to more accurately reflect use cases
- Added output manifest creation from input manifest to match timestamps in filenames of input and output manifests
- Refactored pds ingest to use AnyPath objects for handling file locations
- Added error handling to pds ingest

4.13 2.1.1 (released)

- Update dependency specification to speed up dependency resolution wrt botocore/urllib3
- Improve database initialization to work with libera_cdk changes
- Fix bug in Dockerfile that incorrectly set the default entrypoint
- Add preliminary instrument kernel

4.14 2.1.0 (released)

- Improve API to Manifest and Manifest.add_files
- Add manifest filename enforcement to Manifest class
- Update filenaming conventions for product filenames and SPICE kernels
- Allow adding an s3 bucket/prefix as a basepath for filenames

4.15 2.0.1 (released)

- Remove the extras dependency spec because of the way SQLAlchemy imports models

4.16 2.0.0 (released)

- Add filenames classes
- Add manifest file class
- Add construction record parser
- Update DB schema to store construction records
- Update kernel generation CLI to use manifest file pattern
- Shift database and spice related libraries to extras (not installed by default)
- Add smart_copy_file function that can copy files to and from S3 and filesystem locations transparently
- Remove HDF-EOS5 filehandling code
- Add quality flag classes
- Change license to BSD3

4.17 1.0.0 (released)

- Stub out project structure
- Add build and release processes to readme
- Switch to Poetry for project dependency configuration and build management
- Add geolocation module
- Add tools in spiceutil module for caching SPICE kernels from NAIF
- Add missing unit testing coverage
- Add spice.md documentation on how the package uses and manages SPICE kernels
- Add database tooling, dev database, and ORM setup
- Add smart_open for opening local or S3 objects
- Add logging utility functions for setting up application logging

LIBERA SCIENCE DATA PROCESSING UTILITIES

Libera Utils is a package containing modules that are commonly used throughout the Libera Science Data Center codebase and processing algorithms. This package is published on PyPI to support our L2 algorithm developers with standardized code for interacting with our AWS resources and a consistent API for common tasks required of all developers.

5.1 Documentation

Documentation site, including full API listing: <https://lasp-libera-sdc-libera-utils.readthedocs-hosted.com>

Additional documentation helpful for Level 2 Algorithm Developers is also available in the Libera SDC Developer Guide. Please contact the Libera SDC Team at LASP for access to the Developer Guide.

5.2 Installation

```
pip install libera-utils
```

Other suffixed versions such as release candidate versions (version strings suffixed with rc followed by the candidate number, e.g. 1.2.3rc2) may also be available but are likely to contain bugs.

DEVELOPING LIBERA UTILS

Libera Utils is versioned formally using semantic versioning and released as new features are made available and bugs are fixed. You can find the complete release history on PyPI. Release candidate (rc) versions are also released in order for the SDC Team to test new functionality without breaking downstream code using generous dependency specifications.

We recommend pinning major and minor release versions (e.g. 2.2) as minor releases may contain minor breaking changes. Patch releases will be restricted to bug fixes that do not cause breaking changes to existing APIs.

PYTHON MODULE INDEX

|
libera_utils, 31
libera_utils.aws, 31
libera_utils.aws.constants, 32
libera_utils.aws.ecr_upload, 99
libera_utils.aws.processing_step_function_trigger,
102
libera_utils.aws.s3_utilities, 103
libera_utils.aws.utils, 105
libera_utils.backports, 106
libera_utils.backports.enum_3_11, 106
libera_utils.cli, 106
libera_utils.config, 107
libera_utils.db, 109
libera_utils.db.dynamodb_utils, 110
libera_utils.geolocation, 111
libera_utils.io, 118
libera_utils.io.caching, 118
libera_utils.io.filenaming, 119
libera_utils.io.hdf, 161
libera_utils.io.manifest, 162
libera_utils.io.smart_open, 173
libera_utils.kernel_maker, 176
libera_utils.logutil, 180
libera_utils.packets, 186
libera_utils.quality_flags, 187
libera_utils.spice_utils, 192
libera_utils.time, 204
libera_utils.version, 208

Symbols

__init__() (libera_utils.config), 109
 __init__() (libera_utils.aws.constants.CkObject method), 32
 __init__() (libera_utils.aws.constants.DataLevel method), 34
 __init__() (libera_utils.aws.constants.DataProductIdentifier method), 43
 __init__() (libera_utils.aws.constants.LiberaAccountSuffix method), 52
 __init__() (libera_utils.aws.constants.LiberaApid method), 59
 __init__() (libera_utils.aws.constants.LiberaDataBucketName method), 61
 __init__() (libera_utils.aws.constants.ManifestType method), 70
 __init__() (libera_utils.aws.constants.ProcessingStepIdentifier method), 79
 __init__() (libera_utils.aws.constants.SpKObject method), 87
 __init__() (libera_utils.aws.ecr_upload.DockerConfigManager method), 101
 __init__() (libera_utils.config.ConfigurationFormatter method), 108
 __init__() (libera_utils.io.filenaming.AbstractValidFilename method), 121
 __init__() (libera_utils.io.filenaming.AttitudeKernelFilename method), 124
 __init__() (libera_utils.io.filenaming.EphemerisKernelFilename method), 127
 __init__() (libera_utils.io.filenaming.L0Filename method), 130
 __init__() (libera_utils.io.filenaming.LiberaDataProductFilename method), 134
 __init__() (libera_utils.io.filenaming.ManifestFilename method), 137
 __init__() (libera_utils.io.filenaming.ProductName method), 141
 __init__() (libera_utils.io.manifest.Manifest method), 164
 __init__() (libera_utils.io.manifest.ManifestFileRecord method), 167
 __init__() (libera_utils.logutil.JsonLogFormatter method), 183
 __init__() (libera_utils.quality_flags.FlagBit method), 189
 __init__() (libera_utils.quality_flags.FrozenFlagMeta method), 191
 __init__() (libera_utils.quality_flags.QualityFlag method), 191
 __init__() (libera_utils.spice_utils.KernelFileCache method), 195
 __init__() (libera_utils.spice_utils.KernelFileRecord method), 197
 __init__() (libera_utils.spice_utils.SpiceBody method), 198
 __init__() (libera_utils.spice_utils.SpiceFrame method), 198
 __init__() (libera_utils.spice_utils.SpiceId method), 199
 __init__() (libera_utils.spice_utils.SpiceInstrument method), 199
 calculate_applicable_time() (libera_utils.io.filenaming.AbstractValidFilename static method), 122, 149
 _calculate_applicable_time() (libera_utils.io.filenaming.AttitudeKernelFilename static method), 124
 calculate_applicable_time() (libera_utils.io.filenaming.EphemerisKernelFilename static method), 127
 _calculate_applicable_time() (libera_utils.io.filenaming.L0Filename static method), 131
 calculate_applicable_time() (libera_utils.io.filenaming.LiberaDataProductFilename static method), 134
 _calculate_applicable_time() (libera_utils.io.filenaming.ManifestFilename static method), 138
 _copy_local_to_local() (in module libera_utils.io.smart_open), 174
 _copy_local_to_s3() (in module lib-

era_utils.io.smart_open), 174
 _copy_s3_to_local() (in module *lib-
era_utils.io.smart_open*), 174
 _copy_s3_to_s3() (in module *lib-
era_utils.io.smart_open*), 175
 _format_filename_parts() (lib-
*era_utils.io.filenaming.AbstractValidFilename
class method*), 122, 149
 _format_filename_parts() (lib-
*era_utils.io.filenaming.AttitudeKernelFilename
class method*), 125, 151
 _format_filename_parts() (lib-
*era_utils.io.filenaming.EphemerisKernelFilename
class method*), 128, 152
 _format_filename_parts() (lib-
*era_utils.io.filenaming.LOFilename
class
method*), 131, 154
 _format_filename_parts() (lib-
*era_utils.io.filenaming.LiberaDataProductFilename
class method*), 134, 156
 _format_filename_parts() (lib-
*era_utils.io.filenaming.ManifestFilename
class method*), 138, 158
 _format_return_value() (lib-
era_utils.config._ConfigurationCache method),
109
 _from_filename_parts() (lib-
*era_utils.io.filenaming.AbstractValidFilename
class method*), 122, 149
 _from_filename_parts() (lib-
*era_utils.io.filenaming.AttitudeKernelFilename
class method*), 125
 _from_filename_parts() (lib-
*era_utils.io.filenaming.EphemerisKernelFilename
class method*), 128
 _from_filename_parts() (lib-
*era_utils.io.filenaming.LOFilename
class
method*), 131
 _from_filename_parts() (lib-
*era_utils.io.filenaming.LiberaDataProductFilename
class method*), 135
 _from_filename_parts() (lib-
*era_utils.io.filenaming.ManifestFilename
class method*), 138
 _generate_filename() (lib-
era_utils.io.manifest.Manifest method), 165,
170
 _json_serialize_default() (in module *lib-
era_utils.logutil*), 184
 _parse_filename_parts() (lib-
*era_utils.io.filenaming.AbstractValidFilename
method*), 122, 149
 _parse_filename_parts() (lib-
*era_utils.io.filenaming.AttitudeKernelFilename
method*), 125, 151
 _parse_filename_parts() (lib-
*era_utils.io.filenaming.EphemerisKernelFilename
method*), 128, 153
 _parse_filename_parts() (lib-
*era_utils.io.filenaming.LOFilename
method*), 132, 155
 _parse_filename_parts() (lib-
*era_utils.io.filenaming.LiberaDataProductFilename
method*), 135, 156
 _parse_filename_parts() (lib-
*era_utils.io.filenaming.ManifestFilename
method*), 138, 158
 _parse_numeric_types() (lib-
era_utils.config._ConfigurationCache method),
109
A
 AbstractValidFilename (class in *lib-
era_utils.io.filenaming*), 121, 148
 add_archive_time_to_ddb_item() (in module *lib-
era_utils.db.dynamodb_utils*), 110
 add_desired_time_range() (lib-
era_utils.io.manifest.Manifest method), 165,
170
 add_files() (*libera_utils.io.manifest.Manifest method*),
165, 170
 angle_between() (in module *libera_utils.geolocation*),
111, 115
 archive_prefix (*libera_utils.io.filenaming.AbstractValidFilename
property*), 122, 150
 archive_prefix (*libera_utils.io.filenaming.AttitudeKernelFilename
property*), 125, 151
 archive_prefix (*libera_utils.io.filenaming.EphemerisKernelFilename
property*), 128, 153
 archive_prefix (*libera_utils.io.filenaming.LOFilename
property*), 132, 155
 archive_prefix (*libera_utils.io.filenaming.LiberaDataProductFilename
property*), 135, 157
 archive_prefix (*libera_utils.io.filenaming.ManifestFilename
property*), 139, 158
 array_from_packets() (in module *lib-
era_utils.packets*), 186, 187
 as_integer_ratio() (lib-
era_utils.quality_flags.FlagBit method),
189
 AttitudeKernelFilename (class in *lib-
era_utils.io.filenaming*), 123, 150
B
 bit_count() (*libera_utils.quality_flags.FlagBit
method*), 189
 bit_length() (*libera_utils.quality_flags.FlagBit
method*), 189

- build_docker_image() (in module *libera_utils.aws.ecr_upload*), 99, 101
- ## C
- cache_dir (*libera_utils.spice_utils.KernelFileCache* property), 196, 200
- calculate_checksum() (in module *libera_utils.io.manifest*), 162, 172
- capitalize() (*libera_utils.aws.constants.DataLevel* method), 36
- capitalize() (*libera_utils.aws.constants.DataProductIdentifier* method), 45
- capitalize() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 54
- capitalize() (*libera_utils.aws.constants.LiberaDataBucketName* method), 63
- capitalize() (*libera_utils.aws.constants.ManifestType* method), 72
- capitalize() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 81
- capitalize() (*libera_utils.io.filenaming.ProductName* method), 143
- cartesian_to_planetographic() (in module *libera_utils.geolocation*), 112, 115
- casefold() (*libera_utils.aws.constants.DataLevel* method), 36
- casefold() (*libera_utils.aws.constants.DataProductIdentifier* method), 45
- casefold() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 54
- casefold() (*libera_utils.aws.constants.LiberaDataBucketName* method), 63
- casefold() (*libera_utils.aws.constants.ManifestType* method), 72
- casefold() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 81
- casefold() (*libera_utils.io.filenaming.ProductName* method), 144
- center() (*libera_utils.aws.constants.DataLevel* method), 36
- center() (*libera_utils.aws.constants.DataProductIdentifier* method), 45
- center() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 54
- center() (*libera_utils.aws.constants.LiberaDataBucketName* method), 63
- center() (*libera_utils.aws.constants.ManifestType* method), 72
- center() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 81
- center() (*libera_utils.io.filenaming.ProductName* method), 144
- check_file_structure() (*libera_utils.io.manifest.Manifest* class method), 165, 170
- CkObject (class in *libera_utils.aws.constants*), 32, 87
- clear() (*libera_utils.spice_utils.KernelFileCache* method), 196, 200
- config (in module *libera_utils.config*), 107, 109
- ConfigurationFormatter (class in *libera_utils.config*), 107, 108
- configure_static_logging() (in module *libera_utils.logutil*), 181, 184
- configure_task_logging() (in module *libera_utils.logutil*), 181, 184
- conjugate() (*libera_utils.quality_flags.FlagBit* method), 190
- convert_cds_integer_to_datetime() (in module *libera_utils.time*), 204, 206
- count() (*libera_utils.aws.constants.DataLevel* method), 37
- count() (*libera_utils.aws.constants.DataProductIdentifier* method), 46
- count() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 54
- count() (*libera_utils.aws.constants.LiberaDataBucketName* method), 63
- count() (*libera_utils.aws.constants.ManifestType* method), 72
- count() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 81
- count() (*libera_utils.io.filenaming.ProductName* method), 144
- count() (*libera_utils.spice_utils.KernelFileRecord* method), 197
- count() (*libera_utils.spice_utils.SpiceId* method), 199
- create_ddb_metadata_applicable_date_item() (in module *libera_utils.db.dynamodb_utils*), 110
- create_ddb_metadata_file_item() (in module *libera_utils.db.dynamodb_utils*), 110, 111
- ## D
- data_product_id (*libera_utils.aws.constants.CkObject* property), 32, 87
- data_product_id (*libera_utils.aws.constants.SpkiObject* property), 87, 99
- data_product_id (*libera_utils.io.filenaming.AbstractValidFilename* property), 122, 150
- data_product_id (*libera_utils.io.filenaming.AttitudeKernelFilename* property), 125, 151
- data_product_id (*libera_utils.io.filenaming.EphemerisKernelFilename* property), 128, 153

data_product_id (*libera_utils.io.filenaming.LOFilename* property), 132, 155
data_product_id (*libera_utils.io.filenaming.LiberaDataProductFilename* property), 135, 157
data_product_id (*libera_utils.io.filenaming.ManifestFilename* property), 139, 158
data_product_id (*libera_utils.io.filenaming.ProductName* property), 144, 160
DataLevel (class in *libera_utils.aws.constants*), 33, 87
DataProductIdentifier (class in *libera_utils.aws.constants*), 41, 89
denominator (*libera_utils.quality_flags.FlagBit* attribute), 190
DockerConfigManager (class in *libera_utils.aws.ecr_upload*), 101
download_kernel() (*libera_utils.spice_utils.KernelFileCache* method), 196, 200
dump() (*libera_utils.aws.constants.DataProductIdentifier* method), 46, 91
dump() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 81, 98

E

ecr_name (*libera_utils.aws.constants.ProcessingStepIdentifier* property), 82, 98
ecr_upload_cli_func() (in module *libera_utils.aws.ecr_upload*), 100, 101
empty_local_cache_dir() (in module *libera_utils.io.caching*), 119
encode() (*libera_utils.aws.constants.DataLevel* method), 37
encode() (*libera_utils.aws.constants.DataProductIdentifier* method), 46
encode() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 54
encode() (*libera_utils.aws.constants.LiberaDataBucketName* method), 64
encode() (*libera_utils.aws.constants.ManifestType* method), 72
encode() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 82
encode() (*libera_utils.io.filenaming.ProductName* method), 144
endswith() (*libera_utils.aws.constants.DataLevel* method), 37
endswith() (*libera_utils.aws.constants.DataProductIdentifier* method), 46
endswith() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 55
endswith() (*libera_utils.aws.constants.LiberaDataBucketName* method), 64
endswith() (*libera_utils.aws.constants.ManifestType* method), 72
endswith() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 82
endswith() (*libera_utils.io.filenaming.ProductName* method), 144
ensure_spice() (in module *libera_utils.spice_utils*), 193, 202
EphemerisKernelFilename (class in *libera_utils.io.filenaming*), 126, 152
et2utc_wrapper() (in module *libera_utils.time*), 204, 206
et_2_datetime() (in module *libera_utils.time*), 205, 207
et_2_timestamp() (in module *libera_utils.time*), 205, 207
expandtabs() (*libera_utils.aws.constants.DataLevel* method), 37
expandtabs() (*libera_utils.aws.constants.DataProductIdentifier* method), 46
expandtabs() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 55
expandtabs() (*libera_utils.aws.constants.LiberaDataBucketName* method), 64
expandtabs() (*libera_utils.aws.constants.ManifestType* method), 72
expandtabs() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 82
expandtabs() (*libera_utils.io.filenaming.ProductName* method), 144

F

file_name (*libera_utils.spice_utils.KernelFileRecord* attribute), 197, 201
filename_parts (*libera_utils.io.filenaming.AbstractValidFilename* property), 123, 150
filename_parts (*libera_utils.io.filenaming.AttitudeKernelFilename* property), 125
filename_parts (*libera_utils.io.filenaming.EphemerisKernelFilename* property), 129
filename_parts (*libera_utils.io.filenaming.LOFilename* property), 132
filename_parts (*libera_utils.io.filenaming.LiberaDataProductFilename* property), 135
filename_parts (*libera_utils.io.filenaming.ManifestFilename* property), 139
find() (*libera_utils.aws.constants.DataLevel* method), 37
find() (*libera_utils.aws.constants.DataProductIdentifier* method), 46
find() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 55

era_utils.io.filenaming.LiberaDataProductFilenames_cached() (*libera_utils.spice_utils.KernelFileCache method*), 136

generate_prefixed_path() (*libera_utils.io.filenaming.ManifestFilename method*), 139

get() (*libera_utils.config._ConfigurationCache method*), 109

get_archive_bucket_name() (*libera_utils.aws.constants.ProcessingStepIdentifier method*), 82, 98

get_aws_account_number() (*in module libera_utils.aws.utils*), 105

get_current_revision_str() (*in module libera_utils.io.filenaming*), 120, 161

get_current_version_str() (*in module libera_utils.io.filenaming*), 120, 161

get_dynamodb_table() (*in module libera_utils.db.dynamodb_utils*), 110, 111

get_earth_radii() (*in module libera_utils.geolocation*), 112, 116

get_ecr_docker_client() (*in module libera_utils.aws.ecr_upload*), 100, 102

get_local_cache_dir() (*in module libera_utils.io.caching*), 119

get_spice_packet_data_from_filepaths() (*in module libera_utils.kernel_maker*), 177, 179

get_ulid_code() (*in module libera_utils.io.manifest*), 162, 172

get_value() (*libera_utils.config.ConfigurationFormatter method*), 108

H

h5dump() (*in module libera_utils.io.hdf*), 161, 162

I

imag (*libera_utils.quality_flags.FlagBit attribute*), 190

index() (*libera_utils.aws.constants.DataLevel method*), 37

index() (*libera_utils.aws.constants.DataProductIdentifier method*), 46

index() (*libera_utils.aws.constants.LiberaAccountSuffix method*), 55

index() (*libera_utils.aws.constants.LiberaDataBucketName method*), 64

index() (*libera_utils.aws.constants.ManifestType method*), 73

index() (*libera_utils.aws.constants.ProcessingStepIdentifier method*), 83

index() (*libera_utils.io.filenaming.ProductName method*), 144

index() (*libera_utils.spice_utils.KernelFileRecord method*), 198

index() (*libera_utils.spice_utils.SpiceId method*), 199

is_cached() (*libera_utils.spice_utils.KernelFileCache method*), 197, 200

is_gzip() (*in module libera_utils.io.smart_open*), 173, 175

is_s3() (*in module libera_utils.io.smart_open*), 173, 175

isalnum() (*libera_utils.aws.constants.DataLevel method*), 37

isalnum() (*libera_utils.aws.constants.DataProductIdentifier method*), 46

isalnum() (*libera_utils.aws.constants.LiberaAccountSuffix method*), 55

isalnum() (*libera_utils.aws.constants.LiberaDataBucketName method*), 64

isalnum() (*libera_utils.aws.constants.ManifestType method*), 73

isalnum() (*libera_utils.aws.constants.ProcessingStepIdentifier method*), 83

isalnum() (*libera_utils.io.filenaming.ProductName method*), 144

isalpha() (*libera_utils.aws.constants.DataLevel method*), 37

isalpha() (*libera_utils.aws.constants.DataProductIdentifier method*), 47

isalpha() (*libera_utils.aws.constants.LiberaAccountSuffix method*), 55

isalpha() (*libera_utils.aws.constants.LiberaDataBucketName method*), 64

isalpha() (*libera_utils.aws.constants.ManifestType method*), 73

isalpha() (*libera_utils.aws.constants.ProcessingStepIdentifier method*), 83

isalpha() (*libera_utils.io.filenaming.ProductName method*), 145

isascii() (*libera_utils.aws.constants.DataLevel method*), 37

isascii() (*libera_utils.aws.constants.DataProductIdentifier method*), 47

isascii() (*libera_utils.aws.constants.LiberaAccountSuffix method*), 55

isascii() (*libera_utils.aws.constants.LiberaDataBucketName method*), 64

isascii() (*libera_utils.aws.constants.ManifestType method*), 73

isascii() (*libera_utils.aws.constants.ProcessingStepIdentifier method*), 83

isascii() (*libera_utils.io.filenaming.ProductName method*), 145

isdecimal() (*libera_utils.aws.constants.DataLevel method*), 38

isdecimal() (*libera_utils.aws.constants.DataProductIdentifier method*), 47

isdecimal() (*libera_utils.aws.constants.LiberaAccountSuffix method*), 55

isdecimal() (libera_utils.aws.constants.LiberaDataBucketName method), 64
 isdecimal() (libera_utils.aws.constants.ManifestType method), 73
 isdecimal() (libera_utils.aws.constants.ProcessingStepIdentifier method), 83
 isdecimal() (libera_utils.io.filenaming.ProductName method), 145
 isdigit() (libera_utils.aws.constants.DataLevel method), 38
 isdigit() (libera_utils.aws.constants.DataProductIdentifier method), 47
 isdigit() (libera_utils.aws.constants.LiberaAccountSuffix method), 56
 isdigit() (libera_utils.aws.constants.LiberaDataBucketName method), 65
 isdigit() (libera_utils.aws.constants.ManifestType method), 73
 isdigit() (libera_utils.aws.constants.ProcessingStepIdentifier method), 83
 isdigit() (libera_utils.io.filenaming.ProductName method), 145
 isidentifier() (libera_utils.aws.constants.DataLevel method), 38
 isidentifier() (libera_utils.aws.constants.DataProductIdentifier method), 47
 isidentifier() (libera_utils.aws.constants.LiberaAccountSuffix method), 56
 isidentifier() (libera_utils.aws.constants.LiberaDataBucketName method), 65
 isidentifier() (libera_utils.aws.constants.ManifestType method), 73
 isidentifier() (libera_utils.aws.constants.ProcessingStepIdentifier method), 83
 isidentifier() (libera_utils.io.filenaming.ProductName method), 145
 islower() (libera_utils.aws.constants.DataLevel method), 38
 islower() (libera_utils.aws.constants.DataProductIdentifier method), 47
 islower() (libera_utils.aws.constants.LiberaAccountSuffix method), 56
 islower() (libera_utils.aws.constants.LiberaDataBucketName method), 65
 islower() (libera_utils.aws.constants.ManifestType method), 73
 islower() (libera_utils.aws.constants.ProcessingStepIdentifier method), 83
 islower() (libera_utils.io.filenaming.ProductName method), 145
 isnumeric() (libera_utils.aws.constants.DataLevel method), 38
 isnumeric() (libera_utils.aws.constants.DataProductIdentifier method), 47
 isnumeric() (libera_utils.aws.constants.LiberaAccountSuffix method), 56
 isnumeric() (libera_utils.aws.constants.LiberaDataBucketName method), 65
 isnumeric() (libera_utils.aws.constants.ManifestType method), 73
 isnumeric() (libera_utils.aws.constants.ProcessingStepIdentifier method), 83
 isnumeric() (libera_utils.io.filenaming.ProductName method), 145
 isprintable() (libera_utils.aws.constants.DataLevel method), 38
 isprintable() (libera_utils.aws.constants.DataProductIdentifier method), 47
 isprintable() (libera_utils.aws.constants.LiberaAccountSuffix method), 56
 isprintable() (libera_utils.aws.constants.LiberaDataBucketName method), 65
 isprintable() (libera_utils.aws.constants.ManifestType method), 74
 isprintable() (libera_utils.aws.constants.ProcessingStepIdentifier method), 83
 isprintable() (libera_utils.io.filenaming.ProductName method), 145
 isspace() (libera_utils.aws.constants.DataLevel method), 38
 isspace() (libera_utils.aws.constants.DataProductIdentifier method), 47
 isspace() (libera_utils.aws.constants.LiberaAccountSuffix method), 56
 isspace() (libera_utils.aws.constants.LiberaDataBucketName method), 65
 isspace() (libera_utils.aws.constants.ManifestType method), 74
 isspace() (libera_utils.aws.constants.ProcessingStepIdentifier method), 83
 isspace() (libera_utils.io.filenaming.ProductName method), 145
 istitle() (libera_utils.aws.constants.DataLevel method), 38
 istitle() (libera_utils.aws.constants.DataProductIdentifier method), 47
 istitle() (libera_utils.aws.constants.LiberaAccountSuffix method), 56
 istitle() (libera_utils.aws.constants.LiberaDataBucketName method), 65
 istitle() (libera_utils.aws.constants.ManifestType method), 74
 istitle() (libera_utils.aws.constants.ProcessingStepIdentifier method), 84
 istitle() (libera_utils.io.filenaming.ProductName method), 145
 isupper() (libera_utils.aws.constants.DataLevel method), 38

- isupper() (*libera_utils.aws.constants.DataProductIdentifier* module, 99
method), 47 *libera_utils.aws.processing_step_function_trigger*
- isupper() (*libera_utils.aws.constants.LiberaAccountSuffix* module, 102
method), 56 *libera_utils.aws.s3_utilities*
- isupper() (*libera_utils.aws.constants.LiberaDataBucketName* module, 103
method), 65 *libera_utils.aws.utils*
- isupper() (*libera_utils.aws.constants.ManifestType* module, 105
method), 74 *libera_utils.backports*
- isupper() (*libera_utils.aws.constants.ProcessingStepIdentifier* module, 106
method), 84 *libera_utils.backports.enum_3_11*
- isupper() (*libera_utils.io.filenaming.ProductName* module, 106
method), 145 *libera_utils.cli*
module, 106
- ## J
- join() (*libera_utils.aws.constants.DataLevel* method),
38 *libera_utils.config*
module, 107
- join() (*libera_utils.aws.constants.DataProductIdentifier*
method), 48 *libera_utils.db*
module, 109
- join() (*libera_utils.aws.constants.LiberaAccountSuffix*
method), 56 *libera_utils.db.dynamodb_utils*
module, 110
- join() (*libera_utils.aws.constants.LiberaDataBucketName*
method), 65 *libera_utils.geolocation*
module, 111
- join() (*libera_utils.aws.constants.ManifestType*
method), 74 *libera_utils.io*
module, 118
- join() (*libera_utils.aws.constants.ProcessingStepIdentifier*
method), 84 *libera_utils.io.caching*
module, 118
- join() (*libera_utils.io.filenaming.ProductName*
method), 146 *libera_utils.io.filenaming*
module, 119
- JsonLogFormatter (class in *libera_utils.logutil*), 183,
184 *libera_utils.io.hdf*
module, 161
- ## K
- kernel_basename (*libera_utils.spice_utils.KernelFileCache* prop-
erty), 197, 201 *libera_utils.io.manifest*
module, 162
- kernel_path (*libera_utils.spice_utils.KernelFileCache*
property), 197, 201 *libera_utils.io.smart_open*
module, 173
- kernel_type (*libera_utils.spice_utils.KernelFileRecord*
attribute), 198, 201 *libera_utils.kernel_maker*
module, 176
- KernelFileCache (class in *libera_utils.spice_utils*),
195, 199 *libera_utils.logutil*
module, 180
- KernelFileRecord (class in *libera_utils.spice_utils*),
197, 201 *libera_utils.packets*
module, 186
- libera_utils.quality_flags*
module, 187
- libera_utils.spice_utils*
module, 192
- libera_utils.time*
module, 204
- ## L
- L0Filename (class in *libera_utils.io.filenaming*), 130,
154 *libera_utils.version*
module, 208
- libera_utils*
module, 31 *LiberaAccountSuffix* (class in *lib-*
era_utils.aws.constants), 50, 91
- libera_utils.aws*
module, 31 *LiberaApid* (class in *libera_utils.aws.constants*), 59, 93
- libera_utils.aws.constants*
module, 32 *LiberaDataBucketName* (class in *lib-*
era_utils.aws.constants), 59, 93
- libera_utils.aws.ecr_upload* *LiberaDataProductFilename* (class in *lib-*
era_utils.io.filenaming), 133, 155

ljust() (*libera_utils.aws.constants.DataLevel* method), 39
ljust() (*libera_utils.aws.constants.DataProductIdentifier* method), 48
ljust() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 56
ljust() (*libera_utils.aws.constants.LiberaDataBucketName* method), 65
ljust() (*libera_utils.aws.constants.ManifestType* method), 74
ljust() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 84
ljust() (*libera_utils.io.filenaming.ProductName* method), 146
lower() (*libera_utils.aws.constants.DataLevel* method), 39
lower() (*libera_utils.aws.constants.DataProductIdentifier* method), 48
lower() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 57
lower() (*libera_utils.aws.constants.LiberaDataBucketName* method), 66
lower() (*libera_utils.aws.constants.ManifestType* method), 74
lower() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 84
lower() (*libera_utils.io.filenaming.ProductName* method), 146
ls_kernel_coverage() (in module *libera_utils.spice_utils*), 194, 203
ls_kernels() (in module *libera_utils.spice_utils*), 194, 203
ls_spice_constants() (in module *libera_utils.spice_utils*), 194, 203
lstrip() (*libera_utils.aws.constants.DataLevel* method), 39
lstrip() (*libera_utils.aws.constants.DataProductIdentifier* method), 48
lstrip() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 57
lstrip() (*libera_utils.aws.constants.LiberaDataBucketName* method), 66
lstrip() (*libera_utils.aws.constants.ManifestType* method), 74
lstrip() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 84
lstrip() (*libera_utils.io.filenaming.ProductName* method), 146
M
main() (in module *libera_utils.cli*), 106, 107
make_azel_ck() (in module *libera_utils.kernel_maker*), 177, 179
make_jpss_ck() (in module *libera_utils.kernel_maker*), 177, 179
make_jpss_kernels_from_manifest() (in module *libera_utils.kernel_maker*), 177, 179
make_jpss_spk() (in module *libera_utils.kernel_maker*), 178, 179
maketrans() (*libera_utils.aws.constants.DataLevel* static method), 39
maketrans() (*libera_utils.aws.constants.DataProductIdentifier* static method), 48
maketrans() (*libera_utils.aws.constants.LiberaAccountSuffix* static method), 57
maketrans() (*libera_utils.aws.constants.LiberaDataBucketName* static method), 66
maketrans() (*libera_utils.aws.constants.ManifestType* static method), 74
maketrans() (*libera_utils.io.filenaming.ProductName* static method), 146
Manifest (class in *libera_utils.io.manifest*), 162, 168
ManifestError, 168, 171
ManifestFilename (class in *libera_utils.io.filenaming*), 137, 157
ManifestFileRecord (class in *libera_utils.io.manifest*), 167, 171
ManifestType (class in *libera_utils.aws.constants*), 68, 95
model_config (*libera_utils.io.manifest.Manifest* attribute), 165, 170
model_config (*libera_utils.io.manifest.ManifestFileRecord* attribute), 168, 172
model_extra (*libera_utils.io.manifest.Manifest* property), 166
model_extra (*libera_utils.io.manifest.ManifestFileRecord* property), 168
model_fields_set (*libera_utils.io.manifest.Manifest* property), 166
model_fields_set (*libera_utils.io.manifest.ManifestFileRecord* property), 168
module
libera_utils, 31
libera_utils.aws, 31
libera_utils.aws.constants, 32
libera_utils.aws.ecr_upload, 99
libera_utils.aws.processing_step_function_trigger, 102
libera_utils.aws.s3_utilities, 103
libera_utils.aws.utils, 105
libera_utils.backports, 106
libera_utils.backports.enum_3_11, 106
libera_utils.cli, 106
libera_utils.config, 107

- libera_utils.db, 109
 - libera_utils.db.dynamodb_utils, 110
 - libera_utils.geolocation, 111
 - libera_utils.io, 118
 - libera_utils.io.caching, 118
 - libera_utils.io.filenaming, 119
 - libera_utils.io.hdf, 161
 - libera_utils.io.manifest, 162
 - libera_utils.io.smart_open, 173
 - libera_utils.kernel_maker, 176
 - libera_utils.logutil, 180
 - libera_utils.packets, 186
 - libera_utils.quality_flags, 187
 - libera_utils.spice_utils, 192
 - libera_utils.time, 204
 - libera_utils.version, 208
 - mro() (*libera_utils.quality_flags.FrozenFlagMeta* method), 191
- N**
- numerator (*libera_utils.quality_flags.FlagBit* attribute), 190
 - numid (*libera_utils.spice_utils.SpiceId* attribute), 199, 201
- O**
- output_manifest_from_input_manifest() (*libera_utils.io.manifest.Manifest* class method), 166, 171
- P**
- parse_cli_args() (*in module libera_utils.cli*), 106, 107
 - parse_packets() (*in module libera_utils.packets*), 186, 187
 - partition() (*libera_utils.aws.constants.DataLevel* method), 39
 - partition() (*libera_utils.aws.constants.DataProductIdentifier* method), 48
 - partition() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 57
 - partition() (*libera_utils.aws.constants.LiberaDataBucketName* method), 66
 - partition() (*libera_utils.aws.constants.ManifestType* method), 74
 - partition() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 84
 - partition() (*libera_utils.io.filenaming.ProductName* method), 146
 - path (*libera_utils.io.filenaming.AbstractValidFilename* property), 123, 150
 - path (*libera_utils.io.filenaming.AttitudeKernelFilename* property), 126
 - path (*libera_utils.io.filenaming.EphemerisKernelFilename* property), 129
 - path (*libera_utils.io.filenaming.LOFilename* property), 133
 - path (*libera_utils.io.filenaming.LiberaDataProductFilename* property), 136
 - path (*libera_utils.io.filenaming.ManifestFilename* property), 139
 - print_version_info() (*in module libera_utils.cli*), 107
 - processing_step_id (*libera_utils.aws.constants.CkObject* property), 32, 87
 - processing_step_id (*libera_utils.aws.constants.SpikObject* property), 87, 99
 - processing_step_id (*libera_utils.io.filenaming.AbstractValidFilename* property), 123, 150
 - processing_step_id (*libera_utils.io.filenaming.AttitudeKernelFilename* property), 126, 152
 - processing_step_id (*libera_utils.io.filenaming.EphemerisKernelFilename* property), 129, 153
 - processing_step_id (*libera_utils.io.filenaming.LOFilename* property), 133, 155
 - processing_step_id (*libera_utils.io.filenaming.LiberaDataProductFilename* property), 136, 157
 - processing_step_id (*libera_utils.io.filenaming.ManifestFilename* property), 139, 159
 - processing_step_id (*libera_utils.io.filenaming.ProductName* property), 146, 160
 - ProcessingStepIdentifier (class *in libera_utils.aws.constants*), 77, 96
 - ProductName (class *in libera_utils.io.filenaming*), 140, 159
 - push_image_to_ecr() (*in module libera_utils.aws.ecr_upload*), 100, 102
- Q**
- QualityFlag (class *in libera_utils.quality_flags*), 191, 192
- R**
- real (*libera_utils.quality_flags.FlagBit* attribute), 190
 - regex_match() (*libera_utils.io.filenaming.AbstractValidFilename* method), 123, 150
 - regex_match() (*libera_utils.io.filenaming.AttitudeKernelFilename* method), 126

rsplit() (*libera_utils.aws.constants.DataProductIdentifierSpkObject* (class in *libera_utils.aws.constants*), 87, 99 method), 49
rsplit() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 58
rsplit() (*libera_utils.aws.constants.LiberaDataBucketName* method), 67
rsplit() (*libera_utils.aws.constants.ManifestType* method), 75
rsplit() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 85
rsplit() (*libera_utils.io.filenaming.ProductName* method), 147
rstrip() (*libera_utils.aws.constants.DataLevel* method), 40
rstrip() (*libera_utils.aws.constants.DataProductIdentifier* method), 49
rstrip() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 58
rstrip() (*libera_utils.aws.constants.LiberaDataBucketName* method), 67
rstrip() (*libera_utils.aws.constants.ManifestType* method), 76
rstrip() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 85
rstrip() (*libera_utils.io.filenaming.ProductName* method), 147

S
s3_copy_cli_handler() (in module *libera_utils.aws.s3_utilities*), 103, 104
s3_list_archive_files() (in module *libera_utils.aws.s3_utilities*), 103, 104
s3_list_cli_handler() (in module *libera_utils.aws.s3_utilities*), 104, 105
s3_put_cli_handler() (in module *libera_utils.aws.s3_utilities*), 104, 105
s3_put_in_archive_for_processing_step() (in module *libera_utils.aws.s3_utilities*), 104, 105
sce2s_wrapper() (in module *libera_utils.time*), 205, 207
scs2e_wrapper() (in module *libera_utils.time*), 206, 207
serialize_filename() (*libera_utils.io.manifest.Manifest* method), 166, 171
smart_copy_file() (in module *libera_utils.io.smart_open*), 173, 175
smart_open() (in module *libera_utils.io.smart_open*), 174, 176
SpiceBody (class in *libera_utils.spice_utils*), 198, 201
SpiceFrame (class in *libera_utils.spice_utils*), 198, 201
SpiceId (class in *libera_utils.spice_utils*), 199, 201
SpiceInstrument (class in *libera_utils.spice_utils*), 199, 201

split() (*libera_utils.aws.constants.DataLevel* method), 40
split() (*libera_utils.aws.constants.DataProductIdentifier* method), 49
split() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 58
split() (*libera_utils.aws.constants.LiberaDataBucketName* method), 67
split() (*libera_utils.aws.constants.ManifestType* method), 76
split() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 85
split() (*libera_utils.io.filenaming.ProductName* method), 147
splitlines() (*libera_utils.aws.constants.DataLevel* method), 40
splitlines() (*libera_utils.aws.constants.DataProductIdentifier* method), 49
splitlines() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 58
splitlines() (*libera_utils.aws.constants.LiberaDataBucketName* method), 67
splitlines() (*libera_utils.aws.constants.ManifestType* method), 76
splitlines() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 86
splitlines() (*libera_utils.io.filenaming.ProductName* method), 148
startswith() (*libera_utils.aws.constants.DataLevel* method), 40
startswith() (*libera_utils.aws.constants.DataProductIdentifier* method), 50
startswith() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 58
startswith() (*libera_utils.aws.constants.LiberaDataBucketName* method), 67
startswith() (*libera_utils.aws.constants.ManifestType* method), 76
startswith() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 86
startswith() (*libera_utils.io.filenaming.ProductName* method), 148
step_function_trigger() (in module *libera_utils.aws.processing_step_function_trigger*), 103
strid (*libera_utils.spice_utils.SpiceId* attribute), 199, 201
strip() (*libera_utils.aws.constants.DataLevel* method), 40
strip() (*libera_utils.aws.constants.DataProductIdentifier* method), 50
strip() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 58

strip() (*libera_utils.aws.constants.LiberaDataBucketName* method), 67

strip() (*libera_utils.aws.constants.ManifestType* method), 76

strip() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 86

strip() (*libera_utils.io.filenaming.ProductName* method), 148

sub_observer_point() (in module *libera_utils.geolocation*), 113, 116

sub_solar_point() (in module *libera_utils.geolocation*), 113, 116

surface_intercept_point() (in module *libera_utils.geolocation*), 114, 117

swapcase() (*libera_utils.aws.constants.DataLevel* method), 40

swapcase() (*libera_utils.aws.constants.DataProductIdentifier* method), 50

swapcase() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 58

swapcase() (*libera_utils.aws.constants.LiberaDataBucketName* method), 67

swapcase() (*libera_utils.aws.constants.ManifestType* method), 76

swapcase() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 86

swapcase() (*libera_utils.io.filenaming.ProductName* method), 148

T

target_position() (in module *libera_utils.geolocation*), 114, 117

title() (*libera_utils.aws.constants.DataLevel* method), 41

title() (*libera_utils.aws.constants.DataProductIdentifier* method), 50

title() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 59

title() (*libera_utils.aws.constants.LiberaDataBucketName* method), 68

title() (*libera_utils.aws.constants.ManifestType* method), 76

title() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 86

title() (*libera_utils.io.filenaming.ProductName* method), 148

to_bytes() (*libera_utils.quality_flags.FlagBit* method), 190

transform_filename() (*libera_utils.io.manifest.Manifest* class method), 166, 171

transform_files() (*libera_utils.io.manifest.Manifest* class method), 166, 171

translate() (*libera_utils.aws.constants.DataLevel* method), 41

translate() (*libera_utils.aws.constants.DataProductIdentifier* method), 50

translate() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 59

translate() (*libera_utils.aws.constants.LiberaDataBucketName* method), 68

translate() (*libera_utils.aws.constants.ManifestType* method), 76

translate() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 86

translate() (*libera_utils.io.filenaming.ProductName* method), 148

U

upper() (*libera_utils.aws.constants.DataLevel* method), 41

upper() (*libera_utils.aws.constants.DataProductIdentifier* method), 50

upper() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 59

upper() (*libera_utils.aws.constants.LiberaDataBucketName* method), 68

upper() (*libera_utils.aws.constants.ManifestType* method), 76

upper() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 86

upper() (*libera_utils.io.filenaming.ProductName* method), 148

utc2et_wrapper() (in module *libera_utils.time*), 206, 207

V

validate() (*libera_utils.aws.constants.DataProductIdentifier* class method), 50, 91

validate() (*libera_utils.aws.constants.ProcessingStepIdentifier* class method), 86, 99

validate_checksums() (*libera_utils.io.manifest.Manifest* method), 166, 171

version() (in module *libera_utils.version*), 208

W

with_all_none() (in module *libera_utils.quality_flags*), 188, 192

write() (*libera_utils.io.manifest.Manifest* method), 166, 171

write_kernel_input_file() (in module *libera_utils.kernel_maker*), 178, 180

write_kernel_setup_file() (in module *libera_utils.kernel_maker*), 178, 180

Z

- zfill() (*libera_utils.aws.constants.DataLevel* method), 41
- zfill() (*libera_utils.aws.constants.DataProductIdentifier* method), 50
- zfill() (*libera_utils.aws.constants.LiberaAccountSuffix* method), 59
- zfill() (*libera_utils.aws.constants.LiberaDataBucketName* method), 68
- zfill() (*libera_utils.aws.constants.ManifestType* method), 77
- zfill() (*libera_utils.aws.constants.ProcessingStepIdentifier* method), 86
- zfill() (*libera_utils.io.filenaming.ProductName* method), 148