
pytokio Documentation

Release 0.21.0.dev1

Glenn K. Lockwood

May 18, 2022

Contents:

1 Quick Start	3
1.1 Installation	4
1.2 Architecture	7
1.3 Command Line Tools	12
1.4 TOKIO Time Series Format	12
1.5 tokio package	15
1.6 pytokio Release Process	92
2 Indices and tables	95
3 Copyright	97
Python Module Index	99
Index	101

pytokio is a Python library that provides the APIs necessary to develop analysis routines that combine data from different I/O monitoring tools that may be available in your HPC data center. The design and capabilities of pytokio have been documented in the [pytokio architecture paper](#) presented at the 2018 Cray User Group.

CHAPTER 1

Quick Start

Step 1. Download pytokio: Download the latest pytokio from the [pytokio release page](#) and unpack it somewhere:

```
$ wget https://github.com/NERSC/pytokio/releases/download/v0.10.1/pytokio-0.10.1.tar.  
→gz  
$ tar -zxf pytokio-0.10.1.tar.gz
```

Step 2. (Optional): Configure 'site.json': pytokio ships with a `site.json` configuration file that's located in the tarball's `tokio/` subdirectory. You can edit this to reflect the location of various data sources and configurations on your system:

```
$ vi pytokio-0.10.1/tokio/site.json  
...
```

However it is also perfectly fine to not worry about this now, as this file is only used for higher-level interfaces.

Step 3. Install pytokio: Install the pytokio package using your favorite package installation mechanism:

```
$ ls  
pytokio-0.10.1      pytokio-0.10.1.tar.gz  
  
$ pip install pytokio-0.10.1/
```

or:

```
$ cd pytokio-0.10.1/  
$ python setup.py install --prefix=/path/to/installdir
```

or:

```
$ cd pytokio-0.10.1/  
$ pip install --user .
```

Alternatively, pytokio does not technically require a proper installation and it is sufficient to clone the git repo, add it to `PYTHONPATH`, and `import tokio` from there:

```
$ cd pytokio-0.10.1/
$ export PYTHONPATH=$PYTHONPATH:`pwd`
```

Then verify that pytokio can be imported:

```
$ python
>>> import tokio
>>> tokio.__version__
'0.10.1'
```

pytokio supports both Python 2.7 and 3.6 and, at minimum, requires h5py, numpy, and pandas. The full requirements are listed in `requirements.txt`.

Step 4. (Optional) Test pytokio CLI tools: pytokio includes some basic CLI wrappers around many of its interfaces which are installed in your Python package install directory's `bin/` directory:

```
$ export PATH=$PATH:/path/to/installdir/bin
$ cache_darshanlogs.py --perf /path/to/a/darshanlog.darshan
{
  "counters": {
    "mpio": {
      ...
    }
  }
}
```

Because pytokio is a *framework* for tying together different data sources, exactly which CLI tools will work on your system is dependent on what data sources are available to you. Darshan is perhaps the most widely deployed source of data. If you have Darshan logs collected in a central location on your system, you can try using pytokio's `summarize_darshanlogs.py` tool to create an index of all logs generated on a single day:

```
$ summarize_darshanlogs.py /global/darshanlogs/2018/10/8/fbench_*.darshan
{
  "/global/darshanlogs/2018/10/8/fbench_IOR_CORI2_id15540806_10-8-6559-
  ↪7673881787757600104_1.darshan": {
    "/global/project": {
      "read_bytes": 0,
      "write_bytes": 206144000000
    }
  },
  ...
}
```

All pytokio CLI tools' options can be displayed by running them with the `-h` option.

Finally, if you have downloaded the entire pytokio repository, there are some sample Darshan logs (and other files) in the `tests/inputs` directory which you can also use to verify basic functionality.

1.1 Installation

1.1.1 Downloading pytokio

There are two ways to get pytokio:

1. The source distribution, which contains everything needed to install pytokio, use its bundled CLI tools, and begin developing new applications with it. This tarball is available on the [pytokio release page](#).
2. The full repository, which includes tests, example notebooks, and this documentation. This is most easily obtained via git (`git clone https://github.com/nersc/pytokio`).

If you are just kicking the tires on pytokio, download #1. If you want to create your own connectors or tools, contribute to development, or run into any issues that you would like to debug, install #2.

1.1.2 Editing the Site Configuration

The `site.json` file, located in the `tokio/` directory, contains optional parameters that allow various pytokio tools to automatically discover the location of specific monitoring data and expose a more fully integrated feel through its APIs.

The file is set up as JSON containing key-value pairs. No one key has to be specified (or set to a valid value), as each key is only consulted when a specific tool requests it. If you simply never use a tool, its configuration keys will never be examined.

Configuration Options

As of pytokio 0.10, the following keys can be defined:

- **lmt_timestep** Number of seconds between successive measurements contained in the LMT database. Only used by `summarize_job` tool to establish padding to account for cache flushes.
- **mount_to_fsname** Dictionary that maps mount points (expressed as regular expressions) to logical file system names. Used by several CLI tools to made output more digestible for humans.
- **fsname_to_backend_name** Dictionary that maps logical file system names to backend file system names. Needed for cases where the name of a file system as described to users (e.g., “the scratch file system”) has a different backend name (“snx11168”) that monitoring tools may use. Allows users to access data from file systems without knowing names used only by system admins.
- **hdf5_files** *Time-indexed file path template* describing where TOKIO Time Series HDF5 files are stored, and where in the file path their timestamp is encoded.
- **isdct_files** *Time-indexed file path template* describing where NERSC-style ISDCT tar files files are stored, and where in the file path their timestamp is encoded.
- **lfsstatus_fullness_files** *Time-indexed file path template* describing where NERSC-style Lustre file system fullness logs are stored, and where in the file path their timestamp is encoded.
- **lfsstatus_map_files** *Time-indexed file path template* describing where NERSC-style Lustre file system OSS-OST mapping logs are stored, and where in the file path their timestamp is encoded.
- **hpss_report_files** *Time-indexed file path template* describing where HPSS daily report logs are stored, and where in the file path their timestamp is encoded.
- **jobinfo_jobid_providers** *Provider list* to inform which TOKIO connectors should be used to find job info through the `tokio.tools.jobinfo` API
- **lfsstatus_fullness_providers** *Provider list* to inform which TOKIO connectors should be used to find file system fullness data through the `tokio.tools.lfsstatus` API

Special Configuration Values

There are two special types of value described above:

Time-indexed file path templates are strings that describe a file path that is passed through `strftime` with a user-specified time to resolve where pytokio can find a specific file containing data relevant to that time. Consider the following example:

```
"isdct_files": "/global/project/projectdirs/pma/www/daily/%Y-%m-%d/Intel_DCT_%Y%m%d.
↳tgz",
```

If pytokio is asked to find the ISDCT log file generated for January 14, 2017, it will use this template string and try to extract the requested data from the following file:

```
/global/project/projectdirs/pma/www/daily/2017-01-14/Intel_DCT_20170114.tgz
```

Time-indexed file path templates need not only be strings; they can be lists or dicts as well with the following behavior:

- str: search for files matching this template
- list of str: search for files matching each template
- dict: use the key to determine the element in the dictionary to use as the template. That value is treated as a new template and is processed recursively.

This is documented in more detail in `tokio.tools.common.enumerate_dated_files()`.

Provider lists are used by tools that can extract the same piece of information from multiple data sources. For example, `tokio.tools.jobinfo` provides an API to convert a job id into a start and end time, and it can do this by either consulting Slurm's `sacct` command or a site-specific jobs database. The provider list for this tool would look like

```
"jobinfo_jobid_providers": [
    "slurm",
    "nersc_jobsdb"
],
```

where `slurm` and `nersc_jobsdb` are magic strings recognized by the `tokio.tools.jobinfo.get_job_startend()` function.

1.1.3 Installing pytokio

pytokio can be used either as an installed Python package or as just an unraveled tarball. It has no components that require compilation and its only path-dependent component is `site.json` which can be overridden using the `PYTOKIO_CONFIG` environment variable.

As described above, installing the Python package is accomplished by any one of the following:

```
$ pip install /path/to/pytokio-0.10.1/
$ pip install --user /path/to/pytokio-0.10.1/
$ cd /path/to/pytokio-0.10.1/ && python setup.py install --prefix=/path/to/installdir
```

You may also wish to install a single packaged blob. In these cases though, you will not be able to edit the default `site.json` and will have to create an external `site.json` and define its path in the `PYTOKIO_CONFIG` environment variable:

```
$ pip install pytokio
$ pip install /path/to/pytokio-0.10.1.tar.gz
$ vi ~/pytokio-config.json
...
$ export PYTOKIO_CONFIG=$HOME/pytokio-config.json
```

For this reason, pytokio is not distributed as wheels or eggs. While they should work without problems when `PYTOKIO_CONFIG` is defined (or you never use any features that require looking up configuration values), installing such bdist is not officially supported.

1.1.4 Testing the Installation

The [pytokio git repository](#) contains a comprehensive, self-contained test suite in its `tests/` subdirectory that can be run after installation if `nose` is installed:

```
$ pip install /path/to/pytokio-0.10.1
...
$ git clone https://github.com/nersc/pytokio
$ cd pytokio/tests
$ ./run_tests.sh
.....
```

This test suite also contains a number of small sample inputs in the `tests/inputs/` subdirectory that may be helpful for basic testing.

1.2 Architecture

Note: This documentation is drawn from the [pytokio architecture paper](#) presented at the 2018 Cray User Group. For a more detailed description, please consult that paper.

The Total Knowledge of I/O (TOKIO) framework connects data from component-level monitoring tools across the I/O subsystems of HPC systems. Rather than build a universal monitoring solution and deploy a scalable data store to retain all monitoring data, TOKIO connects to *existing* monitoring tools and databases, indexes these tools' data, and presents the data from multiple connectors in a single, coherent view to downstream analysis tools and user interfaces.

To do this, `pytokio` is built upon the following design criteria:

1. Use existing tools already in production.
2. Leave data where it is.
3. Make data as accessible as possible.

`pytokio` is comprised of four layers:

Each layer is then composed of modules which are largely independent of each other to allow TOKIO to integrate with whatever selection of tools your HPC center has running in production.

1.2.1 Connectors

Connectors are independent, modular components that provide an interface between individual component-level tools you have installed in your HPC environment and the higher-level TOKIO layers. Each connector interacts with the native interface of a component-level tool and provides data from that tool in the form of a tool-independent interface.

Note: A complete list of implemented connectors can be found in the [tokio.connectors](#) documentation.

As a concrete example, consider the [LMT component-level tool](#) which exposes Lustre file system workload data through a MySQL database. The LMT database connector is responsible for establishing and destroying connections to the MySQL database as well as tracking stateful entities such as database cursors. It also encodes the schema of the LMT database tables, effectively abstracting the specific workings of the LMT database from the information that the LMT tool provides. In this sense, a user of the LMT database connector can use a more semantically meaningful interface (e.g., `tokio.connectors.lmtdb.LmtDb.get_mds_data()` to retrieve metadata server loads) without having to craft SQL queries or write any boilerplate MySQL code.

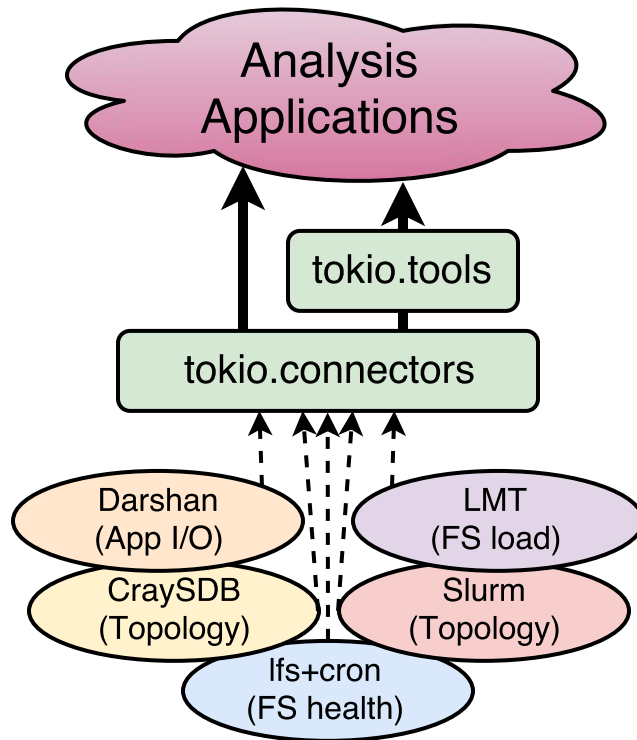


Fig. 1: Overview of pytokio's four layers.

At the same time, the LMT database connector does *not* modify the data retrieved from the LMT MySQL database before returning it. As such, using the LMT database connector still requires an understanding of the underlying LMT tool and the significance of the data it returns. This design decision restricts the role of connectors to being convenient interfaces into existing tools that eliminate the need to write glue code between component-level tools and higher-level analysis functions.

All connectors also provide serialization and deserialization methods for the tools to which they connect. This allows the data from a component-level tool to be stored for offline analysis, shared among collaborators, or cached for rapid subsequent accesses. Continuing with the LMT connector example, the data retrieved from the LMT MySQL database may be serialized to formats such as SQLite. Conversely, the LMT connector is also able to load LMT data from these alternative formats for use via the same downstream connector interface (e.g., `tokio.connectors.lmtdb.LmtDb.get_mds_data()`). This dramatically simplifies some tasks such as publishing analysis data that originated from a restricted-access data source or testing new analysis code.

pytokio implements each connector as a Python class. Connectors which rely on stateful connections, such as those which load data from databases, generally wrap a variety of database interfaces and may or may not have caching interfaces. Connectors which operate statelessly, such as those that load and parse discrete log files, are generally derived from Python dictionaries or lists and self-populate when initialized. Where appropriate, these connectors also have methods to return different representations of themselves; for example, some connectors provide a `to_dataframe()` method (such as `tokio.connectors.slurm.to_dataframe()`) which returns the requested connector data as a pandas DataFrame.

1.2.2 Tools

TOKIO tools are implemented on top of connectors as a set of interfaces that are semantically closer to how analysis applications may wish to access component-level data. They typically serve two purposes:

1. encapsulating site-specific information on how certain data sources are indexed or where they may be found
2. providing higher-level abstractions atop one or more connectors to mask the complexities or nuances of the underlying data sources

pytokio factors out all of its site-specific knowledge of connectors into a single site-specific configuration file, *site.json*, as described in the *Install Guide*. This configuration file is composed of arbitrary JSON-encoded key-value pairs which are loaded whenever pytokio is imported, and the specific meaning of any given key is defined by whichever tool accesses it. Thus, this site-specific configuration data does not prescribe any specific schema or semantic on site-specific information, and it does not contain any implicit assumptions about which connectors or tools are available on a given system.

The other role of TOKIO tools are to combine site-specific knowledge and multiple connectors to provide a simpler set of interfaces that are semantically closer to a question that an I/O user or administrator may actually ask. Continuing with the Darshan tool example from the previous section, such a question may be, “How many GB/sec did job 2468187 achieve?” Answering this question involves several steps:

1. Retrieve the start date for job id 2468187 from the system workload manager or a job accounting database
2. Look in the Darshan repository for logs that match `jobid=2468187` on that date
3. Run the `darshan-parser --perf` tool on the matching Darshan log and retrieve the estimated maximum I/O performance

pytokio provides connectors and tools to accomplish each one of these tasks:

1. The **Slurm connector** provides `tokio.connectors.slurm.Slurm.get_job_startend()` which retrieves a job’s start and end times when given a Slurm job id
2. The **Darshan tools** provides `tokio.tools.darshan.find_darshanlogs()` which returns a list of matching Darshan logs when given a job id and the date on which that job ran

3. The **Darshan connector** provides `tokio.connectors.darshan.Darshan.darshan_parser_perf()` which retrieves I/O performance data from a single Darshan log

Because this is such a routine process when analyzing application I/O performance, the Darshan tools interface implements this entire sequence in a single, higher-level function called `tokio.tools.darshan.load_darshanlogs()`. This function, depicted below, effectively links two connectors (Slurm and Darshan) and provides a single function to answer the question of “how well did job #2468187 perform?”

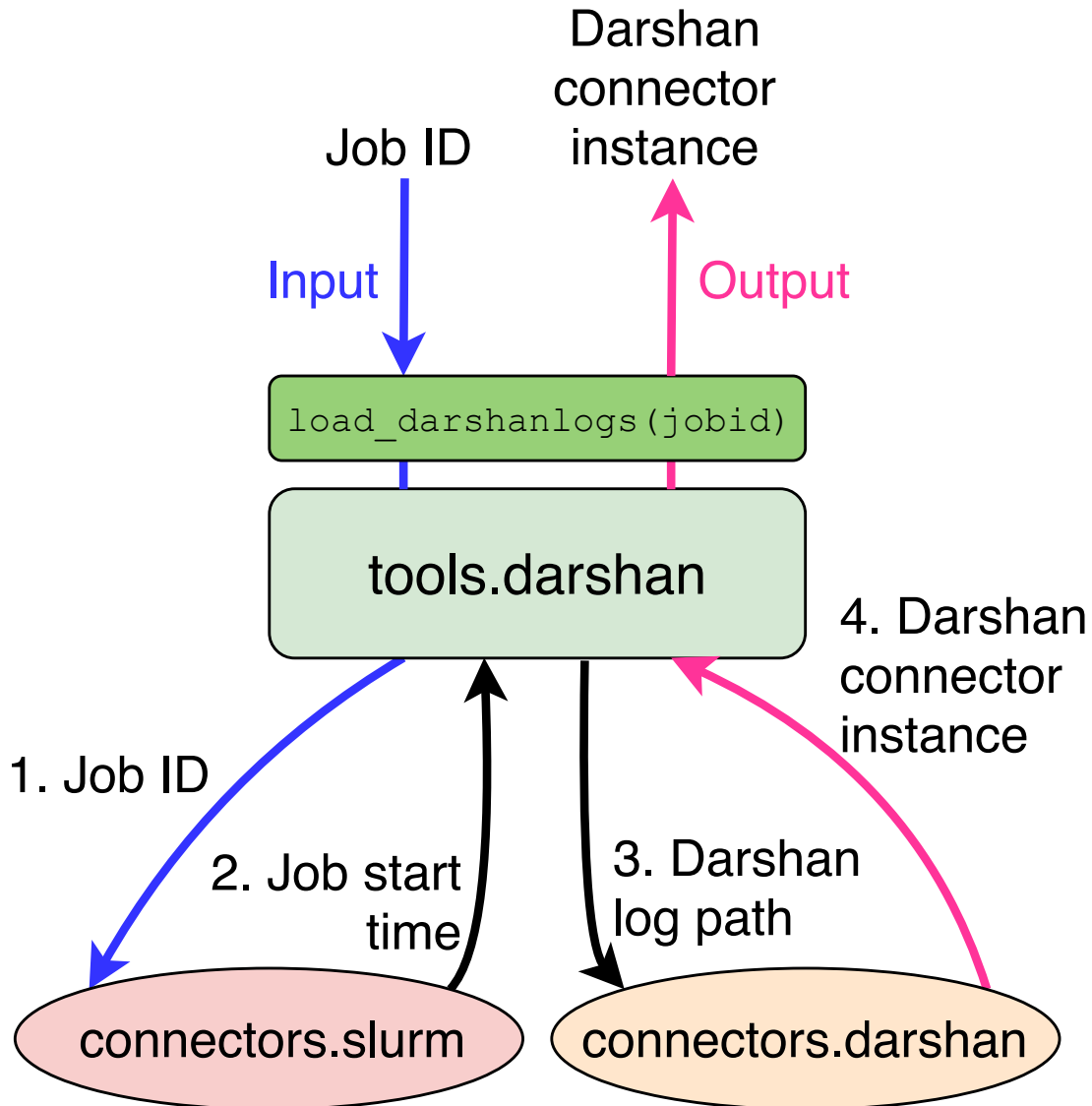


Fig. 2: Darshan tools interface for converting a Slurm Job ID into `tokio.connectors.darshan.Darshan` objects.

This simplifies the process of developing user-facing tools to analyze Darshan logs. Any analysis tool which uses application I/O performance and operates from job ids can replace hundreds of lines of boilerplate code with a single function call into the Darshan tool, and it alleviates users from having to understand the Darshan log repository directory structure to quickly find profiling data for their jobs.

TOKIO tools interfaces are also what facilitate portable, highly integrated analyses and services for I/O performance analysis. In the aforementioned examples, the Darshan tools interface assumes that Slurm is the system workload

manager and the preferred way to get start and end times for a job id. However, there is also a more generic `tokio.tools.jobinfo` tool interface which serves as a connector-agnostic interface that retrieves basic job metrics (start and end times, node lists, etc) using a site-configurable, prioritized list of connectors.

Consider the end-to-end example:

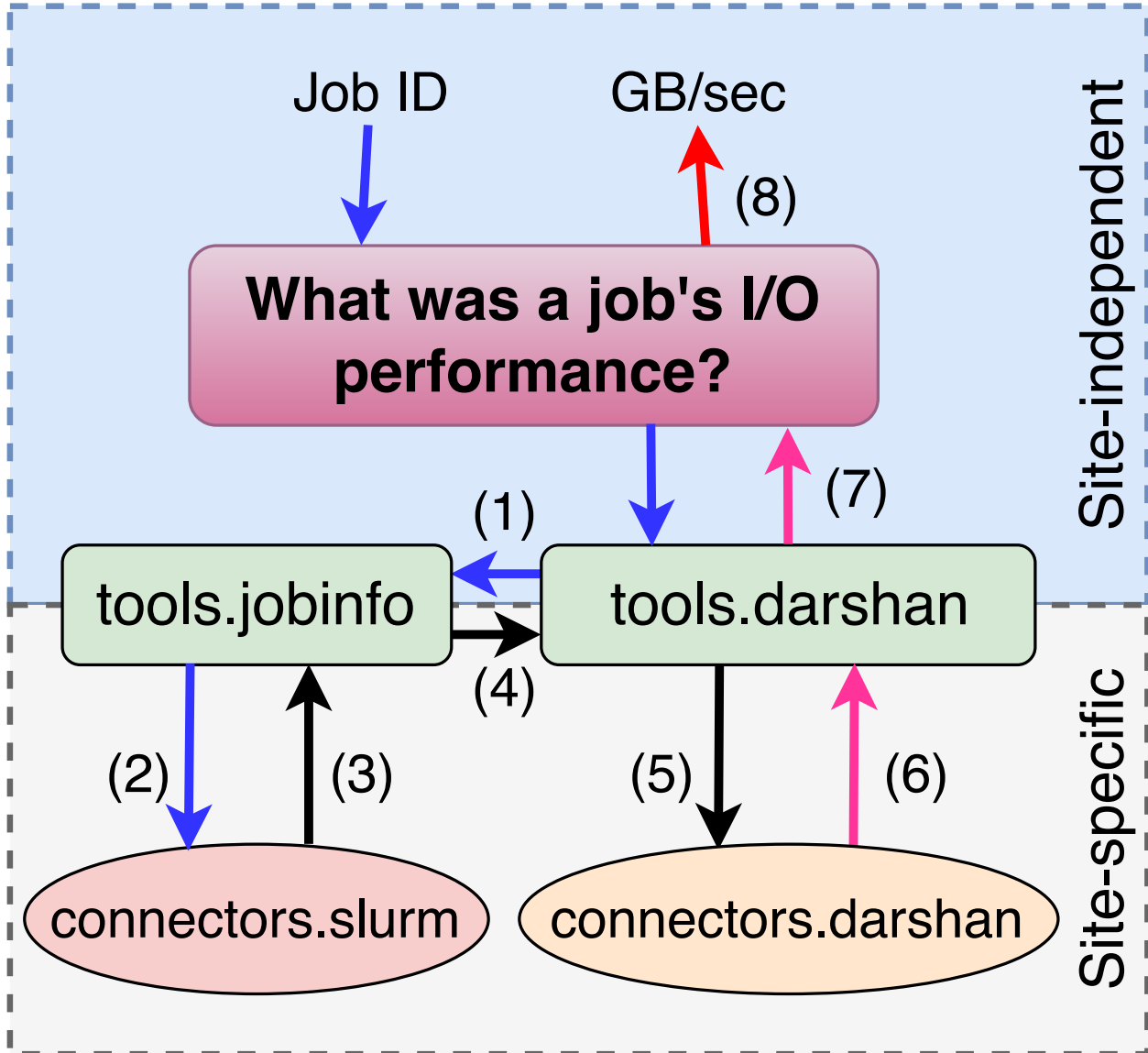


Fig. 3: Example of how the `tokio.tools.jobinfo` tools interface enables portability across different HPC sites.

In this case, an analysis application’s purpose is to answer the question, “What was a job’s I/O performance?” To accomplish this, the analysis takes a job id as its sole input and makes a single call into the pytokio Darshan tool’s `tokio.tools.darshan.load_darshanlogs()` function. Then

1. The Darshan tool first uses the jobinfo tool to convert the job id into a start/end time in a site-independent way.
2. The jobinfo tool uses the site configuration to use the Slurm connector to convert the job id...
3. ... into a start/end time,
4. which is passed back to the Darshan tool.

5. The Darshan tool then uses the job start time to determine where the job's Darshan log is located in the site-specific repository, and uses this log path. . .
6. . . . to retrieve a connector interface into the log.
7. The Darshan tool returns this connector interface to the analysis application,
8. which extracts the relevant performance metric and returns it to the end user

Through this entire process, the analysis application's only interface into pytokio was a single call into the Darshan tools interface. Beyond this, pytokio was responsible for determining both the proper mechanism to convert a job id into a job start time and the location of Darshan logs on the system. Thus, this analysis application is entirely free of site-specific knowledge and can be run at any HPC center to obtain I/O performance telemetry when given a job id. The only requirement is that pytokio is installed at the HPC center, and it is correctly configured to reflect that center's site-specific configurations.

1.2.3 Analyses

TOKIO connectors and tools interfaces are simply mechanisms to access I/O telemetry from throughout an HPC center. Higher-level analysis applications are required to actually pytokio's interfaces and deliver to meaningful insight to an end-user. That said, pytokio includes a number of example analysis applications and services that broadly fall into three categories.

1. Command-line interfaces
2. Statistical analysis tools
3. Data and analysis services

Many of these tools are packaged separately from pytokio and simply call on pytokio as a dependency.

1.3 Command Line Tools

pytokio implements its bundled CLI tools as thin wrappers around the `tokio.cli` package. These CLI tools are documented within that module's API documentation.

1.4 TOKIO Time Series Format

pytokio uses the TOKIO Time Series (TTS) format to serialize time series data generated by various storage and data systems in a standardized format. TTS is based on the HDF5 data format and is implemented within the `tokio.connectors.hdf5` connector.

The datasets supported by the TTS format and HDF5 connector are:

- `dataservers/cpuidle`
- `dataservers/cpload`
- `dataservers/cpusys`
- `dataservers/cpuuser`
- `dataservers/membuffered`
- `dataservers/memcached`
- `dataservers/memfree`

- dataservers/memslab
- dataservers/memslab_unrecl
- dataservers/memtotal
- dataservers/memused
- dataservers/netinbytes
- dataservers/netoutbytes
- datatargets/readbytes
- datatargets/readoprates
- datatargets/readops
- datatargets/readrates
- datatargets/writebytes
- datatargets/writeoprates
- datatargets/writeops
- datatargets/writerates
- failover/datatargets
- failover/mdtargets
- fullness/bytes
- fullness/bytestotal
- fullness/inodes
- fullness/inodestotal
- mdservers/cpuidle
- mdservers/cpload
- mdservers/cpusys
- mdservers/cpuuser
- mdservers/membuffered
- mdservers/memcached
- mdservers/memfree
- mdservers/memslab
- mdservers/memslab_unrecl
- mdservers/memtotal
- mdservers/memused
- mdservers/netinbytes
- mdservers/netoutbytes
- mdtargets/closerates
- mdtargets/closes
- mdtargets/getattrrates

- mdtargets/getattrs
- mdtargets/getxattrrates
- mdtargets/getxattrs
- mdtargets/linkrates
- mdtargets/links
- mdtargets/mkdirrates
- mdtargets/mkdirs
- mdtargets/mknodrates
- mdtargets/mknods
- mdtargets/openrates
- mdtargets/opens
- mdtargets/readbytes
- mdtargets/readoprates
- mdtargets/readops
- mdtargets/readrates
- mdtargets/renamerates
- mdtargets/renames
- mdtargets/rmdirrates
- mdtargets/rmdirs
- mdtargets/setattrrates
- mdtargets/setattrs
- mdtargets/statfsrates
- mdtargets/statfss
- mdtargets/unlinkrates
- mdtargets/unlinks
- mdtargets/writebytes
- mdtargets/writeoprates
- mdtargets/writeops
- mdtargets/writerates

The TTS format strives to achieve semantic consistency in that a row that is labeled as 2019-07-11 03:45:05 in a table such as:

Timestamp	OST0000	OST0001
2019-07-11 03:45:00	52382342	98239803
2019-07-11 03:45:05	23498237	92374926
2019-07-11 03:45:10	90384233	19375629

will contain data corresponding to the time from 3:45:05 (inclusive) to 3:45:10 (exclusive).

1.5 tokio package

The Total Knowledge of I/O (TOKIO) reference implementation, pytokio.

1.5.1 Subpackages

tokio.analysis package

Various functions that may be of use in analyzing TOKIO data. These are provided as a convenience rather than a set of core functionality.

Submodules

tokio.analysis.umami module

Class and tools to generate TOKIO UMAMI plots

class tokio.analysis.umami.**Umami** (***kws*)

Bases: `collections.OrderedDict`

Subclass of dictionary that stores all of the data needed to generate an UMAMI diagram. It is keyed by a metric name, and values are UmamiMetric objects which contain timestamps (x values) and measurements (y values)

`_to_dict_for_pandas` (*stringify_key=False*)

Convert this object into a DataFrame, indexed by timestamp, with each column as a metric. The Umami attributes (labels, etc) are not expressed.

plot (*output_file=None, highlight_index=-1, linewidth=1, linecolor='#853692', colorscale=['#DA0017', '#FD6A07', '#40A43A', '#2C69A9'], fontsize=12, figsize=(6.0, 1.3333333333333333)*)

Create a graphical representation of the UMAMI object

Parameters

- **output_file** (*str or None*) – save umami diagram to file of given name
- **highlight_index** (*int*) – index of measurement to highlight
- **linewidth** (*int*) – linewidth for both timeseries and boxplot lines
- **linecolor** (*str*) – color of line in timeseries panels
- **colorscale** (*list of str*) – colors to use for data below the 25th, 50th, 75th, and 100th percentiles
- **fontsize** (*int*) – font size for UMAMI labels
- **figsize** (*tuple of float*) – x, y dimensions of a single UMAMI row; multiplied by `len(self.keys())` to determine full diagram height

Returns List of `matplotlib.axis.Axis` objects corresponding to each panel in the UMAMI diagram

Return type `list`

to_dataframe ()

Return a representation of self as `pandas.DataFrame`

Returns numerical representation of the values being plotted

Return type `pandas.DataFrame`

to_dict()

Convert this object (and all of its constituent `UmamiMetric` objects) into a dictionary

to_json()

Serialize self into a JSON string

Returns JSON representation of numerical data being plotted

Return type `str`

class `tokio.analysis.umami.UmamiMetric` (*timestamps, values, label, big_is_good=True*)

Bases: `object`

A single row of an UMAMI diagram.

Logically contains timeseries data from a single connector, where the *timestamps* attribute is a list of timestamps (seconds since epoch), and the 'values' attribute is a list of values corresponding to each timestamp. The number of timestamps and attributes must always be the same.

append (*timestamp, value*)

Can only add values along with a timestamp.

pop()

Analogous to the list `.pop()` method.

to_json()

Create JSON-encoded string representation of self

Returns JSON-encoded representation of values stored in `UmamiMetric`

Return type `str`

`tokio.analysis.umami._serialize_datetime` (*obj*)

Special serializer function that converts datetime into something that can be encoded in json

tokio.cli package

pytokio implements its command-line tools within this package. Each such CLI tool either implements some useful analysis on top of pytokio connectors, tools, or analysis or maps some of the internal python APIs to command-line arguments.

Most of these tools implement a `--help` option to explain the command-line options.

Submodules

tokio.cli.archive_collectdes module

Dumps a lot of data out of ElasticSearch using the Python API and native scrolling support. Output either as native json from ElasticSearch or as serialized TOKIO TimeSeries (TTS) HDF5 files.

Can use `PYTOKIO_ES_USER` and `PYTOKIO_ES_PASSWORD` environment variables to pass on to the Elasticsearch connector for http authentication.

`tokio.cli.archive_collectdes.dataset2metadataset_key` (*dataset_key*)

Return the metadataset name corresponding to a dataset name

Parameters `dataset_name` (*str*) – Name of a dataset

Returns Name of corresponding metadataset name

Return type `str`

`tokio.cli.archive_collectdes.main` (*argv=None*)

Entry point for the CLI interface

`tokio.cli.archive_collectdes.metadataset2dataset_key` (*metadataset_name*)

Return the dataset name corresponding to a metadataset name

Metadatasets are not ever stored in the HDF5 and instead are only used to store data needed to correctly calculate dataset values. This function maps a metadataset name to its corresponding dataset name.

Parameters `metadataset_name` (*str*) – Name of a metadataset

Returns Name of corresponding dataset name, or None if *metadataset_name* does not appear to be a metadataset name.

Return type `str`

`tokio.cli.archive_collectdes.normalize_cpu_datasets` (*inserts, datasets*)

Normalize CPU load datasets

Divide each element of CPU datasets by the number of CPUs counted at each point in time. Necessary because these measurements are reported on a per-core basis, but not all cores may be reported for each timestamp.

Parameters

- **inserts** (*list of tuples*) – list of inserts that were used to populate datasets
- **datasets** (*dict of TimeSeries*) – all of the datasets being populated

Returns Nothing

`tokio.cli.archive_collectdes.pages_to_hdf5` (*pages, output_file, init_start, init_end, query_start, query_end, timestep, num_servers, devices_per_server, threads=1*)

Stores a page from Elasticsearch query in an HDF5 file Take pages from Elasticsearch query and store them in `output_file`

Parameters

- **pages** (*list*) – A list of page objects (dictionaries)
- **output_file** (*str*) – Path to an HDF5 file in which page data should be stored
- **init_start** (*datetime.datetime*) – Lower bound of time (inclusive) to be stored in the `output_file`. Used when creating a non-existent HDF5 file.
- **init_end** (*datetime.datetime*) – Upper bound of time (inclusive) to be stored in the `output_file`. Used when creating a non-existent HDF5 file.
- **query_start** (*datetime.datetime*) – Retrieve data greater than or equal to this time from Elasticsearch
- **query_end** (*datetime.datetime*) – Elasticsearch
- **timestep** (*int*) – Time, in seconds, between successive sample intervals to be used when initializing `output_file`
- **num_servers** (*int*) – Number of discrete servers in the cluster. Used when initializing `output_file`.
- **devices_per_server** (*int*) – Number of SSDs per server. Used when initializing `output_file`.
- **threads** (*int*) – Number of parallel threads to utilize when parsing the Elasticsearch output

`tokio.cli.archive_collectdes.process_page` (*page*)

Go through a list of docs and insert their data into a numpy matrix. In the future this should be a flush function attached to the CollectdEs connector class.

Parameters `page` (*dict*) – A single page of output from an Elasticsearch scroll query. Should contain a `hits` key.

`tokio.cli.archive_collectdes.reset_timeseries` (*timeseries, start, end, value=-0.0*)

Zero out a region of a `tokio.timeseries.TimeSeries` dataset

Parameters

- **timeseries** (`tokio.timeseries.TimeSeries`) – data from a subset should be zeroed
- **start** (`datetime.datetime`) – Time at which zeroing of all columns in *timeseries* should begin
- **end** (`datetime.datetime`) – Time at which zeroing all columns in *timeseries* should end (exclusive)
- **value** – value which should be set in every element being reset

Returns Nothing

`tokio.cli.archive_collectdes.update_datasets` (*inserts, datasets*)

Insert list of tuples into a dataset

Insert a list of tuples into a `tokio.timeseries.TimeSeries` object serially

Parameters

- **inserts** (*list of tuples*) – List of tuples which should be serially inserted into a dataset. The tuples can be of the form
 - dataset name (str)
 - timestamp (`datetime.datetime`)
 - column name (str)
 - valueor
 - dataset name (str)
 - timestamp (`datetime.datetime`)
 - column name (str)
 - value
 - reducer name (str)where
 - *dataset name* is the key used to retrieve a target `tokio.timeseries.TimeSeries` object from the *datasets* argument
 - *timestamp* and *column name* reference the element to be updated
 - *value* is the new value to insert into the given (*timestamp, column name*) location within *dataset*.
 - *reducer name* is `None` (to just replace whatever value currently exists in the (*timestamp, column name*) location, or ‘sum’ to add *value* to the existing value.

- **datasets** (*dict*) – Dictionary mapping dataset names (*str*) to *tokio.timeseries.TimeSeries* objects

Returns number of elements in *inserts* which were not inserted because their timestamp value was out of the range of the dataset to be updated.

Return type *int*

tokio.cli.archive_esnet_snmp module

Retrieves ESnet SNMP counters and store them in TOKIO Timeseries format

class `tokio.cli.archive_esnet_snmp.Archiver` (*query_start, query_end, interfaces, timestep, timeout=30.0, *args, **kwargs*)

Bases: *dict*

A dictionary containing TimeSeries objects

Contains the TimeSeries objects being populated from a remote data source. Implemented as a class so that a single object can store all of the TimeSeries objects that are generated by multiple method calls.

__init__ (*query_start, query_end, interfaces, timestep, timeout=30.0, *args, **kwargs*)

Initializes the archiver and stores its settings

Parameters

- **query_start** (*datetime.datetime*) – Lower bound of time to be archived, inclusive
- **query_end** (*datetime.datetime*) – Upper bound of time to be archived, inclusive
- **interfaces** (*list of tuples*) – List of endpoints and interfaces to archive. Each tuple is of the form (endpoint, interface).
- **timestep** (*int*) – Number of seconds between successive data points. The ESnet service may not honor this request.
- **timeout** (*float*) – Seconds before HTTP connection times out

archive (*input_file=None*)

Extract and encode data from ESnet's SNMP service

Queries the ESnet SNMP REST service, interprets resulting data, and populates a dictionary of TimeSeries objects with those values.

Parameters **esnetsnmp** (*tokio.connectors.esnet_snmp.EsnetSnmp*) – Connector instance

finalize ()

Convert datasets to deltas where necessary and tack on metadata

Perform a few finishing actions to all datasets contained in self after they have been populated. Such actions are configured entirely in self.config and require no external input.

init_datasets (*dataset_names*)

Populate empty datasets within self

Creates and attaches TimeSeries objects to self based on a given column list

Parameters **dataset_names** (*list of str*) – keys corresponding to self.config defining which datasets are being initialized

set_timeseries_metadata (*dataset_names*)

Set metadata constants (version, units, etc) on datasets and groups

Parameters `dataset_names` (*list of str*) – keys corresponding to self.config for the datasets whose metadata should be set

```
tokio.cli.archive_esnet_snmp.archive_esnet_snmp(init_start, init_end, interfaces,
                                                timestep, output_file, query_start,
                                                query_end, input_file=None,
                                                **kwargs)
```

Retrieves remote data and stores it in TOKIO time series format

Given a start and end time, retrieves all of the relevant contents of a remote data source and encodes them in the TOKIO time series HDF5 data format.

Parameters

- **init_start** (*datetime.datetime*) – The first timestamp to be included in the HDF5 file
- **init_end** (*datetime.datetime*) – The timestamp following the last timestamp to be included in the HDF5 file.
- **interfaces** (*list of tuples*) – List of (endpoint, interface) elements to query.
- **timestep** (*int*) – Number of seconds between successive entries in the HDF5 file to be created.
- **output_file** (*str*) – Path to the file to be created.
- **query_start** (*datetime.datetime*) – Time after which remote data should be retrieved, inclusive.
- **query_end** (*datetime.datetime*) – Time before which remote data should be retrieved, inclusive.
- **input_file** (*str or None*) – Path to a cached input. If specified, the remote REST API will not be contacted and the contents of this file will be instead loaded.
- **kwargs** (*dict*) – Extra arguments to be passed to Archiver.__init__()

```
tokio.cli.archive_esnet_snmp.endpoint_name(endpoint, interface)
Create a single key from an endpoint, interface pair
```

Parameters

- **endpoint** (*str*) – The name of an endpoint
- **interface** (*str*) – The interface on the given endpoint

Returns A single key combining endpoint and interface

Return type `str`

```
tokio.cli.archive_esnet_snmp.init_hdf5_file(datasets, init_start, init_end, hdf5_file)
Initialize the datasets at full dimensions in the HDF5 file if necessary
```

```
tokio.cli.archive_esnet_snmp.main(argv=None)
Entry point for the CLI interface
```

tokio.cli.archive_lmtadb module

Retrieve the contents of an LMT database and cache it locally.

```
class tokio.cli.archive_lmtadb.DatasetDict(query_start, query_end, timestep,
                                           sort_hex=True, *args, **kwargs)
Bases: dict
```


A dictionary containing TimeSeries objects

Contains the TimeSeries objects being populated from an LMT database. Implemented as a class so that a single object can store all of the TimeSeries objects that are generated by multiple method calls.

archive_mds_data (*lmtdb*)

Extract and encode data from LMT's MDS_DATA table

Queries the LMT database, interprets resulting rows, and populates a dictionary of TimeSeries objects with those values.

Parameters **lmtdb** (*LmtDb*) – database object

archive_mds_ops_data (*lmtdb*)

Extract and encode data from LMT's MDS_OPS_DATA table

Queries the LMT database, interprets resulting rows, and populates a dictionary of TimeSeries objects with those values. Avoids JOINing the MDS_VARIABLE_INFO table and instead uses an internal mapping of OPERATION_IDS to demultiplex the data in MDS_OPS_DATA into different HDF5 datasets.

Parameters **lmtdb** (*LmtDb*) – database object

archive_oss_data (*lmtdb*)

Extract and encode data from LMT's OSS_DATA table

Queries the LMT database, interprets resulting rows, and populates a dictionary of TimeSeries objects with those values.

Parameters **lmtdb** (*LmtDb*) – database object

archive_ost_data (*lmtdb*)

Extract and encode data from LMT's OST_DATA table

Queries the LMT database, interprets resulting rows, and populates a dictionary of TimeSeries objects with those values.

Parameters **lmtdb** (*LmtDb*) – database object

convert_deltas (*dataset_names*)

Convert datasets from absolute values to values per timestep

Given a list of dataset names, determine if they need to be converted from monotonically increasing counters to counts per timestep, and convert those that do. For those that don't, trim off the final row since it is not needed to calculate the difference between rows.

Parameters **dataset_names** (*list of str*) – keys corresponding to self.config for the datasets to be converted/corrected

finalize ()

Convert datasets to deltas where necessary and tack on metadata

Perform a few finishing actions to all datasets contained in self after they have been populated. Such actions are configured entirely in self.config and require no external input.

init_datasets (*dataset_names, columns*)

Populate empty datasets within self

Creates and attaches TimeSeries objects to self based on a given column list

Parameters

- **dataset_names** (*list of str*) – keys corresponding to self.config defining which datasets are being initialized

- **columns** (*list of str*) – column names to use in the TimeSeries datasets being created

set_timeseries_metadata (*dataset_names*)

Set metadata constants (version, units, etc) on datasets and groups

Parameters *dataset_names* (*list of str*) – keys corresponding to self.config for the datasets whose metadata should be set

`tokio.cli.archive_lmtdb.archive_lmtdb` (*lmtdb, init_start, init_end, timestep, output_file, query_start, query_end*)

Given a start and end time, retrieve all of the relevant contents of an LMT database.

`tokio.cli.archive_lmtdb.init_hdf5_file` (*datasets, init_start, init_end, hdf5_file*)

Initialize the datasets at full dimensions in the HDF5 file if necessary

`tokio.cli.archive_lmtdb.main` (*argv=None*)

Entry point for the CLI interface

tokio.cli.archive_mmperfmon module

Retrieves mmperfmon counters and store them in TOKIO Timeseries format

Command-line tool that loads a `tokio.connectors.mmperfmon.Mmperfmon` object and encodes it as a TOKIO TimeSeries object. Syntax to create a new HDF5 is:

```
$ archive_mmperfmon --timestep=60 --init-start 2019-05-15T00:00:00 \
  --init-end 2019-05-16T00:00:00 mmperfmon.2019-05-15.tgz
```

where `mmperfmon.2019-05-15.tgz` is one or more files that can be loaded by `tokio.connectors.mmperfmon.Mmperfmon.from_file()`.

When updating an existing HDF5 file, the minimum required syntax is:

```
$ archive_mmperfmon --timestep=60 mmperfmon.2019-05-15.tgz
```

The init start/end times are only required when creating an empty HDF5 file.

class `tokio.cli.archive_mmperfmon.Archiver` (*init_start, init_end, timestep, num_luns, num_servers, *args, **kwargs*)

Bases: `dict`

A dictionary containing TimeSeries objects

Contains the TimeSeries objects being populated from a remote data source. Implemented as a class so that a single object can store all of the TimeSeries objects that are generated by multiple method calls.

__init__ (*init_start, init_end, timestep, num_luns, num_servers, *args, **kwargs*)

Initializes the archiver and stores its settings

Parameters

- **init_start** (*datetime.datetime*) – Lower bound of time to be archived, inclusive
- **init_end** (*datetime.datetime*) – Upper bound of time to be archived, exclusive
- **timestep** (*int*) – Number of seconds between successive data points.
- **num_luns** (*int or None*) – Number of LUNs expected to appear in mmperfmon outputs. If None, autodetect.

- **num_servers** (*int or None*) – Number of NSD servers expected to appear in mmperfmon outputs. If None, autodetect.

archive (*mmpm*)

Extracts and encode data from an Mmperfmon object

Uses the mmperfmon connector to populate one or more TimeSeries objects.

Parameters **mmpm** (`tokio.connectors.mmperfmon.Mmperfmon`) – Instance of the mmperfmon connector class containing all of the data to be archived

finalize ()

Convert datasets to deltas where necessary and tack on metadata

Perform a few finishing actions to all datasets contained in self after they have been populated. Such actions are configured entirely in self.config and require no external input.

init_dataset (*dataset_name, columns*)

Initialize an empty dataset within self

Creates and attaches a TimeSeries object to self

Parameters

- **dataset_name** (*str*) – name of dataset to be initialized
- **columns** (*list of str*) – columns to initialize

init_datasets (*mmpm*)

Initialize all datasets that can be created from an Mmperfmon instance

This method examines an mmpm and identifies all TimeSeries datasets that can be derived from it, then calculates the dimensions of said datasets based on how many unique columns were found. This is required because the precise number of columns is difficult to generalize a priori on SAN file systems with arbitrarily connected LUNs and servers.

Also caches the mappings between LUN and NSD server names and their functions (data or metadata).

Parameters **mmpm** (`tokio.connectors.mmperfmon.Mmperfmon`) – Object from which possible datasets should be identified and sized.

lun_type (*lun_name*)

Infers the dataset name to which a LUN should belong

Returns the dataset name in which a given GPFS LUN name belongs. This is required for block-based file systems in which servers serve both data and metadata.

This function relies on `tokio.config.CONFIG['mmperfmon_lun_map']`.

Parameters **lun_name** (*str*) – The name of a LUN

Returns The name of a dataset in which *lun_name* should be filed.

Return type *str*

server_type (*server_name*)

Infers the type of server (data or metadata) from its name

Returns the type of server that *server_name* is. This relies on `tokio.config.CONFIG['mmperfmon_md_servers']` which encodes a regex that matches metadata server names.

This method only makes sense for GPFS clusters that have distinct metadata servers.

Parameters **server_name** (*str*) – Name of the server

Returns “mdserver” or “dataserver”

Return type `str`

set_timeseries_metadata (*dataset_names*)

Set metadata constants (version, units, etc) on datasets and groups

Parameters **dataset_names** (*list of str*) – datasets whose metadata should be set

`tokio.cli.archive_mmperfmon.archive_mmperfmon` (*init_start, init_end, timestep, num_luns, num_servers, output_file, input_files*)

Retrieves remote data and stores it in TOKIO time series format

Given a start and end time, retrieves all of the relevant contents of a remote data source and encodes them in the TOKIO time series HDF5 data format.

Parameters

- **init_start** (*datetime.datetime*) – The first timestamp to be included in the HDF5 file
- **init_end** (*datetime.datetime*) – The timestamp following the last timestamp to be included in the HDF5 file.
- **timestep** (*int or None*) – Number of seconds between successive entries in the HDF5 file to be created. If None, autodetect.
- **num_luns** (*int or None*) – Number of LUNs expected to appear in mmperfmon outputs. If None, autodetect.
- **num_servers** (*int or None*) – Number of NSD servers expected to appear in mmperfmon outputs. If None, autodetect.
- **output_file** (*str*) – Path to the file to be created.
- **input_files** (*list of str*) – List of paths to input files from which mmperfmon connectors should be instantiated.

`tokio.cli.archive_mmperfmon.init_hdf5_file` (*datasets, init_start, init_end, hdf5_file*)

Creates HDF5 datasets within a file based on TimeSeries objects

Idempotently ensures that *hdf5_file* contains a dataset corresponding to each `tokio.timeseries.TimeSeries` object contained in the *datasets* object.

Parameters

- **datasets** (*Archiver*) – Dictionary keyed by dataset name and whose values are `tokio.timeseries.TimeSeries` objects. One HDF5 dataset will be created for each TimeSeries object.
- **init_start** (*datetime.datetime*) – If a dataset does not already exist within the HDF5 file, create it using this as a lower bound for the timesteps, inclusive
- **init_end** (*datetime.datetime*) – If a dataset does not already exist within the HDF5 file, create one using this as the upper bound for the timesteps, exclusive
- **hdf5_file** (*str*) – Path to the HDF5 file in which datasets should be initialized

`tokio.cli.archive_mmperfmon.main` (*argv=None*)

Entry point for the CLI interface

tokio.cli.cache_collectdes module

Dump a lot of data out of Elasticsearch using the Python API and native scrolling support.

Instantiates a `tokio.connectors.collectd_es.CollectdEs` object and relies on the `tokio.connectors.collectd_es.CollectdEs.query_timeseries()` method to populate a data structure that is then serialized to JSON.

```
tokio.cli.cache_collectdes.main (argv=None)  
    Entry point for the CLI interface
```

tokio.cli.cache_darshan module

Expose several methods of `tokio.connectors.darshan` via a command-line interface.

```
tokio.cli.cache_darshan.main (argv=None)  
    Entry point for the CLI interface
```

tokio.cli.cache_esnet_snmp module

Provide a CLI interface for `tokio.connectors.esnet_snmp.EsnetSnmp.to_dataframe()` and `tokio.connectors.esnet_snmp.EsnetSnmp.save_cache()` methods.

```
tokio.cli.cache_esnet_snmp.main (argv=None)  
    Entry point for the CLI interface
```

tokio.cli.cache_isdct module

Provide a CLI interface for `tokio.connectors.nersc_isdct.NerscIsdct.to_dataframe()` and `tokio.connectors.nersc_isdct.NerscIsdct.save_cache()` methods.

```
tokio.cli.cache_isdct.main (argv=None)  
    Entry point for the CLI interface
```

tokio.cli.cache_lfsstatus module

Provides CLI interfaces into the `tokio.tools.lfsstatus` tool's `tokio.tools.lfsstatus.get_failures()` and `tokio.tools.lfsstatus.get_fullness()` methods.

```
tokio.cli.cache_lfsstatus.main (argv=None)  
    Entry point for the CLI interface
```

tokio.cli.cache_lmtdb module

Retrieve the contents of an LMT database and cache it locally.

```
tokio.cli.cache_lmtdb.main (argv=None)  
    Entry point for the CLI interface
```

```
tokio.cli.cache_lmtdb.retrieve_tables (lmtdb, datetime_start, datetime_end, limit=None)  
    Given a start and end time, retrieve and cache all of the relevant contents of an LMT database.
```

tokio.cli.cache_mmpferfmon module

Provide a CLI interface for `tokio.connectors.mmpferfmon.Mmpferfmon.to_dataframe()` and `tokio.connectors.mmpferfmon.Mmpferfmon.save_cache()` methods.

`tokio.cli.cache_mmpferfmon.main` (*argv=None*)
Entry point for the CLI interface

tokio.cli.cache_nersc_globuslogs module

Command-line interface into the nersc_globuslogs connector

`tokio.cli.cache_nersc_globuslogs.main` (*argv=None*)
Entry point for the CLI interface

tokio.cli.cache_nersc_jobsdb module

Provides CLI interfaces for `tokio.connectors.nersc_jobsdb.NerscJobsDb.get_concurrent_jobs()`

`tokio.cli.cache_nersc_jobsdb.main` (*argv=None*)
Entry point for the CLI interface

tokio.cli.cache_slurm module

Provides CLI interfaces for `tokio.connectors.slurm.Slurm.to_dataframe()` and `tokio.connectors.slurm.Slurm.to_json()`.

`tokio.cli.cache_slurm.main` (*argv=None*)
Entry point for the CLI interface

tokio.cli.cache_topology module

Provides CLI interface for `tokio.tools.topology.get_job_diameter()`.

`tokio.cli.cache_topology.main` (*argv=None*)
Entry point for the CLI interface

tokio.cli.compare_isdct module

Compare two NERSC ISDCT dumps and report

1. the devices that appeared or were removed
2. the numeric counters whose values changed
3. the string counters whose contents changed

`tokio.cli.compare_isdct._convert_counters` (*counters, conversion_factor, label*)
Convert a single flat dictionary of counters of bytes into another unit

`tokio.cli.compare_isdct.convert_byte_keys` (*input_dict*, *conversion_factor*=9.313225746154785e-10, *label*='gibs')

Convert all keys ending in `_bytes` to some other unit. Accepts either the raw diff dict or the reduced dict from `reduce_diff()`

`tokio.cli.compare_isdct.discover_errors` (*diff_dict*)

Look through all diffs and report serial numbers of devices that show changes in counters that may indicate a hardware issue.

`tokio.cli.compare_isdct.main` (*argv*=None)

Entry point for the CLI interface

`tokio.cli.compare_isdct.print_summary` (*old_isdctfile*, *new_isdctfile*, *diff_dict*)

Print a human-readable summary of `diff_dict`

`tokio.cli.compare_isdct.reduce_diff` (*diff_dict*)

Take the raw output of `.diff()` and aggregate the results of each device

`tokio.cli.compare_isdct.summarize_errors` (*diff_dict*, *isdct_data*)

Print a human-readable summary of any bad SSDs

`tokio.cli.compare_isdct.summarize_reduced_diffs` (*reduced_diffs*)

Print a human-readable summary of the relevant reduced diff data

tokio.cli.darshan_bad_ost module

Given one or more Darshan logs containing both POSIX and Lustre counters, attempt to determine the performance each file saw and try to correlate poorly performing files with specific Lustre OSTs.

This tool first estimates per-file I/O bandwidths by dividing the total bytes read/written to each file by the time the application spent performing I/O to that file. It then uses data from Darshan's Lustre module to map these performance estimates to the OSTs over which each file was striped. With the list of OSTs and performance measurements corresponding to each OST, the Pearson correlation coefficient is then calculated between performance and each individual OST.

Multiple Darshan logs can be passed to increase the number of observations used for correlation. This tool does not work unless the Darshan log(s) contain data from the Lustre module.

`tokio.cli.darshan_bad_ost.correlate_ost_performance` (*darshan_logs*)

Generate a DataFrame containing files, performance measurements, and OST mappings and attempt to correlate performance with individual OSTs.

`tokio.cli.darshan_bad_ost.darshanlogs_to_ost_dataframe` (*darshan_logs*)

Given a set of Darshan log file paths, create a dataframe containing each file, its observed performance, and a matrix of values corresponding to what fraction of that file's contents were probably striped on each OST.

`tokio.cli.darshan_bad_ost.estimate_darshan_perf` (*ranks_data*)

Calculate performance in a sideways fashion: find the longest I/O time across any rank for this file, then divide the sum of all bytes read/written by this longest io time. This function expects to receive a dict that is keyed by MPI ranks (or a single "-1" key) and whose values are dicts corresponding to Darshan POSIX counters.

`tokio.cli.darshan_bad_ost.main` (*argv*=None)

Entry point for the CLI interface

`tokio.cli.darshan_bad_ost.summarize_darshan_perf` (*darshan_logs*)

Given a list of Darshan log file paths, calculate the performance observed from each file and identify OSTs over which each file was striped. Return this summary of file performances and stripes.

tokio.cli.darshan_scoreboard module

Process the Darshan daily summary generated by either `summarize_darshanlogs` or `index_darshanlogs` tools and generate a scoreboard of top sources of I/O based on user, file system, and/or application.

`tokio.cli.darshan_scoreboard.main` (*argv=None*)

Entry point for the CLI interface

`tokio.cli.darshan_scoreboard.print_top` (*categorized_data, max_show=10*)

Print the biggest I/O {users, exes, file systems}

`tokio.cli.darshan_scoreboard.query_index_db` (*db_filenames, limit_fs=None, limit_user=None, limit_exe=None, exclude_fs=None, exclude_user=None, exclude_exe=None, max_results=None*)

Reduce Darshan log index by fs, user, and/or exe

`tokio.cli.darshan_scoreboard.vprint` (*string, level*)

Print a message if verbosity is enabled

Parameters

- **string** (*str*) – Message to print
- **level** (*int*) – Minimum verbosity level required to print

tokio.cli.find_darshanlogs module

Provides CLI interface for the `tokio.tools.darshan.load_darshanlogs()` tool which locates darshan logs in the system-wide repository.

`tokio.cli.find_darshanlogs.main` (*argv=None*)

Entry point for the CLI interface

tokio.cli.index_darshanlogs module

tokio.cli.summarize_h5lmt module

Generate summary metrics from an h5lmt file. Will be eventually replaced by the `summarize_tts` command-line tool.

`tokio.cli.summarize_h5lmt.bin_dataset` (*hdf5_file, dataset_name, num_bins*)

Group timeseries dataset into bins

Parameters `dataset` (*h5py.Dataset*) – dataset to be binned up

Returns list of dictionaries corresponding to bins. Each dictionary contains data summarized over that bin's time interval.

`tokio.cli.summarize_h5lmt.bin_datasets` (*hdf5_file, dataset_names, orient='columns', num_bins=24*)

Group many timeseries datasets into bins

Takes a TOKIO HDF file and converts it into bins of reduced data (e.g., bin by hourly totals)

Parameters

- **hdf5_file** (*connectors.Hdf5*) – HDF5 file from where data should be retrieved
- **dataset_names** (*list of str*) – dataset names to be aggregated

- **columns** (*str*) – either ‘columns’ or ‘index’; same semantic meaning as `pandas.DataFrame.from_dict`
- **num_binds** (*int*) – number of bins to generate per day

Returns

Dictionary of lists. Keys are metrics, and values (lists) are the aggregated value of that metric in a single timestep bin. For example:

```
{
  "sum_some_metric": [ 0, 2, 3, 1],
  "sum_someother_metric": [9.9, 2.3, 5.1, 0.2],
}
```

`tokio.cli.summarize_h5lmt.main` (*argv=None*)

Entry point for the CLI interface

`tokio.cli.summarize_h5lmt.print_data_summary` (*data*, *units='TiB'*)

Print the output of the `summarize_reduced_data` function in a human-readable format

`tokio.cli.summarize_h5lmt.print_datum` (*datum=None*, *units='GiB'*)

Print out the relevant fields from a bin containing aggregated values

Parameters

- **datum** (*dict*) – the results of `bin_datum`
- **units** (*str*) – units to print (KiB, MiB, GiB, TiB)

`tokio.cli.summarize_h5lmt.summarize_reduced_data` (*data*)

Take a list of LMT data sets and return summaries of each relevant key

tokio.cli.summarize_job module

Take a darshan log or job start/end time and pull scalar data from every available TOKIO connector/tool configured for the system to present a single system-wide view of performance for the time during which that job was running.

`tokio.cli.summarize_job._identify_fs_from_path` (*path*, *mounts*)

Scan a list of mount points and try to identify the one that matches the given path

`tokio.cli.summarize_job.get_biggest_api` (*darshan_data*)

Determine the most-used API and file system based on the Darshan log

`tokio.cli.summarize_job.get_biggest_fs` (*darshan_data*)

Determine the most-used file system based on the Darshan log

`tokio.cli.summarize_job.main` (*argv=None*)

Entry point for the CLI interface

`tokio.cli.summarize_job.merge_dicts` (*dict1*, *dict2*, *assertion=True*, *prefix=None*)

Take two dictionaries and merge their keys. Optionally raise an exception if a duplicate key is found, and optionally merge the new dict into the old after adding a prefix to every key.

`tokio.cli.summarize_job.retrieve_concurrent_job_data` (*results*, *jobhost*, *concurrent_jobs*)

Get information about all jobs that were running during a time period

`tokio.cli.summarize_job.retrieve_darshan_data` (*results*, *darshan_log_file*, *silent_errors=False*)

Extract the performance data from the Darshan log

`tokio.cli.summarize_job.retrieve_jobid(results, jobid, file_count)`

Get JobId from either Slurm or the CLI argument

`tokio.cli.summarize_job.retrieve_lmt_data(results, file_system)`

Figure out the H5LMT file corresponding to this run

`tokio.cli.summarize_job.retrieve_ost_data(results, ost, ost_fullness=None, ost_map=None)`

Get Lustre server status via lfsstatus tool

`tokio.cli.summarize_job.retrieve_topology_data(results, jobinfo_cache_file, nodemap_cache_file)`

Get the diameter of the job (Cray XC)

`tokio.cli.summarize_job.serialize_datetime(obj)`

Special serializer function that converts datetime into something that can be encoded in json

`tokio.cli.summarize_job.summarize_byterate_df(dataframe, readwrite, timestep=None)`

Calculate some interesting statistics from a dataframe containing byte rate data.

`tokio.cli.summarize_job.summarize_cpu_df(dataframe, servertime)`

Calculate some interesting statistics from a dataframe containing CPU load data.

`tokio.cli.summarize_job.summarize_darshan(darshan_data)`

Synthesize new Darshan summary metrics based on the contents of a `connectors.darshan.Darshan` object that is partially or fully populated

`tokio.cli.summarize_job.summarize_darshan_posix(darshan_data)`

Extract key metrics from the POSIX module in a Darshan log

`tokio.cli.summarize_job.summarize_mds_ops_df(dataframe, opname, timestep=None)`

Summarize various metadata op counts over a time range

`tokio.cli.summarize_job.summarize_missing_df(dataframe)`

Populate the fraction missing counter from a given DataFrame

tokio.cli.summarize_tts module

Summarize the contents of a TOKIO TimeSeries (TTS) HDF5 file generated by `tokio.timeseries.TimeSeries.commit_dataset()`. This will eventually be merged with the functionality provided by the `summarize_h5lmt` command-line tool.

`tokio.cli.summarize_tts.humanize_units(byte_count, divisor=1024.0)`

Convert a raw byte count into human-readable base2 units

`tokio.cli.summarize_tts.main(argv=None)`

Entry point for the CLI interface

`tokio.cli.summarize_tts.print_column_summary(results)`

Format and print the summary data calculated by `summarize_columns()`

`tokio.cli.summarize_tts.print_timestep_summary(summary)`

Format and print the summary data calculated by `summarize_timesteps()`

`tokio.cli.summarize_tts.print_tts_hdf5_summary(results)`

Format and print the summary data calculated by `summarize_tts_hdf5()`

`tokio.cli.summarize_tts.summarize_columns(hdf5_file)`

Summarize read/write bytes for each column

`tokio.cli.summarize_tts.summarize_timesteps` (*hdf5_file*)

Summarizes total read/write bytes at each timestamp.

Summarizes read/write bytes for each time step using the HDF5 interface instead of converting to a DataFrame or TimeSeries first. Returns a dict of form:

```
{
  "1546761600": {
    "read_bytes": 6135848142.0,
    "write_bytes": 6135848142.0
  },
  "1546761630": {
    "read_bytes": 5261439143.0,
    "write_bytes": 6135848142.0
  },
  "1546761660": {
    "read_bytes": 4321548241.0
    "write_bytes": 6135848142.0,
  },
  ...
}
```

`tokio.cli.summarize_tts.summarize_tts_hdf5` (*hdf5_file*)

Generate summary data based on the contents of TOKIO timeseries HDF5 file

tokio.connectors package

Connector interfaces for pytokio. Each connector provides a Python interface into one component-level monitoring tool.

Submodules

tokio.connectors._hdf5 module

Helper classes and functions used by the HDF5 connector

This contains some of the black magic required to make older H5LMT files compatible with the TOKIO HDF5 schemas and API.

class `tokio.connectors._hdf5.MappedDataset` (*map_function=None, map_kwargs=None, transpose=False, force2d=False, *args, **kwargs*)

Bases: `h5py._hl.dataset.Dataset`

`h5py.Dataset` that applies a function to the results of `__getitem__` before returning the data. Intended to dynamically generate certain datasets that are simple derivatives of others.

`__getitem__` (*key*)

Apply the map function to the result of the parent class and return that transformed result instead. Transpose is very ugly, but required for `h5lmt` support.

`__init__` (*map_function=None, map_kwargs=None, transpose=False, force2d=False, *args, **kwargs*)

Configure a `MappedDataset`

Attach a map function to a `h5py.Dataset` (or derivative) and store the arguments to be fed into that map function whenever this object gets sliced.

Parameters

- **map_function** (*function*) – function to be called on the value returned when parent class is sliced
- **map_kwargs** (*dict*) – kwargs to be passed into map_function
- **transpose** (*bool*) – when True, transpose the results of map_function before returning them. Required by some H5LMT datasets.
- **force2d** (*bool*) – when True, convert a 1d array into a 2d array with a single column. Required by some H5LMT datasets.

`tokio.connectors._hdf5._apply_timestep` (*return_value*, *parent_dataset*, *func*=<function <lambda>>)

Apply a transformation function to a return value

Transforms the data returned when slicing a `h5py.Dataset` object by applying a function to the dataset's values. For example if `return_value` are 'counts per timestep' and you want to convert to 'counts per second', you would specify `func=lambda x, y: x * y`

Parameters

- **return_value** – the value returned when slicing `h5py.Dataset`
- **parent_dataset** – the `h5py.Dataset` which generated `return_value`
- **func** – a function which takes two arguments: the first is `return_value`, and the second is the timestep of `parent_dataset`

Returns A modified version of `return_value` (usually a `numpy.ndarray`)

`tokio.connectors._hdf5._one_column` (*return_value*, *col_idx*, *apply_timestep_func*=None, *parent_dataset*=None)

Extract a specific column from a dataset

Parameters

- **return_value** – the value returned by the parent `DataSet` object that we will modify
- **col_idx** – the column index for the column we are demultiplexing
- **apply_timestep_func** (*function*) – if provided, apply this function with `return_value` as the first argument and the timestep of `parent_dataset` as the second.
- **parent_dataset** (*Dataset*) – if provided, indicates that `return_value` should be divided by the timestep of `parent_dataset` to convert values to rates before returning

Returns A modified version of `return_value` (usually a `numpy.ndarray`)

`tokio.connectors._hdf5.convert_counts_rates` (*hdf5_file*, *from_key*, *to_rates*, **args*, ***kwargs*)

Convert a dataset between counts/sec and counts/timestep

Retrieve a dataset from an HDF5 file, convert it to a `MappedDataset`, and attach a multiply/divide function to it so that subsequent slices return a transformed set of data.

Parameters

- **hdf5_file** (*h5py.File*) – object from which dataset should be loaded
- **from_key** (*str*) – dataset name key to load from `hdf5_file`
- **to_rates** (*bool*) – convert from per-timestep to per-sec (True) or per-sec to per-timestep (False)

Returns A `MappedDataset` configured to convert to/from rates when dereferenced

`tokio.connectors._hdf5.demux_column` (*hdf5_file*, *from_key*, *column*, *apply_timestep_func=None*, **args*, ***kwargs*)

Extract a single column from an HDF5 dataset

MappedDataset map function to present a single column from a dataset as an entire dataset. Required to bridge the h5lmt metadata table (which encodes all metadata ops in a single dataset) and the TOKIO HDF5 format (which encodes a single metadata op per dataset)

Parameters

- **hdf5_file** (*h5py.File*) – the HDF5 file containing the dataset of interest
- **from_key** (*str*) – the dataset name from which a column should be extracted
- **column** (*str*) – the column heading to be returned
- **transpose** (*bool*) – transpose the dataset before returning it

Returns A MappedDataset configured to extract a single column when dereferenced

`tokio.connectors._hdf5.get_timestamps` (*hdf5_file*, *dataset_name*)

Return the timestamps dataset for a given dataset name

`tokio.connectors._hdf5.get_timestamps_key` (*hdf5_file*, *dataset_name*)

Read into an HDF5 file and extract the name of the dataset containing the timestamps correspond to the given *dataset_name*

`tokio.connectors._hdf5.map_dataset` (*hdf5_file*, *from_key*, **args*, ***kwargs*)

Create a MappedDataset

Creates a MappedDataset from an *h5py.File* (or derivative). Functionally similar to *h5py.File.__getitem__()*.

Parameters

- **hdf5_file** (*h5py.File* or `connectors.hdf5.Hdf5`) – file containing dataset of interest
- **from_key** (*str*) – name of dataset to apply mapping function to

`tokio.connectors._hdf5.reduce_dataset_name` (*key*)

Divide a dataset name into its base and modifier

Parameters **dataset_name** (*str*) – Key to reference a dataset that may or may not have a modifier suffix

Returns First string is the base key, the second string is the modifier.

Return type tuple of (*str*, *str* or *None*)

tokio.connectors.cachingdb module

This module provides generic infrastructure for retrieving data from a relational database that contains immutable data. It can use a local caching database (sqlite3) to allow for reanalysis on platforms that cannot access the original remote database or to reduce the load on remote databases.

class `tokio.connectors.cachingdb.CachingDb` (*dbhost=None*, *dbuser=None*, *dbpassword=None*, *dbname=None*, *cache_file=None*)

Bases: `object`

Connect relational database with an optional caching layer interposed.

`__init__` (*dbhost=None, dbuser=None, dbpassword=None, dbname=None, cache_file=None*)

Connect to a relational database.

If instantiated with a `cache_file` argument, all queries will go to that SQLite-based cache database. If this class is not instantiated with a `cache_file` argument, all queries will go out to the remote database.

If none of the connection arguments (`db*`) are specified, do not connect to a remote database and instead rely entirely on the caching database or a separate call to the `connect ()` method.

Parameters

- **dbhost** (*str, optional*) – hostname for the remote database
- **dbuser** (*str, optional*) – username to use when connecting to database
- **dbpassword** (*str, optional*) – password for authenticating to database
- **dbname** (*str, optional*) – name of database to use when connecting
- **cache_file** (*str, optional*) – Path to an SQLite3 database to use as a caching layer.

Variables

- **saved_results** (*dict*) – in-memory data cache, keyed by table names and whose values are dictionaries with keys `rows` and `schema`. `rows` are a list of row tuples returned from earlier queries, and `schema` is the SQL statement required to create the table corresponding to `rows`.
- **last_hit** (*int*) – a flag to indicate whether the last query was found in the caching database or the remote database
- **cache_file** (*str*) – path to the caching database's file
- **cache_db** (*sqlite3.Connection*) – caching database connection handle
- **cache_db_ps** (*str*) – paramstyle of the caching database as defined by [PEP-0249](#)
- **remote_db** – remote database connection handle
- **remote_db_ps** (*str*) – paramstyle of the remote database as defined by [PEP-0249](#)

`_query_mysql` (*query_str, query_variables*)

Run a query against the MySQL database and return the full output.

Parameters

- **query_str** (*str*) – SQL query expressed as a string
- **query_variables** (*tuple*) – parameters to be substituted into `query_str` if `query_str` is a parameterized query

`_query_sqlite3` (*query_str, query_variables*)

Run a query against the cache database and return the full output.

Parameters

- **query_str** (*str*) – SQL query expressed as a string
- **query_variables** (*tuple*) – parameters to be substituted into `query_str` if `query_str` is a parameterized query

`close ()`

Destroy connection objects.

Close the remote database connection handler and reset state of remote connection attributes.

close_cache ()

Close the cache database handler and reset caching db attributes.

connect (*dbhost*, *dbuser*, *dbpassword*, *dbname*)

Establish remote db connection.

Connects to a remote MySQL database and defines the connection handler and paramstyle attributes.

Parameters

- **dbhost** (*str*) – hostname for the remote database
- **dbuser** (*str*) – username to use when connecting to database
- **dbpassword** (*str*) – password for authenticating to database
- **dbname** (*str*) – name of database to use when connecting

connect_cache (*cache_file*)

Open the cache database file and set the handler attribute.

Parameters **cache_file** (*str*) – Path to the SQLite3 caching database file to be used.

drop_cache (*tables=None*)

Flush saved results from memory.

If tables are specified, only drop those tables' results. If no tables are provided, flush everything.

Parameters **tables** (*list*, *optional*) – List of table names (*str*) to flush. If omitted, flush all tables in cache.

query (*query_str*, *query_variables=()*, *table=None*, *table_schema=None*)

Pass a query through all layers of cache and return on the first hit.

If a table is specified, the results of this query can be saved to the cache db into a table of that name.

Parameters

- **query_str** (*str*) – SQL query expressed as a string
- **query_variables** (*tuple*) – parameters to be substituted into *query_str* if *query_str* is a parameterized query
- **table** (*str*, *optional*) – name of table in the cache database to save the results of the query
- **table_schema** (*str*, *optional*) – when *table* is specified, the SQL line to initialize the table in which the query results will be cached.

Returns Tuple of tuples corresponding to rows of fields as returned by the SQL query.

Return type *tuple*

save_cache (*cache_file*)

Commit the in-memory cache to a cache database.

This method is currently very memory-inefficient and not good for caching giant pieces of a database without something wrapping it to feed it smaller pieces.

Note: This manipulates the *cache_db** attributes in a dirty way to prevent closing and re-opening the original cache db. If the *self.open_cache()* is ever changed to include tracking more state, this function must also be updated to retain that state while the old cache db state is being temporarily shuffled out.

Parameters `cache_file` (*str*) – Path to the cache file to be used to write out the cache contents. This file will temporarily pre-empt the `cache_file` attribute and should be a different file.

`tokio.connectors.cachingdb.get_paramstyle_symbol` (*paramstyle*)
 Infer the correct paramstyle for a database.paramstyle

Provides a generic way to determine the paramstyle of a database connection handle. See [PEP-0249](#) for more information.

Parameters `paramstyle` (*str*) – Result of a generic database handler’s *paramstyle* attribute

Returns The string corresponding to the paramstyle of the given database connection.

Return type `str`

tokio.connectors.collectd_es module

Retrieve data generated by collectd and stored in Elasticsearch

class `tokio.connectors.collectd_es.CollectdEs` (**args, **kwargs*)

Bases: `tokio.connectors.es.EsConnection`

collectd-Elasticsearch connection handler

classmethod `from_cache` (**args, **kwargs*)

Initializes an EsConnection object from a cache file.

This path is designed to be used for testing.

Parameters `cache_file` (*str*) – Path to the JSON formatted list of pages

query_cpu (*start, end*)

Query Elasticsearch for collectd cpu plugin data.

Parameters

- **start** (*datetime.datetime*) – lower bound for query (inclusive)
- **end** (*datetime.datetime*) – upper bound for query (exclusive)

query_disk (*start, end*)

Query Elasticsearch for collectd disk plugin data.

Parameters

- **start** (*datetime.datetime*) – lower bound for query (inclusive)
- **end** (*datetime.datetime*) – upper bound for query (exclusive)

query_memory (*start, end*)

Query Elasticsearch for collectd memory plugin data.

Parameters

- **start** (*datetime.datetime*) – lower bound for query (inclusive)
- **end** (*datetime.datetime*) – upper bound for query (exclusive)

query_timeseries (*query_template, start, end, source_filter=None, filter_function=None, flush_every=None, flush_function=None*)

Map connection-wide attributes to `super(self).query_timeseries` arguments

Parameters

- **query_template** (*dict*) – a query object containing at least one `@timestamp` field
- **start** (*datetime.datetime*) – lower bound for query (inclusive)
- **end** (*datetime.datetime*) – upper bound for query (exclusive)
- **source_filter** (*bool or list*) – Return all fields contained in each document's `_source` field if True; otherwise, only return source fields contained in the provided list of str. If None, use the default for this connector.
- **filter_function** (*function, optional*) – Function to call before each set of results is appended to the `scroll_pages` attribute; if specified, return value of this function is what is appended. If None, use the default for this connector.
- **flush_every** (*int or None*) – trigger the flush function once the number of docs contained across all `scroll_pages` reaches this value. If None, do not apply `flush_function`. If None, use the default for this connector.
- **flush_function** (*function, optional*) – function to call when `flush_every` docs are retrieved. If None, use the default for this connector.

to_dataframe()

Converts `self.scroll_pages` to a DataFrame

Returns Contents of the last query's pages

Return type `pandas.DataFrame`

tokio.connectors.common module

Common methods and classes used by connectors

class `tokio.connectors.common.CacheableDict` (*input_file=None*)

Bases: `dict`

Generic class to support connectors that are dicts that can be cached as JSON

When deriving from this class, the child object will have to define its own `load_native()` method to be invoked when `input_file` is not JSON.

__init__ (*input_file=None*)

Either initialize as empty or load from cache

Parameters `input_file` (*str*) – Path to either a JSON file representing the dict or a native file that will be parsed into a JSON-compatible format

_save_cache (*output, **kwargs*)

Generates serialized representation of self

Parameters `output` – Object with a `.write()` method into which the serialized form of self will be passed

load (*input_file=None*)

Wrapper around the filetype-specific loader.

Infer the type of input being given, dispatch the correct loading function, and populate keys/values.

Parameters `input_file` (*str or None*) – The input file to load. If not specified, uses whatever `self.input_file` is

load_json (*input_file=None*)

Loads input from serialized JSON

Load the serialized format of this object, encoded as a json dictionary. This is the converse of the `save_cache()` method.

Parameters `input_file` (*str or None*) – The input file to load. If not specified, uses whatever `self.input_file` is

load_native (*input_file=None*)

Parse an uncached, native object

This is a stub that should be overloaded on derived classes.

Parameters `input_file` (*str or None*) – The input file to load. If not specified, uses whatever `self.input_file` is

save_cache (*output_file=None, **kwargs*)

Serializes `self` into a JSON output.

Save the dictionary in a JSON file. This output can be read back in using `load_json()`.

Parameters

- **output_file** (*str or None*) – Path to file to which json should be written. If `None`, write to `stdout`. Default is `None`.
- **kwargs** (*dict*) – Additional arguments to be passed to `json.dumps()`

class `tokio.connectors.common.SubprocessOutputDict` (*cache_file=None, from_string=None, silent_errors=False*)

Bases: `dict`

Generic class to support connectors that parse the output of a subprocess

When deriving from this class, the child object will have to

1. Define `subprocess_cmd` after initializing this parent object
2. Define `self.__repr__` (if necessary)
3. Define its own `self.load_str`
4. Define any introspective analysis methods

_load_subprocess (**args*)

Run a subprocess and pass its `stdout` to a self-initializing parser

load (*cache_file=None*)

Load based on initialization state of object

Parameters `cache_file` (*str or None*) – The cached input file to load. If not specified, uses whatever `self.cache_file` is

load_cache (*cache_file=None*)

Load subprocess output from a cached text file

Parameters `cache_file` (*str or None*) – The cached input file to load. If not specified, uses whatever `self.cache_file` is

load_str (*input_str*)

Load subprocess output from a string

Parameters `input_str` (*str*) – The text that came from the subprocess's `stdout` and should be parsed by this method.

save_cache (*output_file=None*)

Serialize subprocess output to a text file

Parameters `output_file` (*str*) – Path to a file to which the output cache should be written. If None, write to stdout.

```
class tokio.connectors.common.SubprocessOutputList (cache_file=None,
                                                    from_string=None,
                                                    silent_errors=False)
```

Bases: `list`

Generic class to support connectors that parse the output of a subprocess

When deriving from this class, the child object will have to

1. Define `subprocess_cmd` after initializing this parent object
2. Define `self.__repr__` (if necessary)
3. Define its own `self.load_str`
4. Define any introspective analysis methods

`_load_subprocess` (**args*)

Run a subprocess and pass its stdout to a self-initializing parser

`load` (*cache_file=None*)

Load based on initialization state of object

Parameters `cache_file` (*str* or *None*) – The cached input file to load. If not specified, uses whatever `self.cache_file` is

`load_cache` (*cache_file=None*)

Load subprocess output from a cached text file

Parameters `cache_file` (*str* or *None*) – The cached input file to load. If not specified, uses whatever `self.cache_file` is

`load_str` (*input_str*)

Load subprocess output from a string

Parameters `input_str` (*str*) – The text that came from the subprocess’s stdout and should be parsed by this method.

`save_cache` (*output_file=None*)

Serialize subprocess output to a text file

Parameters `output_file` (*str*) – Path to a file to which the output cache should be written. If None, write to stdout.

```
tokio.connectors.common.walk_file_collection (input_source)
```

Walk all member files of an input source.

Iterator that visits every member of an input source (either directory or tarfile) and yields its file name, last modify time, and a file handle to its contents.

Parameters `input_source` (*str*) – A path to either a directory containing files or a tarfile containing files.

Yields *tuple* – Attributes for a member of `input_source` with the following data:

- `str`: fully qualified path corresponding to its name
- `float`: last modification time expressed as seconds since epoch
- `file`: handle to access the member’s contents

tokio.connectors.craysdb module

This connection provides an interface to the Cray XT/XC service database (SDB). It is intended to be used to determine information about a node's configuration within the network fabric to provide topological information.

class `tokio.connectors.craysdb.CraySdbProc` (*args, **kwargs)

Bases: `tokio.connectors.common.SubprocessOutputDict`

Dictionary subclass that self-populates with Cray SDB data.

Presents certain views of the Cray Service Database (SDB) as a dictionary-like object through the Cray SDB CLI.

`__init__` (*args, **kwargs)

Load the processor configuration table from the SDB.

Parameters

- ***args** – Passed to `tokio.connectors.common.SubprocessOutputDict`
- ****kwargs** – Passed to `tokio.connectors.common.SubprocessOutputDict`

`__repr__` ()

Serialize self in a format compatible with `xtdb2proc`.

Returns the object in the same format as the `xtdb2proc` output so that this object can be circularly serialized and deserialized.

Returns: str: String representation of the processor mapping table in a format compatible with the output of `xtdb2proc`.

`load_str` (*input_str*)

Load the `xtdb2proc` data for a Cray system.

Parses the `xtdb2proc` output text and inserts keys/values into self.

Parameters `input_str` (*str*) – stdout of the `xtdb2proc` command

tokio.connectors.darshan module

Connect to Darshan logs.

This connector provides an interface into Darshan logs created by Darshan 3.0 or higher and represents the counters and data contained therein as a Python dictionary. This dictionary has the following structure, where `block` denote literal key names.

- `header` which contains key-value pairs corresponding to each line in the header. `exe` and `metadata` are lists; the other keys correspond to a single scalar value.
 - `compression`, `end_time`, `end_time_string`, `exe`, etc
- `counters`
 - `modulename` which is `posix`, `lustre`, `stdio`, etc
 - * `recordname`, which is usually the full path to a file opened by the profiled application `_or_` `__perf` (contains performance summary metrics) or `_total` (contains aggregate file statistics)
 - `ranknum` which is a string (0, 1, etc or -1)
 - `counternames`, which depends on the Darshan module defined by `modulename` above
- `mounts` which is the mount table with keys of a path to a mount location and values of the file system type

The *counter names* are module-specific and have their module name prefix stripped off. The following counter names are examples of what a Darshan log may expose through this connector for the `posix` module:

- `BYTES_READ` and `BYTES_WRITTEN` - number of bytes read/written to the file
- `MAX_BYTE_WRITTEN` and `MAX_BYTE_READ` - highest byte written/read; useful if an application re-reads or re-writes a lot of data
- `WRITES` and `READS` - number of write and read ops issued
- `F_WRITE_TIME` and `F_READ_TIME` - amount of time spent inside write and read calls (in seconds)
- `F_META_TIME` - amount of time spent in metadata (i.e., non-read/write) calls

Similarly the `lustre` module provides the following counter keys:

- `MDTS` - number of MDTs in the underlying file system
- `OSTS` - number of OSTs in the underlying file system
- `OST_ID_0` - the OBD index for the 0th OST over which the file is striped
- `STRIPE_OFFSET` - the setting used to define stripe offset when the file was created
- `STRIPE_SIZE` - the size, in bytes, of each stripe
- `STRIPE_WIDTH` - how many OSTs the file touches

Note: This connector presently relies on `darshan-parser` to convert the binary logs to ASCII, then convert the ASCII into Python objects. In the future, we plan on using the Python API provided by `darshan-utils` to circumvent the ASCII translation.

```
class tokio.connectors.darshan.Darshan (log_file=None, *args, **kwargs)
```

```
    Bases: tokio.connectors.common.SubprocessOutputDict
```

```
    __init__ (log_file=None, *args, **kwargs)
```

```
        Initialize the object from either a Darshan log or a cache file.
```

```
        Configures the object's internal state to operate on a Darshan log file or a cached JSON representation of a previously processed Darshan log.
```

Parameters

- **log_file** (*str, optional*) – Path to a Darshan log to be processed
- **cache_file** (*str, optional*) – Path to a Darshan log's contents cached
- ***args** – Passed to `tokio.connectors.common.SubprocessOutputDict`
- ****kwargs** – Passed to `tokio.connectors.common.SubprocessOutputDict`

Variables **log_file** (*str*) – Path to the Darshan log file to load

```
    __repr__ ()
```

```
        Serialize self into JSON.
```

Returns JSON representation of the object

Return type `str`

```
    _darshan_parser ()
```

```
        Call darshan-parser to initialize values in self
```

```
    _load_subprocess_iter (*args)
```

```
        Run a subprocess and pass its stdout to a self-initializing parser
```

`_parse_darshan_parser` (*lines*)

Load values from output of darshan-parser

Parameters **lines** – Any iterable that produces lines of darshan-parser output

`darshan_parser_base` (*modules=None, counters=None*)

Populate data produced by darshan-parser --base

Runs the darshan-parser --base and convert all results into key-value pairs which are inserted into the object.

Parameters

- **modules** (*list of str*) – If specified, only return data from the given Darshan modules
- **counters** (*list of str*) – If specified, only return data for the given counters

Returns Dictionary containing all key-value pairs generated by running darshan-parser --base. These values are also accessible via the *BASE* key in the object.

Return type `dict`

`darshan_parser_perf` (*modules=None, counters=None*)

Populate data produced by darshan-parser --perf

Runs the darshan-parser --perf and convert all results into key-value pairs which are inserted into the object.

Parameters

- **modules** (*list of str*) – If specified, only return data from the given Darshan modules
- **counters** (*list of str*) – If specified, only return data for the given counters

Returns Dictionary containing all key-value pairs generated by running darshan-parser --perf. These values are also accessible via the *PERF* key in the object.

Return type `dict`

`darshan_parser_total` (*modules=None, counters=None*)

Populate data produced by darshan-parser --total

Runs the darshan-parser --total and convert all results into key-value pairs which are inserted into the object.

Parameters

- **modules** (*list of str*) – If specified, only return data from the given Darshan modules
- **counters** (*list of str*) – If specified, only return data for the given counters

Returns Dictionary containing all key-value pairs generated by running darshan-parser --total. These values are also accessible via the *TOTAL* key in the object.

Return type `dict`

`load` ()

Load based on initialization state of object

Parameters **cache_file** (*str or None*) – The cached input file to load. If not specified, uses whatever self.cache_file is

`load_str(input_str)`

Load from either a json cache or the output of darshan-parser

Parameters `input_str` – Either (1) stdout of the darshan-parser command as a string, (2) the json-encoded representation of a Darshan object that can be deserialized to initialize self, or (3) an iterator that produces the output of darshan-parser line-by-line

`tokio.connectors.darshan.parse_base_counters(line)`

Parse a counter line from darshan-parser --base.

Parse the line containing an actual counter's data. It is a tab-delimited line of the form

module, rank, record_id, counter, value, file_name, mount_pt, fs_type

Parameters `line (str)` – A single line of output from darshan-parser --base

Returns

Returns a tuple containing eight values. If `line` is not a valid counter line, all values will be None. The returned values are:

0. module name
1. MPI rank
2. record id
3. counter name
4. counter value
5. file name
6. mount point
7. file system type

Return type tuple

`tokio.connectors.darshan.parse_filename_metadata(filename)`

Extracts metadata from a Darshan log's file name

Parameters `filename (str)` – Name of a Darshan log file. Can be basename or a full path.

Returns

key-value pairs describing the metadata extracted from the file name.

Return type dict

`tokio.connectors.darshan.parse_header(line)`

Parse the header lines of darshan-parser.

Accepts a line that may or may not be a header line as printed by darshan-parser. Such header lines take the form:

```
# darshan log version: 3.10
# compression method: ZLIB
# exe: /home/user/bin/myjob.exe --whatever
# uid: 69615
```

If it is a valid header line, return a key-value pair corresponding to its decoded contents.

Parameters `line (str)` – A single line of output from darshan-parser

Returns Returns a (key, value) corresponding to the key and value decoded from the header `line`, or (None, None) if the line does not appear to contain a known header field.

Return type tuple`tokio.connectors.darshan.parse_mounts(line)`

Parse a mount table line from darshan-parser.

Accepts a line that may or may not be a mount table entry from darshan-parser. Such lines take the form:

```
# mount entry: /usr/lib64/libibverbs.so.1.0.0 dvs
```

If *line* is a valid mount table entry, return a key-value representation of its contents.**Parameters** `line` (*str*) – A single line of output from darshan-parser**Returns** Returns a (key, value) corresponding to the mount table entry, or (None, None) if the line is not a valid mount table entry.**Return type** tuple`tokio.connectors.darshan.parse_perf_counters(line)`

Parse a counter line from darshan-parser --perf.

Parse a line containing counter data from darshan-parser --perf. Such lines look like:

```
# total_bytes: 2199023259968
# unique files: slowest_rank_io_time: 0.000000
# shared files: time_by_cumul_io_only: 39.992327
# agg_perf_by_slowest: 28670.996545
```

Parameters `line` (*str*) – A single line of output from darshan-parser --perf**Returns** Returns a single (key, value) pair corresponding to the performance metric encoded in *line*. If *line* is not a valid performance counter line, (None, None) is returned.**Return type** tuple`tokio.connectors.darshan.parse_total_counters(line)`

Parse a counter line from darshan-parser --total.

Parse a line containing counter data from darshan-parser --total. Such lines are of the form:

```
total_MPIO_F_READ_END_TIMESTAMP: 0.000000
```

Parameters `line` (*str*) – A single line of output from darshan-parser --total**Returns** Returns a single (key, value) pair corresponding to a counted metric and its total value. If *line* is not a valid counter line, (None, None) are returned.**Return type** tuple**tokio.connectors.es module**

Retrieve data stored in Elasticsearch

This module provides a wrapper around the Elasticsearch connection handler and methods to query, scroll, and process pages of scrolling data.

class `tokio.connectors.es.EsConnection` (*host*, *port*, *index=None*, *scroll_size='1m'*,
page_size=10000, *timeout=30*, ***kwargs*)Bases: `object`

Elasticsearch connection handler.

Wrapper around an Elasticsearch connection context that provides simpler scrolling functionality for very long documents and callback functions to be run after each page is retrieved.

`__init__` (*host*, *port*, *index=None*, *scroll_size='1m'*, *page_size=10000*, *timeout=30*, ***kwargs*)

Configure and connect to an Elasticsearch endpoint.

Parameters

- **host** (*str*) – hostname for the Elasticsearch REST endpoint
- **port** (*int*) – port where Elasticsearch REST endpoint is bound
- **index** (*str*) – name of index against which queries will be issued
- **scroll_size** (*str*) – how long to keep the scroll search context open (e.g., “1m” for 1 minute)
- **page_size** (*int*) – how many documents should be returned per scrolled page (e.g., 10000 for 10k docs per scroll)
- **timeout** (*int*) – how many seconds to wait for a response from Elasticsearch before the query should time out

Variables

- **client** – Elasticsearch connection handler
- **page** (*dict*) – last page retrieved by a query
- **scroll_pages** (*list*) – dictionary of pages retrieved by query
- **index** (*str*) – name of index against which queries will be issued
- **connect_host** (*str*) – hostname for Elasticsearch REST endpoint
- **connect_port** (*int*) – port where Elasticsearch REST endpoint is bound
- **connect_timeout** (*int*) – seconds before query should time out
- **page_size** (*int*) – max number of documents returned per page
- **scroll_size** (*int*) – duration to keep scroll search context open
- **scroll_id** – identifier for the scroll search context currently in use
- **sort_by** (*str*) – field by which Elasticsearch should sort results before returning them as query results
- **fake_pages** (*list*) – A list of `page` structures that should be returned by `self.scroll()` when the `elasticsearch` module is not actually available. Used only for debugging.
- **local_mode** (*bool*) – If True, retrieve query results from `self.fake_pages` instead of attempting to contact an Elasticsearch server
- **kwargs** (*dict*) – Passed to `elasticsearch.Elasticsearch.__init__` if host and port are defined

`_process_page` ()

Remove a page from the incoming queue and append it

Takes the last received page (`self.page`), updates the internal state of the scroll operation, updates some internal counters, calls the flush function if applicable, and applies the filter function. Then appends the results to `self.scroll_pages`.

Returns True if hits were appended or not

Return type `bool`

close()

Close and invalidate the connection context.

connect (kwargs)**

Instantiate a connection and retain the connection context.

Parameters `kwargs` (*dict*) – Passed to `elasticsearch.Elasticsearch.__init__`

classmethod from_cache (cache_file)

Initializes an `EsConnection` object from a cache file.

This path is designed to be used for testing.

Parameters `cache_file` (*str*) – Path to the JSON formatted list of pages

query (query)

Issue an Elasticsearch query.

Issues a query and returns the resulting page. If the query included a scrolling request, the `scroll_id` attribute is set so that scrolling can continue.

Parameters `query` (*dict*) – Dictionary representing the query to issue

Returns The page resulting from the issued query.

Return type `dict`

query_and_scroll (query, source_filter=True, filter_function=None, flush_every=None, flush_function=None)

Issue a query and retain all results.

Issues a query and scrolls through every resulting page, optionally applying in situ logic for filtering and flushing. All resulting pages are appended to the `scroll_pages` attribute of this object.

The `scroll_pages` attribute must be wiped by whatever is consuming it; if this does not happen, `query_and_scroll()` will continue appending results to the results of previous queries.

Parameters

- **query** (*dict*) – Dictionary representing the query to issue
- **source_filter** (*bool or list*) – Return all fields contained in each document's `_source` field if `True`; otherwise, only return source fields contained in the provided list of `str`.
- **filter_function** (*function, optional*) – Function to call before each set of results is appended to the `scroll_pages` attribute; if specified, return value of this function is what is appended.
- **flush_every** (*int or None*) – trigger the flush function once the number of docs contained across all `scroll_pages` reaches this value. If `None`, do not apply `flush_function`.
- **flush_function** (*function, optional*) – function to call when `flush_every` docs are retrieved.

query_timeseries (query_template, start, end, source_filter=True, filter_function=None, flush_every=None, flush_function=None)

Craft and issue query bounded by time

Parameters

- **query_template** (*dict*) – a query object containing at least one `@timestamp` field

- **start** (*datetime.datetime*) – lower bound for query (inclusive)
- **end** (*datetime.datetime*) – upper bound for query (exclusive)
- **source_filter** (*bool or list*) – Return all fields contained in each document’s `_source` field if True; otherwise, only return source fields contained in the provided list of str.
- **filter_function** (*function, optional*) – Function to call before each set of results is appended to the `scroll_pages` attribute; if specified, return value of this function is what is appended.
- **flush_every** (*int or None*) – trigger the flush function once the number of docs contained across all `scroll_pages` reaches this value. If None, do not apply `flush_function`.
- **flush_function** (*function, optional*) – function to call when `flush_every` docs are retrieved.

save_cache (*output_file=None*)

Persist the response of the last query to a file

This is a little different from other connectors’ `save_cache()` methods in that it only saves down the state of the last query’s results. It does not save any connection information and does not restore the state of a previous `EsConnection` object.

Its principal intention is to be used with testing.

Parameters **output_file** (*str or None*) – Path to file to which json should be written. If None, write to stdout. Default is None.

scroll ()

Request the next page of results.

Requests the next page of results for a query that is scrolling. This can only be performed if the `scroll_id` attribute is set (e.g., by the `query()` method).

Returns The next page in the scrolling context.

Return type `dict`

to_dataframe (*fields*)

Converts `self.scroll_pages` to CSV

Returns Contents of the last query’s pages in CSV format

Return type `str`

`tokio.connectors.es.build_timeseries_query` (*orig_query, start, end, start_key='@timestamp', end_key=None*)

Create a query object with time ranges bounded.

Given a query dict and a start/end datetime object, return a new query object with the correct time ranges bounded. Relies on `orig_query` containing at least one `@timestamp` field to indicate where the time ranges should be inserted.

If `orig_query` is querying records that contain both a “start” and “end” time (e.g., a job) rather than a discrete point in time (e.g., a sampled metric), `start_key` and `end_key` can be used to modify the query to return all records that overlapped with the interval specified by `start_time` and `end_time`.

Parameters

- **orig_query** (*dict*) – A query object containing at least one `@timestamp` field.
- **start** (*datetime.datetime*) – lower bound for query (inclusive)

- **end** (*datetime.datetime*) – upper bound for query (exclusive)
- **start_key** (*str*) – The key containing a timestamp against which a time range query should be applied.
- **end_key** (*str*) – The key containing a timestamp against which the upper bound of the time range should be applied. If None, treat *start_key* as a single point in time rather than the start of a recorded process.

Returns A query object with all instances of @timestamp bounded by *start* and *end*.

Return type dict

`tokio.connectors.es.mutate_query` (*mutable_query*, *field*, *value*, *term*='term')

Inserts a new condition into a query object

See <https://www.elastic.co/guide/en/elasticsearch/reference/current/term-level-queries.html> for complete documentation.

Parameters

- **mutable_query** (*dict*) – a query object to be modified
- **field** (*str*) – the field to which a term query will be applied
- **value** – the value to match for the term query
- **term** (*str*) – one of the following: term, terms, terms_set, range, exists, prefix, wildcard, regexp, fuzzy, type, ids.

Returns Nothing. *mutable_query* is updated in place.

tokio.connectors.esnet_snmp module

Provides interfaces into ESnet’s SNMP REST API

Documentation for the REST API is here:

<http://es.net/network-r-and-d/data-for-researchers/snmp-api/>

Notes

This connector relies either on the `esnet_snmp_url` configuration value being set in the `pytokio` configuration or the `PYOKIO_ESNET_SNMP_URI` being defined in the runtime environment.

Examples

Retrieving the data of multiple endpoints (ESnet routers) and interfaces is a common pattern. To do this, the `EsnetSnmp` object should be initialized with only the intended start/end times, and the object should be asynchronously populated using calls to `EsnetSnmp.get_interface_counters`:

```
import datetime
import tokio.connectors.esnet_snmp

ROUTER = 'sunn-cr5'
INTERFACE = 'to_nersc_ip-d_v4'
TARGET_DATE = datetime.datetime.today() - datetime.timedelta(days=1)

# Because the ESnet API treats the end date as inclusive, we subtract
```

(continues on next page)

(continued from previous page)

```
# one second to avoid counting the first measurement of the following
# day.
esnetsnmp = tokio.connectors.esnet_snmp.EsnetSnmp(
    start=TARGET_DATE,
    end=TARGET_DATE + datetime.timedelta(days=1, seconds=-1))
for direction in 'in', 'out':
    esnetsnmp.get_interface_counters(
        endpoint=ROUTER,
        interface=INTERFACE,
        direction=direction,
        agg_func='average')

for direction in 'in', 'out':
    bytes_per_sec = list(esnetsnmp[ROUTER][INTERFACE][direction].values())
    total_bytes = sum(bytes_per_sec) * esnetsnmp.timestep
    print("%s:%s saw %.2f TiB %s" % (
        ROUTER,
        INTERFACE,
        total_bytes / 2**40,
        direction))
```

For simple queries, it is sufficient to specify the endpoint, interface, and direction directly in the initialization:

```
esnetsnmp = tokio.connectors.esnet_snmp.EsnetSnmp(
    start=TARGET_DATE,
    end=TARGET_DATE + datetime.timedelta(days=1, seconds=-1)
    endpoint=ROUTER,
    interface=INTERFACE,
    direction="in")
print("Total bytes in: %.2f" % (
    sum(list(esnetsnmp[ROUTER][INTERFACE]["in"].values())) / 2**40))
```

class tokio.connectors.esnet_snmp.**EsnetSnmp**(*start, end, endpoint=None, interface=None, direction=None, agg_func=None, interval=None, **kwargs*)

Bases: *tokio.connectors.common.CacheableDict*

Container for ESnet SNMP counters

Dictionary with structure:

```
{
    "endpoint0": {
        "interface_x": {
            "in": {
                timestamp1: value1,
                timestamp2: value2,
                timestamp3: value3,
                ...
            },
            "out": { ... }
        },
        "interface_y": { ... }
    },
    "endpoint1": { ... }
}
```

Various methods are provided to access the data of interest.

`__init__` (*start*, *end*, *endpoint=None*, *interface=None*, *direction=None*, *agg_func=None*, *interval=None*, ***kwargs*)

Retrieves data rate data for an ESnet endpoint

Initializes the object with a start and end time. Optionally runs a REST API query if endpoint, interface, and direction are provided. Assumes that once the start/end time have been specified, they should not be changed.

Parameters

- **start** (*datetime.datetime*) – Start of interval to retrieve, inclusive
- **end** (*datetime.datetime*) – End of interval to retrieve, inclusive
- **endpoint** (*str*, *optional*) – Name of the ESnet endpoint whose data is being retrieved
- **interface** (*str*, *optional*) – Name of the ESnet endpoint interface on the specified endpoint
- **direction** (*str*, *optional*) – “in” or “out” to signify data input into ESnet or data output from ESnet
- **agg_func** (*str*, *optional*) – Specifies the reduction operator to be applied over each interval; must be one of “average,” “min,” or “max.” If None, uses the ESnet default.
- **interval** (*int*, *optional*) – Resolution, in seconds, of the data to be returned. If None, uses the ESnet default.
- **kwargs** (*dict*) – arguments to pass to `super.__init__()`

Variables

- **start** (*datetime.datetime*) – Start of interval represented by this object, inclusive
- **end** (*datetime.datetime*) – End of interval represented by this object, inclusive
- **start_epoch** (*int*) – Seconds since epoch for `self.start`
- **end_epoch** (*int*) – Seconds since epoch for `self.end`

`__insert_result` ()

Parse the raw output of the REST API and update self

ESnet’s REST API will return an object like:

```
{
  "agg": "30",
  "begin_time": 1517471100,
  "end_time": 1517471910,
  "cf": "average",
  "data": [
    [
      1517471100,
      5997486471.266666
    ],
    ...
    [
      1517471910,
      189300026.8
    ]
  ]
}
```

(continues on next page)

(continued from previous page)

```

    ]
}

```

Parameters `result` (*dict*) – JSON structure returned by the ESnet REST API

Returns True if data inserted without errors; False otherwise

Return type `bool`

`gen_url` (*endpoint, interface, direction, agg_func=None, interval=None, **kwargs*)

Builds parameters to be passed to `requests.get()` to retrieve data from the REST API.

This is factored out so that the URI generation can be tested without a working REST endpoint.

Parameters

- **endpoint** (*str*) – Name of ESnet endpoint (usually a router identifier)
- **interface** (*str*) – Name of the ESnet endpoint interface
- **direction** (*str*) – “in” or “out” to signify data input into ESnet or data output from ESnet
- **agg_func** (*str or None*) – Specifies the reduction operator to be applied over each interval; must be one of “average,” “min,” or “max.” If None, uses the ESnet default.
- **interval** (*int or None*) – Resolution, in seconds, of the data to be returned. If None, uses the ESnet default.

Returns `kwargs` to be passed directly to `requests.get()`

Return type `dict`

`get_interface_counters` (*endpoint, interface, direction, agg_func=None, interval=None, **kwargs*)

Retrieves data rate data for an ESnet endpoint

This is factored the way it is so that it can be subclassed to use a faked remote REST endpoint for testing.

Parameters

- **endpoint** (*str*) – Name of ESnet endpoint (usually a router identifier)
- **interface** (*str*) – Name of the ESnet endpoint interface
- **direction** (*str*) – “in” or “out” to signify data input into ESnet or data output from ESnet
- **agg_func** (*str or None*) – Specifies the reduction operator to be applied over each interval; must be one of “average,” “min,” or “max.” If None, uses the ESnet default.
- **interval** (*int or None*) – Resolution, in seconds, of the data to be returned. If None, uses the ESnet default.
- **kwargs** (*dict*) – Extra parameters to pass to `requests.get()`

Returns raw return from the REST API call

Return type `dict`

load_json (*args, **kwargs)

Loads input from serialized JSON

Need to coerce timestamp keys back into ints from strings

to_dataframe (multiindex=False)

Return data as a Pandas DataFrame

Parameters multiindex (*bool*) – If True, return a DataFrame indexed by timestamp, endpoint, interface, and direction

`tokio.connectors.esnet_snmp._get_interval_result` (*result*)

Parse the raw output of the REST API output and return the timestep

Parameters result (*dict*) – the raw JSON output of the ESnet REST API

Returns int or None

`tokio.connectors.esnet_snmp._get_url` (*url, params, **kwargs*)

Wraps the requests.get() call and returns the results as a decoded JSON object.

Parameters

- **url** (*str*) – URL to request
- **params** (*dict*) – Same as requests.get(params)
- **kwargs** – Passed directly to requests.get()

Returns Result of the remote query

Return type dict

tokio.connectors.globuslogs module

Provides an interface for Globus and GridFTP transfer logs

Globus logs are ASCII files that generally look like:

```
DATE=20190809091437.927804 HOST=dttn11.nersc.gov PROG=globus-gridftp-server NL.
↪EVNT=FTP_INFO START=20190809091437.884224 USER=glock FILE=/home/g/glock/results0.
↪tar.gz BUFFER=235104 BLOCK=262144 NBYTES=35616 VOLUME=/ STREAMS=4 STRIPES=1 DEST=[0.
↪0.0.0] TYPE=RETR CODE=226
DATE=20190809091438.022479 HOST=dttn11.nersc.gov PROG=globus-gridftp-server NL.
↪EVNT=FTP_INFO START=20190809091437.963894 USER=glock FILE=/home/g/glock/results1.
↪tar.gz BUFFER=235104 BLOCK=262144 NBYTES=35616 VOLUME=/ STREAMS=4 STRIPES=1 DEST=[0.
↪0.0.0] TYPE=RETR CODE=226
DATE=20190809091438.370175 HOST=dttn11.nersc.gov PROG=globus-gridftp-server NL.
↪EVNT=FTP_INFO START=20190809091438.314961 USER=glock FILE=/home/g/glock/results2.
↪tar.gz BUFFER=235104 BLOCK=262144 NBYTES=35616 VOLUME=/ STREAMS=4 STRIPES=1 DEST=[0.
↪0.0.0] TYPE=RETR CODE=226
```

The keys and values are pretty well demarcated, with the only hiccup being around file names that contain spaces.

class `tokio.connectors.globuslogs.GlobusLog` (*args, **kwargs)

Bases: `tokio.connectors.common.SubprocessOutputList`

Interface into a Globus transfer log

Parses a Globus transfer log which looks like:


```

DATE=20190809091437.927804 HOST=dtn11.nersc.gov PROG=globus-gridftp-server NL.
↪EVNT=FTP_INFO START=20190809091437.884224 USER=glock FILE=/home/g/glock/
↪results0.tar.gz BUFFER=235104 BLOCK=262144 NBYTES=35616 VOLUME=/ STREAMS=4
↪STRIPES=1 DEST=[0.0.0.0] TYPE=RETR CODE=226
DATE=20190809091438.022479 HOST=dtn11.nersc.gov PROG=globus-gridftp-server NL.
↪EVNT=FTP_INFO START=20190809091437.963894 USER=glock FILE=/home/g/glock/
↪results1.tar.gz BUFFER=235104 BLOCK=262144 NBYTES=35616 VOLUME=/ STREAMS=4
↪STRIPES=1 DEST=[0.0.0.0] TYPE=RETR CODE=226
DATE=20190809091438.370175 HOST=dtn11.nersc.gov PROG=globus-gridftp-server NL.
↪EVNT=FTP_INFO START=20190809091438.314961 USER=glock FILE=/home/g/glock/
↪results2.tar.gz BUFFER=235104 BLOCK=262144 NBYTES=35616 VOLUME=/ STREAMS=4
↪STRIPES=1 DEST=[0.0.0.0] TYPE=RETR CODE=226

```

and represents the data in a list-like form:

```

[
  {
    "BLOCK": 262144,
    "BUFFER": 87040,
    "CODE": "226",
    "DATE": 1565445916.0,
    "DEST": [
      "198.125.208.14"
    ],
    "FILE": "/home/g/glock/results_08_F...",
    "HOST": "dtn11.nersc.gov",
    "NBYTES": 6341890048,
    "NL.EVNT": "FTP_INFO",
    "PROG": "globus-gridftp-server",
    "START": 1565445895.0,
    "STREAMS": 1,
    "STRIPES": 1,
    "TYPE": "STOR",
    "USER": "glock",
    "VOLUME": "/"
  },
  ...
]

```

where each list item is a dictionary encoding a single transfer log line. The keys are exactly as they appear in the log file itself, and it is the responsibility of downstream analysis code to attribute meaning to each key.

classmethod `from_file` (*cache_file*)

Instantiates from a cache file

classmethod `from_str` (*input_str*)

Instantiates from a string

load_str (*input_str*)

Parses text from a Globus FTP log

Iterates through a multi-line string and converts each line into a dictionary of key-value pairs.

Parameters `input_str` (*str*) – Multi-line string containing a single Globus log transfer record on each line.

`tokio.connectors.globuslogs._listify_ips` (*ip_str*)

Breaks a string encoding a list of destinations into a list of destinations

Parameters `ip_str` (*str*) – A list of destination hosts encoded as a string

Returns A list of destination host strings

Return type `list`

tokio.connectors.hdf5 module

Provide a TOKIO-aware HDF5 class that knows how to interpret schema versions encoded in a TOKIO HDF5 file and translate a universal schema into file-specific schemas. Also supports dynamically mapping static HDF5 datasets into new derived datasets dynamically.

class `tokio.connectors.hdf5.Hdf5` (*args, **kwargs)

Bases: `h5py._hl.files.File`

Hdf5 file class with extra hooks to parse different schemas

Provides an `h5py.File`-like class with added methods to provide a generic API that can decode different schemata used to store file system load data.

Variables

- **`always_translate`** (*bool*) – If True, looking up datasets by keys will always attempt to map that key to a new dataset according to the schema even if the key matches the name of an existing dataset.
- **`dataset_providers`** (*dict*) – Map of logical dataset names (keys) to dicts that describe the functions used to convert underlying literal dataset data into the format expected when dereferencing the logical dataset name.
- **`schema`** (*dict*) – Map of logical dataset names (keys) to the literal dataset names in the underlying file (values)
- **`_version`** (*str*) – Defined and used at initialization time to determine what schema to apply to map the HDF5 connector API to the underlying HDF5 file.
- **`_timesteps`** (*dict*) – Keyed by dataset name (*str*) and has values corresponding to the timestep (in seconds) between each sampled datum in that dataset.

`__getitem__` (*key*)

Resolve dataset names into actual data

Provides a single interface through which standard keys can be dereferenced and a semantically consistent view of data is returned regardless of the schema of the underlying HDF5 file.

Passes through the underlying `h5py.Dataset` via direct access or a 1:1 mapping between standardized key and an underlying dataset name, or a `numpy` array if an underlying `h5py.Dataset` must be transformed to match the structure and semantics of the data requested.

Can also suffix datasets with special meta-dataset names (e.g., “/missing”) to access data that is related to the root dataset.

Parameters **key** (*str*) – The standard name of a dataset to be accessed.

Returns

- `h5py.Dataset` if key is a literal dataset name
- `h5py.Dataset` if key maps directly to a literal dataset name given the file schema version
- `numpy.ndarray` if key maps to a provider function that can calculate the requested data

Return type `h5py.Dataset` or `numpy.ndarray`

`__init__` (**args, **kwargs*)
Initialize an HDF5 file

This is just an HDF5 file object; the magic is in the additional methods and indexing that are provided by the TOKIO Time Series-specific HDF5 object.

Parameters `ignore_version` (*bool*) – If true, do not throw `KeyError` if the HDF5 file does not contain a valid version.

`_get_columns_h5lmt` (*dataset_name*)
Get the column names of an h5lmt dataset

`_get_missing_h5lmt` (*dataset_name, inverse=False*)
Return the `FSMissingGroup` dataset from an H5LMT file

Encodes a hot mess of hacks to return something that looks like what `get_missing()` would return for a real dataset.

Parameters

- `dataset_name` (*str*) – name of dataset to access
- `inverse` (*bool*) – return 0 for missing and 1 for present if True

Returns Array of `numpy.int8` of 1 and 0 to indicate the presence or absence of specific elements

Return type `numpy.ndarray`

`_resolve_schema_key` (*key*)
Given a key, either return a key that can be used to index self directly, or return a provider function and arguments to generate the dataset dynamically

`_to_dataframe` (*dataset_name*)
Convert a dataset into a dataframe via TOKIO HDF5 schema

`_to_dataframe_h5lmt` (*dataset_name*)
Convert a dataset into a dataframe via H5LMT native schema

`commit_timeseries` (*timeseries, **kwargs*)
Writes contents of a `TimeSeries` object into a group

Parameters

- `timeseries` (`tokio.timeseries.TimeSeries`) – the time series to save as a dataset within self
- `kwargs` (*dict*) – Extra arguments to pass to `self.create_dataset()`

`get_columns` (*dataset_name*)
Get the column names of a dataset

Parameters `dataset_name` (*str*) – name of dataset whose columns will be retrieved

Returns Array of column names, or empty if no columns defined

Return type `numpy.ndarray`

`get_index` (*dataset_name, target_datetime*)
Turn a datetime object into an integer that can be used to reference specific times in datasets.

`get_missing` (*dataset_name, inverse=False*)
Convert a dataset into a matrix indicating the absence of data

Parameters

- **dataset_name** (*str*) – name of dataset to access
- **inverse** (*bool*) – return 0 for missing and 1 for present if True

Returns Array of `numpy.int8` of 1 and 0 to indicate the presence or absence of specific elements

Return type `numpy.ndarray`

get_timestamps (*dataset_name*)

Return timestamps dataset corresponding to given dataset name

This method returns a dataset, not a numpy array, so you can face severe performance penalties trying to iterate directly on the return value! To iterate over timestamps, it is almost always better to dereference the dataset to get a numpy array and iterate over that in memory.

Parameters **dataset_name** (*str*) – Logical name of dataset whose timestamps should be retrieved

Returns The dataset containing the timestamps corresponding to `dataset_name`.

Return type `h5py.Dataset`

get_timestep (*dataset_name*, *timestamps=None*)

Cache or calculate the timestep for a dataset

get_version (*dataset_name=None*)

Get the version attribute from an HDF5 file dataset

Parameters **dataset_name** (*str*) – Name of dataset to retrieve version. If None, return the global file's version.

Returns The version string for the specified dataset

Return type `str`

set_version (*version*, *dataset_name=None*)

Set the version attribute from an HDF5 file dataset

Provide a portable way to set the global schema version or the version of a specific dataset.

Parameters

- **version** (*str*) – The new version to be set
- **dataset_name** (*str*) – Name of dataset to set version. If None, set the global file's version.

to_dataframe (*dataset_name*)

Convert a dataset into a dataframe

Parameters **dataset_name** (*str*) – dataset name to convert to DataFrame

Returns DataFrame indexed by datetime objects corresponding to timestamps, columns labeled appropriately, and values from the dataset

Return type `pandas.DataFrame`

to_timeseries (*dataset_name*, *light=False*)

Creates a TimeSeries representation of a dataset

Create a TimeSeries dataset object with the data from an existing HDF5 dataset.

Responsible for setting `timeseries.dataset_name`, `timeseries.columns`, `timeseries.dataset`, `timeseries.dataset_metadata`, `timeseries.group_metadata`, `timeseries.timestamp_key`

Parameters

- **dataset_name** (*str*) – Name of existing dataset in self to convert into a Time-Series object
- **light** (*bool*) – If True, don't actually load datasets into memory; reference them directly into the HDF5 file

Returns The in-memory representation of the given dataset.

Return type *tokio.timeseries.TimeSeries*

`tokio.connectors.hdf5.get_insert_indices(my_timestamps, existing_timestamps)`

Given new timestamps and an existing series of timestamps, find the indices overlap so that new data can be inserted into the middle of an existing dataset

`tokio.connectors.hdf5.missing_values(dataset, inverse=False)`

Identify matrix values that are missing

Because we initialize datasets with -0.0, we can scan the sign bit of every element of an array to determine how many data were never populated. This converts negative zeros to ones and all other data into zeros then count up the number of missing elements in the array.

Parameters

- **dataset** – dataset to access
- **inverse** (*bool*) – return 0 for missing and 1 for present if True

Returns Array of numpy.int8 of 1 and 0 to indicate the presence or absence of specific elements

Return type *numpy.ndarray*

tokio.connectors.hpss module

Connect to various outputs made available by HPSS

class `tokio.connectors.hpss.FtpLog(*args, **kwargs)`

Bases: *tokio.connectors.common.SubprocessOutputDict*

Provides an interface for log files containing HPSS FTP transactions

This connector parses FTP logs generated by HPSS 7.3. Older versions are not supported.

HPSS FTP log files contain transfer records that look something like:

```
#0  1  2  3      4  5      6      13  14      15  16      _
↪7      8      9 10 11      12      13  14      15  16      _
Mon Dec 31 00:06:46 2018 dtn01-int.nersc.gov /home/o/operator/.check_ftp.25651 b_
↪      POPN_Cmd      r r ftp      operator fd 0
Mon Dec 31 00:06:46 2018 0.010      dtn01-int.nersc.gov      _
↪33      /home/o/opera... b o PRTR_Cmd r      ftp operator fd 0
Mon Dec 31 00:06:48 2018 0.430      sgn-pub-01.nersc.gov      0_
↪      /home/g/glock... b o RETR_Cmd r      ftp wwwhpss
Mon Feb 4 16:45:04 2019 457.800      sgn-pub-01.nersc.gov      _
↪7184842752 /home/g/glock... b o RETR_Cmd r      ftp wwwhpss
Fri Jul 12 15:32:43 2019 2.080      gert01-224.nersc.gov      _
↪2147483647 /home/n/nickb... b i PSTO_Cmd r      ftp nickb      fd 0
Mon Jul 29 15:44:22 2019 0.800      dtn02.nersc.gov      _
↪464566784 /home/n/nickb... b o PRTR_Cmd r      ftp nickb      fd 0
```

which this class deserializes and represents as a dictionary-like object of the form:

```

{
  "ftp": [
    {
      "bytes": 0,
      "bytes_sec": 0.0,
      "duration_sec": 0.43,
      "end_timestamp": 1546243608.0,
      "hpss_path": "/home/g/glock...",
      "hpss_uid": "wwwhpss",
      "opname": "HL",
      "remote_host": "sgn-pub-01.nersc.gov",
      "start_timestamp": 1546243607.57
    },
    ...
  ],
  "pftp": [
    {
      "bytes": 33,
      "bytes_sec": 3300.0,
      "duration_sec": 0.01,
      "end_timestamp": 1546243606.0,
      "hpss_path": "/home/o/opera...",
      "hpss_uid": "operator",
      "opname": "HL",
      "remote_host": "dtn01-int.nersc.gov",
      "start_timestamp": 1546243605.99
    },
    ...
  ]
}

```

where the top-level keys are either “ftp” or “pftp”, and their values are lists containing every FTP or parallel FTP transaction, respectively.

classmethod from_file (*cache_file*)

Instantiate from a cache file

classmethod from_str (*input_str*)

Instantiate from a string

load_str (*input_str*)

Parse text from an HPSS FTP log

class `tokio.connectors.hpss.HpssDailyReport` (**args*, ***kwargs*)

Bases: `tokio.connectors.common.SubprocessOutputDict`

Representation for the daily report that HPSS can generate

classmethod from_file (*cache_file*)

Instantiate from a cache file

classmethod from_str (*input_str*)

Instantiate from a string

load_str (*input_str*)

Parse the HPSS daily report text

class `tokio.connectors.hpss.HsiLog` (**args*, ***kwargs*)

Bases: `tokio.connectors.common.SubprocessOutputDict`

Provides an interface for log files containing HSI and HTAR transactions

This connector receives input from an HSI log file which takes the form:

```
Sat Aug 10 00:05:26 2019 dtn01.nersc.gov      hsi  57074 31117 LH      0 0.02
↪    543608 12356.7 4 /global/project/projectdir... /home/g/glock/... 57074
Sat Aug 10 00:05:28 2019 cori02-224.nersc.gov htar 58888 14301 create LH 0
↪58178668032 397.20 146472.0 /nersc/projects/blah.tar      5
↪58888
Sat Aug 10 00:05:29 2019 myuniversity.edu    hsi  35136 1391  LH      -1 0.03
↪    0      0.0      0      xyz.bin      /home/g/
↪glock/xyz.bin 35136
```

but uses both tabs and spaces to denote different fields. This connector then presents this data in a dictionary-like form:

```
{
  "hsi": [
    {
      "access_latency_sec": 0.03,
      "account_id": 35136,
      "bytes": 0,
      "bytes_sec": 0.0,
      "client_pid": 1035,
      "cos_id": 0,
      "dest_path": "/home/g/glock/blah.bin",
      "hpss_uid": 35136,
      "opname": "LH",
      "remote_host": "someuniv.edu",
      "return_code": -1,
      "source_path": "blah.bin",
      "end_timestamp": 1565420701
    },
    ...
  ],
  "htar": [
    {
      "account_id": 58888,
      "bytes": 58178668032,
      "bytes_sec": 146472.0,
      "client_pid": 14301,
      "cos_id": 5,
      "duration_sec": 397.2,
      "hpss_path": "/nersc/projects/blah.tar",
      "hpss_uid": 58888,
      "htar_op": "create",
      "opname": "LH",
      "remote_ftp_host": "",
      "remote_host": "cori02-224.nersc.gov",
      "return_code": 0,
      "end_timestamp": 1565420728
    }
  ]
}
```

where the top-level keys are either “hsi” or “htar”, and their values are lists containing every HSI or HTAR transaction, respectively.

The keys generally follow the raw nomenclature used in the HSI logs which can be found on [Mike Gleicher’s website](#). Perhaps most relevant are the opnames, which can be one of

- FU - file unlink. Has no destination filename field or account id.

- FR - file rename. Has no account id.
- LH - transfer into HPSS (“Local to HPSS”)
- HL - transfer out of HPSS (“HPSS to Local”)
- HH - internal file copy (“HPSS-to-HPSS”)

For posterity,

- `access_latency_sec` is the time to open the file. This includes the latency to pull the tape and insert it into the drive.
- `bytes` and `bytes_sec` are the size and rate of data transfer
- `duration_sec` is the time to complete the transfer
- `return_code` is zero on success, nonzero otherwise

classmethod `from_file` (*cache_file*)

Instantiate from a cache file

classmethod `from_str` (*input_str*)

Instantiate from a string

load_str (*input_str*)

Parse an HSI log file containing HSI and HTAR transactions

`tokio.connectors.hpss._find_columns` (*line*, *sep*='=', *gap*=' ', *strict*=False)

Determine the column start/end positions for a header line separator

Takes a line separator such as the one denoted below:

Host	Users	IO_GB
heart	53	148740.6

and returns a tuple of (start index, end index) values that can be used to slice table rows into column entries.

Parameters

- **line** (*str*) – Text comprised of separator characters and spaces that define the extents of columns
- **sep** (*str*) – The character used to draw the column lines
- **gap** (*str*) – The character separating `sep` characters
- **strict** (*bool*) – If true, restrict column extents to only include `sep` characters and not the spaces that follow them.

Returns

Return type list of tuples

`tokio.connectors.hpss._get_ascii_resolution` (*numeric_str*)

Determines the maximum resolution of an ascii-encoded numerical value

Necessary because HPSS logs contain numeric values at different and often-insufficient resolutions. For example, tiny but finite transfers can show up as taking 0.000 seconds, which results in infinitely fast transfers when calculated naively. This function gives us a means to guess at what the real speed might’ve been.

Does not work with scientific notation.

Parameters `numeric_str` (*str*) – An ascii-encoded integer or float

Returns The smallest number that can be expressed using the resolution provided with `numeric_str`

Return type `float`

`tokio.connectors.hpss._hpss_timedelta_to_secs` (*timedelta_str*)

Convert HPSS-encoded timedelta string into seconds

Parameters `timedelta_str` (*str*) – String in form d-HH:MM:SS where d is the number of days, HH is hours, MM minutes, and SS seconds

Returns number of seconds represented by `timedelta_str`

Return type `int`

`tokio.connectors.hpss._parse_section` (*lines*, *start_line=0*)

Parse a single table of the HPSS daily report

Converts a table from the HPSS daily report into a dictionary. For example an example table may appear as:

Host	Users	IO_GB	Ops
heart	53	148740.6	27991
dtn11	5	29538.6	1694
Total	58	178279.2	29685
HPSS ACCOUNTING:		224962.6	

which will return a dict of form:

```
{
  "system": "archive",
  "title": "io totals by hpss client gateway (ui) host",
  "records": {
    "heart": {
      "io_gb": "148740.6",
      "ops": "27991",
      "users": "53",
    },
    "dtn11": {
      "io_gb": "29538.6",
      "ops": "1694",
      "users": "5",
    },
    "total": {
      "io_gb": "178279.2",
      "ops": "29685",
      "users": "58",
    }
  }
}
```

This function is robust to invalid data, and any lines that do not appear to be a valid table will be treated as the end of the table.

Parameters

- **lines** (*list of str*) – Text of the HPSS report
- **start_line** (*int*) – Index of lines defined such that
 - `lines[start_line]` is the table title

- `lines[start_line + 1]` is the table heading row
- `lines[start_line + 2]` is the line separating the table heading and the first row of data
- `lines[start_line + 3:]` are the rows of the table

Returns

Tuple of (dict, int) where

- dict contains the parsed contents of the table
- int is the index of the last line of the table + 1

Return type tuple

`tokio.connectors.hpss._rekey_table(table, key)`

Converts a list of records into a dict of records

Converts a table of records as returned by `_parse_section()` of the form:

```
{
  "records": [
    {
      "host": "heart",
      "io_gb": "148740.6",
      "ops": "27991",
      "users": "53",
    },
    ...
  ]
}
```

Into a table of key-value pairs the form:

```
{
  "records": {
    "heart": {
      "io_gb": "148740.6",
      "ops": "27991",
      "users": "53",
    },
    ...
  }
}
```

Does not handle degenerate keys when re-keying, so only some tables with a uniquely identifying key can be rekeyed.

Parameters

- **table** (*dict*) – Output of the `_parse_section()` function
- **key** (*str*) – Key to pull out of each element of table[‘records’] to use as the key for each record

Returns Table with records expressed as key-value pairs instead of a list

Return type dict

tokio.connectors.lfshealth module

Connectors for the Lustre *lfs df* and *lctl dl -t* commands to determine the health of Lustre file systems from the clients' perspective.

class tokio.connectors.lfshealth.LfsOstFullness (*args, **kwargs)

Bases: *tokio.connectors.common.SubprocessOutputDict*

Representation for the *lfs df* command. Generates a dict of form

```
{ file_system: { ost_name : { keys: values } } }
```

__repr__ ()

Serialize object into an ASCII string

Returns a string that resembles the input used to initialize this object:

```
snx11025-OST0001_UUID 90767651352 63381521692 26424604184 71% /scratch1[OST:1]
```

load_str (input_str)

Parse the output of *lfs df* to initialize self

class tokio.connectors.lfshealth.LfsOstMap (*args, **kwargs)

Bases: *tokio.connectors.common.SubprocessOutputDict*

Representation for the *lctl dl -t* command. Generates a dict of form

```
{ file_system: { ost_name : { keys: values } } }
```

This is a generally logical structure, although this map is always almost fed into a routine that tries to find multiple OSTs on the same OSS (i.e., a failover situation)

__repr__ ()

Serialize object into an ASCII string

Returns a string that resembles the input used to initialize this object

get_failovers ()

Identify OSSes with an abnormal number of OSTs

Identify OSTs that are probably failed over and return a list of abnormal OSSes and the expected number of OSTs per OSS.

load_str (input_str)

Parse the output of *lctl dl -t* to initialize self

tokio.connectors.lmtdb module

Interface with an LMT database. Provides wrappers for common queries using the CachingDb class.

class tokio.connectors.lmtdb.LmtDb (dbhost=None, dbuser=None, dbpassword=None, dbname=None, cache_file=None)

Bases: *tokio.connectors.cachingdb.CachingDb*

Class to wrap the connection to an LMT MySQL database or SQLite database

__init__ (dbhost=None, dbuser=None, dbpassword=None, dbname=None, cache_file=None)

Initialize LmtDb with either a MySQL or SQLite backend

get_mds_data (datetime_start, datetime_end, timechunk=datetime.timedelta(0, 3600))

Schema-agnostic method for retrieving MDS load data.

Wraps get_timeseries_data() but fills in the exact table name used in the LMT database schema.

Parameters

- **datetime_start** (*datetime.datetime*) – lower bound on time series data to retrieve, inclusive
- **datetime_End** (*datetime.datetime*) – upper bound on time series data to retrieve, exclusive
- **timechunk** (*datetime.timedelta*) – divide time range query into sub-ranges of this width to work around N*N scaling of JOINS

Returns Tuple of (results, column names) where results are tuples of tuples as returned by the MySQL query and column names are the names of each column expressed in the individual rows of results.

get_mds_ops_data (*datetime_start*, *datetime_end*, *timechunk=datetime.timedelta(0, 3600)*)

Schema-agnostic method for retrieving metadata operations data.

Wraps `get_timeseries_data()` but fills in the exact table name used in the LMT database schema.

Parameters

- **datetime_start** (*datetime.datetime*) – lower bound on time series data to retrieve, inclusive
- **datetime_End** (*datetime.datetime*) – upper bound on time series data to retrieve, exclusive
- **timechunk** (*datetime.timedelta*) – divide time range query into sub-ranges of this width to work around N*N scaling of JOINS

Returns Tuple of (results, column names) where results are tuples of tuples as returned by the MySQL query and column names are the names of each column expressed in the individual rows of results.

get_oss_data (*datetime_start*, *datetime_end*, *timechunk=datetime.timedelta(0, 3600)*)

Schema-agnostic method for retrieving OSS data.

Wraps `get_timeseries_data()` but fills in the exact table name used in the LMT database schema.

Parameters

- **datetime_start** (*datetime.datetime*) – lower bound on time series data to retrieve, inclusive
- **datetime_End** (*datetime.datetime*) – upper bound on time series data to retrieve, exclusive
- **timechunk** (*datetime.timedelta*) – divide time range query into sub-ranges of this width to work around N*N scaling of JOINS

Returns Tuple of (results, column names) where results are tuples of tuples as returned by the MySQL query and column names are the names of each column expressed in the individual rows of results.

get_ost_data (*datetime_start*, *datetime_end*, *timechunk=datetime.timedelta(0, 3600)*)

Schema-agnostic method for retrieving OST data.

Wraps `get_timeseries_data()` but fills in the exact table name used in the LMT database schema.

Parameters

- **datetime_start** (*datetime.datetime*) – lower bound on time series data to retrieve, inclusive

- **datetime_End** (*datetime.datetime*) – upper bound on time series data to retrieve, exclusive
- **timechunk** (*datetime.timedelta*) – divide time range query into sub-ranges of this width to work around N*N scaling of JOINS

Returns Tuple of (results, column names) where results are tuples of tuples as returned by the MySQL query and column names are the names of each column expressed in the individual rows of results.

get_timeseries_data (*table, datetime_start, datetime_end, timechunk=datetime.timedelta(0, 3600)*)

Break a timeseries query into smaller queries over smaller time ranges. This is an optimization to avoid the O(N*M) scaling of the JOINS in the underlying SQL query.

get_ts_ids (*datetime_start, datetime_end*)

Given a starting and ending time, return the lowest and highest ts_id values that encompass those dates, inclusive.

tokio.connectors.mmperfmon module

Connectors for the GPFS mmperfmon query usage and mmperfmon query gpfsNumberOperations.

The typical output of mmperfmon query usage may look something like:

```
Legend:
1: xxxxxxxx.nersc.gov|CPU|cpu_user
2: xxxxxxxx.nersc.gov|CPU|cpu_sys
3: xxxxxxxx.nersc.gov|Memory|mem_total
4: xxxxxxxx.nersc.gov|Memory|mem_free
5: xxxxxxxx.nersc.gov|Network|lo|net_r
6: xxxxxxxx.nersc.gov|Network|lo|net_s
```

Row	Timestamp	cpu_user	cpu_sys	mem_total	mem_free	net_r	net_s
1	2019-01-11-10:00:00	0.2	0.56	31371.0 MB	18786.5 MB	1.7 kB	1.7 kB
2	2019-01-11-10:01:00	0.22	0.57	31371.0 MB	18785.6 MB	1.7 kB	1.7 kB
3	2019-01-11-10:02:00	0.14	0.55	31371.0 MB	18785.1 MB	1.7 kB	1.7 kB

Whereas the typical output of mmperfmon query gpfsnsdds is:

```
Legend:
1: xxxxxxxx.nersc.gov|GPFSNSDDisk|na07md01|gpfs_nsdds_bytes_read
2: xxxxxxxx.nersc.gov|GPFSNSDDisk|na07md02|gpfs_nsdds_bytes_read
3: xxxxxxxx.nersc.gov|GPFSNSDDisk|na07md03|gpfs_nsdds_bytes_read
```

Row	Timestamp	gpfs_nsdds_bytes_read	gpfs_nsdds_bytes_read	gpfs_nsdds_bytes_
↔ read				
1	2019-03-04-16:01:00	203539391	0	↔
↔ 0				
2	2019-03-04-16:02:00	175109739	0	↔
↔ 0				
3	2019-03-04-16:03:00	57053762	0	↔
↔ 0				

In general, each Legend: entry has the format:

```
col_number: hostname|subsystem[|device_id]|counter_name
```

where

- `col_number` is an arbitrary number
- `hostname` is the fully qualified NSD server hostname
- `subsystem` is the type of component being measured (CPU, memory, network, disk)
- `device_id` is optional and represents the instance of the subsystem being measured (e.g., CPU core ID, network interface, or disk identifier)
- `counter_name` is the specific metric being measured

It is also worth noting that `mmperfmon` treats a timestamp labeled as, for example, `2019-03-04-16:01:00` as containing all data from the period between `2019-03-04-16:00:00` and `2019-03-04-16:01:00`.

class `tokio.connectors.mmperfmon.Mmperfmon` (**args, **kwargs*)

Bases: `tokio.connectors.common.SubprocessOutputDict`

Representation for the `mmperfmon` query command. Generates a dict of form:

```
{
  timestamp0: {
    "something0.nersc.gov": {
      "key0": value0,
      "key1": value1,
      ...
    },
    "something1.nersc.gov": {
      ...
    },
    ...
  },
  timestamp1: {
    ...
  },
  ...
}
```

__repr__ ()

Returns string representation of self

This does not convert back into a format that attempts to resemble the `mmperfmon` output because the process of loading `mmperfmon` output is lossy.

classmethod from_file (*cache_file*)

Instantiate from a cache file

classmethod from_str (*input_str*)

Instantiate from a string

load (*cache_file=None*)

Load either a tarfile, directory, or single `mmperfmon` output file

Tries to load `self.cache_file`; if it is a directory or tarfile, it is handled by `self.load_multiple`; otherwise falls through to the `load_str` code path.

load_cache (*cache_file=None*)

Loads from one of two formats of cache files

Because `self.save_cache()` outputs to a different format from `self.load_str()`, `load_cache()` must be able to ingest both formats.

load_multiple (*input_file*)

Load one or more input files from a directory or tarball

Parameters

- **input_file** (*str*) – Path to either a directory or a tarfile containing
- **text files, each of which contains the output of a single** (*multiple*) –
- **invocation.** (*mmpperfmon*) –

load_str (*input_str*)

Parses the output of the subprocess output to initialize self

Parameters **input_str** (*str*) – Text output of the `mmpperfmon query` command

to_dataframe (*by_host=None, by_metric=None*)

Convert to a `pandas.DataFrame`

to_dataframe_by_host (*host*)

Returns data from a specific host as a `DataFrame`

Parameters **host** (*str*) – Hostname from which a `DataFrame` should be constructed

Returns All measurements from the given host. Columns correspond to different metrics; indexed in time.

Return type `pandas.DataFrame`

to_dataframe_by_metric (*metric*)

Returns data for a specific metric as a `DataFrame`

Parameters **metric** (*str*) – Metric from which a `DataFrame` should be constructed

Returns All measurements of the given metric for all hosts. Columns represent hosts; indexed in time.

Return type `pandas.DataFrame`

to_json (***kwargs*)

Returns a json-encoded string representation of self.

Returns JSON representation of self

Return type `str`

`tokio.connectors.mmpperfmon.get_col_pos` (*line, align=None*)

Return column offsets of a left-aligned text table

For example, given the string:

```
Row          Timestamp cpu_user  cpu_sys   mem_total
123456789x123456789x123456789x123456789x123456789x123456789x
```

would return:

```
[(0, 4), (15, 24), (25, 33), (34, 41), (44, 53)]
```

for `align=None`.

Parameters

- **line** (*str*) – String from which offsets should be determined

- **align** (*str or None*) – Expected column alignment; one of ‘left’, ‘right’, or None (to return the exact start and stop of just the non-space text)

Returns List of tuples of integer offsets denoting the start index (inclusive) and stop index (exclusive) for each column.

Return type *list*

`tokio.connectors.mmperfmon.value_unit_to_bytes` (*value_unit*)

Converts a value+unit string into bytes

Converts a string containing both a numerical value and a unit of that value into a normalized value. For example, “1 MB” will convert to 1048576.

Parameters **value_unit** (*str*) – Of the format “float str” where float is the value and str is the unit by which value is expressed.

Returns Number of bytes represented by value_unit

Return type *int*

tokio.connectors.nersc_globuslogs module

Retrieve Globus transfer logs from NERSC’s Elasticsearch infrastructure

Connects to NERSC’s OMNI service and retrieves Globus transfer logs.

class `tokio.connectors.nersc_globuslogs.NerscGlobusLogs` (**args, **kwargs*)

Bases: `tokio.connectors.es.EsConnection`

Connection handler for NERSC Globus transfer logs

classmethod `from_cache` (**args, **kwargs*)

Initializes an EsConnection object from a cache file.

This path is designed to be used for testing.

Parameters **cache_file** (*str*) – Path to the JSON formatted list of pages

query (*start, end, must=None, scroll=True*)

Queries Elasticsearch for Globus logs

Accepts a start time, end time, and an optional “must” field which can be used to apply additional term queries. For example, `must` may be:

```
[
  {
    "term": {
      "TASKID": "none"
    }
  },
  {
    "term": {
      "TYPE": "STOR"
    }
  }
]
```

which would return only those queries that have no associated TASKID and were sending (storing) data.

Parameters

- **start** (*datetime.datetime*) – lower bound for query (inclusive)

- **end** (*datetime.datetime*) – upper bound for query (exclusive)
- **must** (*list or None*) – list of dictionaries to be inserted as additional term-level query parameters.
- **scroll** (*bool*) – Use the scrolling interface if True. If False, `source_filter/filter_function/flush_every/flush_function` are ignored.

query_timeseries (*query_template, start, end, scroll=True*)

Craft and issue query that returns all overlapping records

Parameters

- **query_template** (*dict*) – a query object containing at least one `@timestamp` field
- **start** (*datetime.datetime*) – lower bound for query (inclusive)
- **end** (*datetime.datetime*) – upper bound for query (exclusive)
- **scroll** (*bool*) – Use the scrolling interface if True. If False, `source_filter/filter_function/flush_every/flush_function` are ignored.

query_type (*start, end, xfer_type*)

Wraps `query()` with a type restriction

Convenience method to constrain a query to a specific transfer type.

Parameters

- **start** (*datetime.datetime*) – lower bound for query (inclusive)
- **end** (*datetime.datetime*) – upper bound for query (exclusive)
- **xfer_type** (*str*) – constrain results to this transfer type (STOR, RETR, etc). Case sensitive.

query_user (*start, end, user*)

Wraps `query()` with a user restriction

Convenience method to constrain a query to a specific user.

Parameters

- **start** (*datetime.datetime*) – lower bound for query (inclusive)
- **end** (*datetime.datetime*) – upper bound for query (exclusive)
- **user** (*str*) – constrain results to this username

to_dataframe ()

Converts `self.scroll_pages` to a DataFrame

Returns Contents of the last query's pages

Return type `pandas.DataFrame`

tokio.connectors.nersc_isdct module

Connect to NERSC's Intel Data Center Tool for SSDs outputs

Processes and aggregates the output of Intel Data Center Tool for SSDs outputs in the format generated by NERSC's daily script. The NERSC infrastructure runs ISDCT in a verbose way on every burst buffer node, then collects all the resulting output text files from a node into a directory bearing that node's nid (e.g., `nid01234/*.txt`). There is

also an optional timestamp file contained in the toplevel directory. Also processes .tar.gz versions of these collected metrics.

class `tokio.connectors.nersc_isdct.NerscIsdct` (*input_file*)

Bases: `tokio.connectors.common.CacheableDict`

Dictionary subclass that self-populates with ISDCT output data

__init__ (*input_file*)

Load the output of a NERSC ISDCT dump.

Parameters `input_file` (*str*) – Path to either a directory or a tar(/gzipped) directory containing the output of NERSC’s ISDCT collection script.

_synthesize_metrics ()

Calculate additional metrics not presented by ISDCT.

Calculates additional convenient metrics that are not directly presented by ISDCT, then adds the resulting key-value pairs to self.

diff (*old_isdct*, *report_zeros=True*)

Highlight differences between self and another NerscIsdct.

Subtract each counter for each serial number in this object from its counterpart in `old_isdct`. Return the changes in each numeric counter and any serial numbers that have appeared or disappeared.

Parameters

- `old_isdct` (`NerscIsdct`) – object with which we should be compared
- `report_zeros` (*bool*) – If True, report all counters even if they showed no change. Default is True.

Returns

Dictionary containing the following keys:

- `added_devices` - device serial numbers which exist in self but not `old_isdct`
- `removed_devices` - device serial numbers which do not exist in self but do in `old_isdct`
- `devices` - dict keyed by device serial numbers and whose values are dicts of keys whose values are the difference between `old_isdct` and self

Return type `dict`

load ()

Wrapper around the filetype-specific loader.

Infer the type of input being given, dispatch the correct loading function, and populate keys/values.

load_native ()

Load ISDCT output from a tar(.gz).

Load a collection of ISDCT outputs as created by the NERSC ISDCT script. Assume that ISDCT output files each contain a single output from a single invocation of the `isdct` tool, and outputs are grouped into directories named according to their `nid` numbers (e.g., `nid00984/somefile.txt`).

to_dataframe (*only_numeric=False*)

Express self as a dataframe.

Parameters `only_numeric` (*bool*) – Only output columns containing numeric data of True; otherwise, output all columns.

Returns Dataframe indexed by serial number and with ISDCT counters as columns

Return type `pandas.DataFrame`

`tokio.connectors.nerisc_isdct._decode_nerisc_nid(path)`

Convert path to ISDCT output into a nid.

Given a path to some ISDCT output file, somehow figure out what the nid name for that node is. This encoding is specific to the way NERSC collects and preserves ISDCT outputs.

Parameters `path (str)` – path to an ISDCT output text file

Returns Node identifier (e.g., nid01234)

Return type `str`

`tokio.connectors.nerisc_isdct._merge_parsed_counters(parsed_counters_list)`

Merge ISDCT outputs into a single object.

Aggregates counters from each record based on the NVMe device serial number, with redundant counters being overwritten.

Parameters `parsed_counters_list (list)` – List of parsed ISDCT outputs as dicts. Each list element is a dict with a single key (a device serial number) and one or more values; each value is itself a dict of key-value pairs corresponding to ISDCT/SMART counters from that device.

Returns Dict with keys given by all device serial numbers found in `parsed_counters_list` and whose values are a dict containing keys and values representing all unique keys across all elements of `parsed_counters_list`.

Return type `dict`

`tokio.connectors.nerisc_isdct._normalize_key(key)`

Coerce all keys into a similar naming convention.

Converts Intel's mix of camel-case and snake-case counters into all snake-case. Contains some nasty acronym hacks that may require modification if/when Intel adds new funny acronyms that contain a mix of upper and lower case letters (e.g., SMBus and NVMe).

Parameters `key (str)` – a string to normalize

Returns `snake_case` version of key

Return type `str`

`tokio.connectors.nerisc_isdct._rekey_smart_buffer(smart_buffer)`

Convert SMART values associated with one register into unique counters.

Take a buffer containing smart values associated with one register and create unique counters. Only necessary for older versions of ISDCT which did not output SMART registers in a standard “Key: value” text format.

Parameters `smart_buffer (dict)` – SMART buffer as defined by `parse_counters_fileobj()`

Returns unique key:value pairs whose key now includes distinguishing device-specific characteristics to avoid collision from other devices that generated SMART data

Return type `dict`

`tokio.connectors.nerisc_isdct.parse_counters_fileobj(fileobj, nodename=None)`

Convert any output of ISDCT into key-value pairs.

Reads the output of a file-like object which contains the output of a single `isdct` command. Understands the output of the following options:

- `isdct show -smart` (SMART attributes)
- `isdct show -sensor` (device health sensors)

- `isdct show -performance` (device performance metrics)
- `isdct show -a` (drive info)

Parameters

- **fileobj** (*file*) – file-like object containing the output of an ISDCT command
- **nodename** (*str*) – name of node corresponding to *fileobj*, if known

Returns dict of dicts keyed by the device serial number.

Return type dict

tokio.connectors.nersc_jobsdb module

Extract job info from the NERSC jobs database. Accessing the MySQL database is not required (i.e., if you have everything stored in a cache, you never have to touch MySQL). However if you do have to connect to MySQL, you must set the following environment variables:

```

NERSC_JOBSDB_HOST
NERSC_JOBSDB_USER
NERSC_JOBSDB_PASSWORD
NERSC_JOBSDB_DB
    
```

If you do not know what to use as these credentials, you will have to rely on a cache database.

class `tokio.connectors.nersc_jobsdb.NerscJobsDb` (*dbhost=None, dbuser=None, dbpassword=None, dbname=None, cache_file=None*)

Bases: `tokio.connectors.cachingdb.CachingDb`

Connect to and interact with the NERSC jobs database. Maintains a query cache where the results of queries are cached in memory. If a query is repeated, its values are simply regurgitated from here rather than touching any databases.

If this class is instantiated with a `cache_file` argument, all queries will go to that SQLite-based cache database if they are not found in the in-memory cache.

If this class is not instantiated with a `cache_file` argument, all queries that do not exist in the in-memory cache will go out to the MySQL database.

The in-memory query caching is possible because the job data in the NERSC jobs database is immutable and can be cached indefinitely once it appears there. At any time the memory cache can be committed to a cache database to be used or transported later.

drop_cache ()
Clear the query cache

get_concurrent_jobs (*start_timestamp, end_timestamp, nersc_host*)
Grab all of the jobs that were running, in part or in full, during the time window bounded by `start_timestamp` and `end_timestamp`. Then calculate the fraction overlap for each job to calculate the number of core hours that were burned overall during the start/end time of interest.

get_job_startend (*jobid, nersc_host*)
Return start and end time for a given job id
Retrieves the time a job started and completed.

Parameters

- **jobid** (*str*) – Job ID of interest

- **nersc_host** (*str*) – NERSC host to which job ID of interest maps

Returns Two-item tuple of (start time, end time)

Return type tuple of datetime.datetime

query (*query_str*, *query_variables=()*, *nocache=False*)

Pass a query through all layers of cache and return on the first hit.

tokio.connectors.nersc_lfsstate module

Tools to parse and index the outputs of Lustre's `lfs` and `lctl` commands to quantify Lustre fullness and health. Assumes inputs are generated by NERSC's Lustre health monitoring cron jobs which periodically issue the following:

```
echo "BEGIN $(date +%s)" >> osts.txt
/usr/bin/lfs df >> osts.txt

echo "BEGIN $(date +%s)" >> ost-map.txt
/usr/sbin/lctl dl -t >> ost-map.txt
```

Accepts ASCII text files, or gzip-compressed text files.

class `tokio.connectors.nersc_lfsstate.NerscLfsOstFullness` (*cache_file=None*)

Bases: `dict`

Subclass of dictionary that self-populates with Lustre OST fullness.

__init__ (*cache_file=None*)

Load the fullness of OSTs

Parameters **cache_file** (*str*, *optional*) – Path to a cache file to load instead of issuing the `lfs df` command

__repr__ ()

Serialize OST fullness into a format that resembles `lfs df`.

Returns

Serialization of the OST fullness in a format similar to `lfs df`. Columns are

- Name of OST (e.g., `snx11025-OST0001_UUID`)
- Total kibibytes on OST
- Used kibibytes on OST
- Available kibibytes on OST
- Percent capacity used
- Mount point, role, and OST ID

Return type `str`

_save_cache (*output*)

Serialize object into a form resembling the output of `lfs df`.

Parameters **output** (*file*) – File-like object into which resulting text should be written.

load_ost_fullness_file ()

Parse the cached output of OST fullness generated by `lfs df`.

Parses the output of a file containing concatenated outputs of `lfs df` separated by lines of the form `BEGIN 0000` where 0000 is the UNIX epoch time.

save_cache (*output_file=None*)

Serialize object into a form resembling the output of `lfs df`.

Parameters **output_file** (*str*) – Path to a file to which the serialized output should be written. If None, print to stdout.

class `tokio.connectors.nersc_lfsstate.NerscLfsOstMap` (*cache_file=None*)

Bases: `dict`

Subclass of dictionary that self-populates with Lustre OST-OSS mapping.

__init__ (*cache_file=None*)

Load the mapping of OSTs to OSSes.

Parameters **cache_file** (*str, optional*) – Path to a cache file to load instead of issuing the `lctl dl -t` command

__repr__ ()

Serialize OST map into a format that resembles `lctl dl -t`.

Returns

Serialization of the OST to OSS mapping in a format similar to `lctl dl -t`. Fixed-width columns are

- index: OST/MDT index
- status: up/down status
- role: `osc`, `mdc`, etc
- role_id: name with unique identifier for target
- uuid: UUID of target
- ref_count: number of references to target
- nid: LNET identifier of the target

Return type `str`

__save_cache (*output*)

Serialize object into a form resembling the output of `lctl dl -t`.

Parameters **output** (*file*) – File-like object into which resulting text should be written.

get_failovers ()

Determine OSSes which are likely affected by a failover.

Figure out the OSTs that are probably failed over and, for each time stamp and file system, return a list of abnormal OSSes and the expected number of OSTs per OSS.

Returns

Dictionary keyed by timestamps and whose values are dicts of the form:

```
{
  'mode': int,
  'abnormal_ips': [list of str]
}
```

where `mode` refers to the statistical mode of OSTs per OSS, and `abnormal_ips` is a list of strings containing the IP addresses of OSSes whose OST counts are not equal to the mode for that time stamp.

Return type `dict`

load_ost_map_file()

Parse the cached output of an OST map generated by `lctl dl -t`.

Reads the input OST map as given by the `cache_file` attribute and populates self with keys of the form:

```
{ timestamp(int) : { file_system: { ost_name : { keys: values } } } }
```

save_cache (*output_file=None*)

Serialize object into a form resembling the output of `lctl dl -t`.

Parameters `output_file` (*str*) – Path to a file to which the serialized output should be written. If None, print to stdout.

`tokio.connectors.nersc_lfsstate._REX_LFS_DF = <_sre.SRE_Pattern object>`

Regular expression to extract OST fullness levels

Matches output of `lfs df` which takes the form:

```
snx11035-OST0000_UUID 90767651352 54512631228 35277748388 61% /scratch2[OST:0]
```

where the columns are

- OST/MDT UID
- kibibytes total
- kibibytes in use
- kibibytes available
- percent fullness
- file system mount, role, and ID

Carries the implicit assumption that all OSTs are prefixed with `snx`.

`tokio.connectors.nersc_lfsstate._REX_OST_MAP = <_sre.SRE_Pattern object>`

Regular expression to match OSC/MDC lines

Matches output of `lctl dl -t` which takes the form:

```
351 UP osc snx11025-OST0007-osc-ffff8875ac1e7c00 3f30f170-90e6-b332-b141-
↪a6d4a94a1829 5 10.100.100.12@o2ib1
```

Intentionally skips MGC, LOV, and LMV lines.

tokio.connectors.slurm module

Connect to Slurm via Slurm CLI outputs.

This connector provides Python bindings to retrieve information made available through the standard Slurm `saccount` and `scontrol` CLI commands. It is currently very limited in functionality.

class `tokio.connectors.slurm.Slurm` (*jobid=None, *args, **kwargs*)

Bases: `tokio.connectors.common.SubprocessOutputDict`

Dictionary subclass that self-populates with Slurm output data

Presents a schema that is keyed as:

```
{
  taskid: {
    slurmfield1: value1
    slurmfield2: value2
    ...
  }
}
```

where taskid can be any of

- jobid
- jobid.<step>
- jobid.batch

__init__ (*jobid=None, *args, **kwargs*)
Load basic information from Slurm

Parameters **jobid** (*str*) – Slurm Job ID associated with data this object contains

Variables **jobid** (*str*) – Slurm Job ID associated with data contained in this object

__repr__ ()
Serialize object in the same format as sacct.

Returns Serialized version of self in a similar format as the `sacct` output so that this object can be circularly serialized and deserialized.

Return type `str`

__recast_keys (**target_keys*)
Convert own keys into native Python objects.

Scan self and convert special keys into native Python objects where appropriate. If no keys are given, scan everything. Do NOT attempt to recast anything that is not a string—this is to avoid relying on `expand_nodelist` if a key is already recast since `expand_nodelist` does not function outside of an environment containing Slurm.

Parameters ***target_keys** (*list, optional*) – Only convert these keys into native Python object types. If omitted, convert all keys.

from_json (*json_string*)
Initialize self from a JSON-encoded string.

Parameters **json_string** (*str*) – JSON representation of self

get_job_ids ()
Return the top-level jobid(s) contained in object.

Retrieve the jobid(s) contained in self without any accompanying taskid information.

Returns list of jobid(s) contained in self.

Return type list of str

get_job_nodes ()
Return a list of all job nodes used.

Creates a list of all nodes used across all tasks for the self.jobid. Useful if the object contains only a subset of tasks executed by the Slurm job.

Returns Set of node names used by the job described by this object

Return type `set`

get_job_startend()

Find earliest start and latest end time for a job.

For an entire job and all its tasks, find the absolute earliest start time and absolute latest end time.

Returns Two-item tuple of (earliest start time, latest end time) in whatever type `self['start']` and `self['end']` are stored

Return type `tuple`

load()

Initialize values either from cache or sacct

load_keys(*keys)

Retrieve a list of keys from sacct and insert them into self.

This always invokes sacct and can be used to overwrite the contents of a cache file.

Parameters ***keys** (*list*) – Slurm attributes to include; names should be valid input to `sacct -format` CLI utility.

load_str(input_str)

Load from either a json cache or the output of sacct

to_dataframe()

Convert self into a Pandas DataFrame.

Returns a Pandas DataFrame representation of this object.

Returns DataFrame representation of the same schema as the Slurm `sacct` command.

Return type `pandas.DataFrame`

to_json(kwargs)**

Return a json-encoded string representation of self.

Serializes self to json using `_RECAST_KEY_MAP` to convert Python types back into JSON-compatible types.

Returns JSON representation of self

Return type `str`

```
class tokio.connectors.slurm.SlurmEncoder (skipkeys=False, ensure_ascii=True,
check_circular=True, allow_nan=True,
sort_keys=False, indent=None, separa-
tors=None, encoding='utf-8', default=None)
```

Bases: `json.encoder.JSONEncoder`

Encode sets as lists and datetimes as ISO 8601.

default(o)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
```

(continues on next page)

(continued from previous page)

```
# Let the base class default method raise the TypeError
return JSONEncoder.default(self, o)
```

`tokio.connectors.slurm._RECAST_KEY_MAP = {'end': (<function <lambda>>, <function <lambda>>)`
 Methods to convert Slurm string outputs into Python objects

This table provides the methods to apply to various Slurm output keys to convert them from strings (the default Slurm output type) into more useful Python objects such as datetimes or lists.

- `value[0]` is the function to cast to Python
- `value[1]` is the function to cast back to a string

Type `dict`

`tokio.connectors.slurm.compact_nodelist (node_string)`

Convert a string of nodes into compact representation.

Wraps `scontrol show hostlist nid05032,nid05033,...` to compress a list of nodes to a Slurm nodelist string. This is effectively the reverse of `expand_nodelist()`

Parameters `node_string (str)` – Comma-separated list of node names (e.g., `nid05032, nid05033, ...`)

Returns The compact representation of `node_string` (e.g., `nid0[5032-5159]`)

Return type `str`

`tokio.connectors.slurm.expand_nodelist (node_string)`

Expand Slurm compact nodelist into a set of nodes.

Wraps `scontrol show hostname nid0[5032-5159]` to expand a Slurm nodelist string into a list of nodes.

Parameters `node_string (str)` – Node list in Slurm's compact notation (e.g., `nid0[5032-5159]`)

Returns Set of strings which encode the fully expanded node names contained in `node_string`.

Return type `set`

`tokio.connectors.slurm.jobs_running_between (start, end, keys=None)`

Generate a list of Slurm jobs that ran between a time range

Parameters

- **start** (`datetime.datetime`) – Find jobs that ended at or after this time
- **end** (`datetime.datetime`) – Find jobs that started at or before this time
- **state** (`str`) – Any valid sacct state
- **keys** (`list`) – List of Slurm fields to return for each running job

Returns Slurm object containing jobs whose runtime overlapped with the start and end times

Return type `tokio.connectors.slurm.Slurm`

`tokio.connectors.slurm.parse_sacct (sacct_str)`

Convert output of `sacct -p` into a dictionary.

Parses the output of `sacct -p` and return a dictionary with the full (raw) contents.

Parameters `sacct_str (str)` – stdout of an invocation of `sacct -p`

Returns Keyed by Slurm Job ID and whose values are dicts containing key-value pairs corresponding to the Slurm quantities returned by `sacct -p`.

Return type `dict`

tokio.tools package

A higher-level interface that wraps various connectors and site-dependent configuration to provide convenient abstractions upon which analysis tools can be portably built.

Submodules

tokio.tools.common module

Common routines used to apply site-specific info to connectors

`tokio.tools.common._expand_check_paths` (*template*, *lookup_key*)

Generate paths to examine from a variable-type template.

template may be one of three data structures:

- `str`: search for files matching this exact template
- list of `str`: search for files matching each template listed.
- `dict`: use *lookup_key* to determine the element in the dictionary to use as the template. That value is treated as a new *template* object and can be of any of these three types.

Parameters

- **template** (*str*, *list*, or *dict*) – Template string(s) which should be passed to `datetime.datetime.strptime` to be converted into specific time-delimited files.
- **lookup_key** (*str* or *None*) – When *type(template)* is `dict`, use this key to identify the key-value to use as template. If *None* and *template* is a `dict`, iterate through all values of *template*.

Returns List of strings, each describing a path to an existing HDF5 file that should contain data relevant to the requested start and end dates.

Return type `list`

`tokio.tools.common._match_files` (*check_paths*, *use_time*, *match_first*, *use_glob*)

Locate file(s) that match a templated file path for a given time

Parameters

- **check_paths** (*list of str*) – List of templates to pass to `strptime`
- **use_time** (*datetime.datetime*) – Time to pass through `strptime` to generate an actual file path to check for existence.
- **match_first** (*bool*) – If `True`, only return the first matching file for each time increment checked. Otherwise, return `_all_` matching files.
- **use_glob** (*bool*) – Expand file globs in template

Returns List of strings, each describing a path to an existing HDF5 file that should contain data relevant to the requested start and end dates.

Return type list

```
tokio.tools.common.enumerate_dated_files(start, end, template,  
                                         lookup_key=None, match_first=True,  
                                         timedelta=datetime.timedelta(1),  
                                         use_glob=False)
```

Locate existing time-indexed files between a start and end time.

Given a start time, end time, and template data structure that describes a pattern by which the files of interest are indexed, locate all existing files that fall between the start and end time.

The template argument (*template*) are paths that are passed through *datetime.strptime* and then checked for existence for every *timedelta* increment between *start* and *end*, inclusive. *template* may be one of three data structures:

- str: search for files matching this template
- list of str: search for files matching each template. If *match_first* is True, only the first hit per list item per time interval is returned; otherwise, every file matching every template in the entire list is returned.
- dict: use *lookup_key* to determine the element in the dictionary to use as the template. That value is treated as a new *template* object and can be of any of these three types.

Parameters

- **start** (*datetime.datetime*) – Begin including files corresponding to this start date, inclusive.
- **end** (*datetime.datetime*) – Stop including files with timestamps that follow this end date. Resulting files *_will_* include this date.
- **template** (*str, list, or dict*) – Template string(s) which should be passed to *datetime.strptime* to be converted into specific time-delimited files.
- **lookup_key** (*str or None*) – When *type(template)* is dict, use this key to identify the key-value to use as template. If None and *template* is a dict, iterate through all values of *template*.
- **match_first** (*bool*) – If True, only return the first matching file for each time increment checked. Otherwise, return *_all_* matching files.
- **timedelta** (*datetime.timedelta*) – Increment to use when iterating between *start* and *end* while looking for matching files.
- **use_glob** (*bool*) – Expand file globs in template

Returns List of strings, each describing a path to an existing HDF5 file that should contain data relevant to the requested start and end dates.

Return type list

tokio.tools.darshan module

Tools to find Darshan logs within a system-wide repository

```
tokio.tools.darshan.find_darshanlogs(datetime_start=None, datetime_end=None, user-  
                                     name=None, jobid=None, log_dir=None, sys-  
                                     tem=None)
```

Return darshan log file paths matching a set of criteria

Attempts to find Darshan logs that match the input criteria.

Parameters

- **datetime_start** (*datetime.datetime*) – date to begin looking for Darshan logs
- **datetime_end** (*datetime.datetime*) – date to stop looking for Darshan logs
- **username** (*str*) – username of user who generated the log
- **jobid** (*int*) – jobid corresponding to Darshan log
- **log_dir** (*str*) – path to Darshan log directory base
- **system** (*str or None*) – key to pass to `enumerate_dated_files`'s `lookup_key` when resolving `darshan_log_dir`

Returns paths of matching Darshan logs as strings

Return type `list`

```
tokio.tools.darshan.load_darshanlogs(datetime_start=None, datetime_end=None, username=None, jobid=None, log_dir=None, system=None, which=None, **kwargs)
```

Return parsed Darshan logs matching a set of criteria

Finds Darshan logs that match the input criteria, loads them, and returns a dictionary of `connectors.darshan.Darshan` objects keyed by the full log file paths to the source logs.

Parameters

- **datetime_start** (*datetime.datetime*) – date to begin looking for Darshan logs
- **datetime_end** (*datetime.datetime*) – date to stop looking for Darshan logs
- **username** (*str*) – username of user who generated the log
- **jobid** (*int*) – jobid corresponding to Darshan log
- **log_dir** (*str*) – path to Darshan log directory base
- **system** (*str*) – key to pass to `enumerate_dated_files`'s `lookup_key` when resolving `darshan_log_dir`
- **which** (*str*) – ‘base’, ‘total’, and/or ‘perf’ as a comma-delimited string
- **kwargs** – arguments to pass to the `connectors.darshan.Darshan` object initializer

Returns keyed by log file name whose values are `connectors.darshan.Darshan` objects

Return type `dict`

tokio.tools.hdf5 module

Retrieve data from TOKIO Time Series files using time as inputs

Provides a mapping between dates and times and a site's time-indexed repository of TOKIO Time Series HDF5 files.

```
tokio.tools.hdf5.enumerate_h5lmts(fsname, datetime_start, datetime_end)
    Alias for tokio.tools.hdf5.enumerate_hdf5()
```

```
tokio.tools.hdf5.enumerate_hdf5(fsname, datetime_start, datetime_end)
    Returns all time-indexed HDF5 files falling between a time range
```

Given a starting and ending datetime, returns the names of all HDF5 files that should contain data falling within that date range (inclusive).

Parameters

- **fsname** (*str*) – Logical file system name; should match a key within the `hdf5_files` config item in `site.json`.
- **datetime_start** (*datetime.datetime*) – Begin including files corresponding to this start date, inclusive.
- **datetime_end** (*datetime.datetime*) – Stop including files with timestamps that follow this end date. Resulting files *will* include this date.

Returns List of strings, each describing a path to an existing HDF5 file that should contain data relevant to the requested start and end dates.

Return type `list`

`tokio.tools.hdf5.get_dataframe_from_time_range` (*fsname*, *dataset_name*, *datetime_start*, *datetime_end*, *fix_errors=False*)

Returns all TOKIO Time Series data within a time range as a DataFrame.

Given a time range,

1. Find all TOKIO Time Series HDF5 files that exist and overlap with that time range
2. Open each and load all data that falls within the given time range
3. Convert loaded data into a single, time-indexed DataFrame

Parameters

- **fsname** (*str*) – Name of file system whose data should be retrieved. Should exist as a key within `tokio.config.CONFIG['hdf5_files']`
- **dataset_name** (*str*) – Dataset within each matching HDF5 file to load
- **datetime_start** (*datetime.datetime*) – Lower bound of time range to load, inclusive
- **datetime_end** (*datetime.datetime*) – Upper bound of time range to load, exclusive
- **fix_errors** (*bool*) – Replace negative values with -0.0. Necessary if any HDF5 files contain negative values as a result of being archived with a buggy version of pytokio.

Returns DataFrame indexed in time and whose columns correspond to those in the given *dataset_name*.

Return type `pandas.DataFrame`

`tokio.tools.hdf5.get_files_and_indices` (*fsname*, *dataset_name*, *datetime_start*, *datetime_end*)

Retrieve filenames and indices within files corresponding to a date range

Given a logical file system name and a dataset within that file system's TOKIO Time Series files, return a list of all file names and the indices within those files that fall within the specified date range.

Parameters

- **fsname** (*str*) – Logical file system name; should match a key within the `hdf5_files` config item in `site.json`.
- **dataset_name** (*str*) – Name of a TOKIO Time Series dataset name
- **datetime_start** (*datetime.datetime*) – Begin including files corresponding to this start date, inclusive.

- `datetime_end(datetime.datetime)` – Stop including files with timestamps that follow this end date. Resulting files `_will_` include this date.

Returns

List of three-item tuples of types (str, int, int), where

- element 0 is the path to an existing HDF5 file
- element 1 is the first index (inclusive) of `dataset_name` within that file containing data that falls within the specified date range
- element 2 is the last index (exclusive) of `dataset_name` within that file containing data that falls within the specified date range

Return type list

tokio.tools.jobinfo module

Site-independent interface to retrieve job info

`tokio.tools.jobinfo.get_job_nodes(jobid, cache_file=None)`

Return a list of all job nodes used.

Creates a list of all nodes used for a jobid.

Returns Set of node names used by the job described by this object

Return type set

Raises `tokio.ConfigError` – When no valid providers are found

`tokio.tools.jobinfo.get_job_startend(jobid, cache_file=None)`

Find earliest start and latest end time for a job.

Returns

Two-item tuple of (earliest start time, latest end time)

Return type tuple of `datetime.datetime`

Raises `tokio.ConfigError` – When no valid providers are found

tokio.tools.lfsstatus module

Given a file system and a datetime, return summary statistics about the OST fullness at that time

`tokio.tools.lfsstatus._summarize_failover(fs_data)`

Summarize failover data for a single time record

Given an `fs_data` dict, generate a dict of summary statistics. Expects `fs_data` dict of the form generated by `parse_lustre_txt.get_failovers`:

```
{
  "abnormal_ips": {
    "10.100.104.140": [
      "OST0087",
      "OST0086",
      ...
    ],
    "10.100.104.43": [
```

(continues on next page)

(continued from previous page)

```

        "OST0025",
        "OST0024",
        ...
    ]
},
"mode": 1
}

```

Parameters `fs_data` (*dict*) – a single timestamp and file system record taken from the output of `nersc_lfsstate.NerscLfsOstMap.get_failovers`

Returns summary metrics about the state of failovers on the file system

Return type `dict`

`tokio.tools.lfsstatus._summarize_fullness` (*fs_data*)

Summarize fullness data for a single time record

Given an `fs_data` dict, generate a dict of summary statistics. Expects `fs_data` dict of form generated by `nersc_lfsstate.NerscLfsOstFullness`:

```

{
  "MDT0000": {
    "mount_pt": "/scratch1",
    "remaining_kib": 2147035984,
    "target_index": 0,
    "total_kib": 2255453580,
    "used_kib": 74137712
  },
  "OST0000": {
    "mount_pt": "/scratch1",
    "remaining_kib": 28898576320,
    "target_index": 0,
    "total_kib": 90767651352,
    "used_kib": 60894630700
  },
  ...
}

```

Parameters `fs_data` (*dict*) – a single timestamp and file system record taken from a `nersc_lfsstate.NerscLfsOstFullness` object

Returns summary metrics about the state of the file system fullness

Return type `dict`

`tokio.tools.lfsstatus.get_failures` (*file_system*, *datetime_target*, ***kwargs*)

Get file system failures

Is a convenience wrapper for `get_summary`.

Parameters

- **file_system** (*str*) – Logical name of file system whose data should be retrieved (e.g., `cscratch`)
- **datetime_target** (*datetime.datetime*) – Time at which requested data should be retrieved

- **cache_file** (*str*) – Basename of file to search for the requested data

Returns various statistics about the file system fullness

Return type dict

```
tokio.tools.lfsstatus.get_failures_lfsstate(file_system, datetime_target,
                                           cache_file=None)
```

Get file system failures from nersc_lfsstate connector

Wrapper around the generic get_lfsstate function.

Parameters

- **file_system** (*str*) – Lustre file system name of the file system whose data should be retrieved (e.g., snx11025)
- **datetime_target** (*datetime.datetime*) – Time at which requested data should be retrieved
- **cache_file** (*str*) – Basename of file to search for the requested data

Returns Whatever is returned by get_lfsstate

```
tokio.tools.lfsstatus.get_fullness(file_system, datetime_target, **kwargs)
```

Get file system fullness

Is a convenience wrapper for *get_summary*.

Parameters

- **file_system** (*str*) – Logical name of file system whose data should be retrieved (e.g., cscratch)
- **datetime_target** (*datetime.datetime*) – Time at which requested data should be retrieved

Returns various statistics about the file system fullness

Return type dict

Raises tokio.ConfigError – When no valid providers are found

```
tokio.tools.lfsstatus.get_fullness_hdf5(file_system, datetime_target)
```

Get file system fullness from an HDF5 object

Given a file system name (e.g., snx11168) and a datetime object, return summary statistics about the OST fullness.

Parameters

- **file_system** (*str*) – Name of file system whose data should be retrieved
- **datetime_target** (*datetime.datetime*) – Time at which requested data should be retrieved

Returns various statistics about the file system fullness

Return type dict

Raises ValueError – if an OST name is encountered which does not conform to a naming convention from which an OST index can be derived

```
tokio.tools.lfsstatus.get_fullness_lfsstate(file_system, datetime_target,
                                           cache_file=None)
```

Get file system fullness from nersc_lfsstate connector

Wrapper around the generic get_lfsstate function.

Parameters

- **file_system** (*str*) – Lustre file system name of the file system whose data should be retrieved (e.g., snx11025)
- **datetime_target** (*datetime.datetime*) – Time at which requested data should be retrieved
- **cache_file** (*str*) – Basename of file to search for the requested data

Returns Whatever is returned by `tokio.tools.lfsstatus.get_lfsstate()`

`tokio.tools.lfsstatus.get_lfsstate` (*file_system, datetime_target, metric, cache_file=None*)
 Get file system fullness or failures

Given a file system name (e.g., snx11168) and a datetime object

1. locate and load the lfs-df (fullness) or ost map (failures) file
2. find the sample immediately preceding the datetime (don't find one that overlaps it)
3. return summary statistics about the OST fullness or OST failures

Parameters

- **file_system** (*str*) – Lustre file system name of the file system whose data should be retrieved (e.g., snx11025)
- **datetime_target** (*datetime.datetime*) – Time at which requested data should be retrieved
- **metric** (*str*) – either “fullness” or “failures”
- **cache_file** (*str*) – Basename of file to search for the requested data

Returns various statistics about the file system fullness

Return type `dict`

Raises

- `ValueError` – if `metric` does not contain a valid option
- `IOError` – when no valid data sources can be found for the given date

tokio.tools.nersc_mmperfmon module

Tool that translates datetimes to mmperfmon outputs containing the relevant data

`tokio.tools.nersc_mmperfmon.enumerate_mmperfmon_txt` (*fsname, datetime_start, datetime_end*)

Returns all time-indexed mmperfmon output text files falling between a time range.

Parameters

- **fsname** (*str*) – Logical file system name; should match a key within the `mmperfmon_output_files` config item in `site.json`.
- **datetime_start** (*datetime.datetime*) – Begin including files corresponding to this start date, inclusive.
- **datetime_end** (*datetime.datetime*) – Stop including files with timestamps that follow this end date. Resulting files `_will_` include this date.

Returns List of strings, each describing a path to an existing file that should contain data relevant to the requested start and end dates.

Return type `list`

tokio.tools.topology module

Perform operations based on the mapping of a job to network topology

`tokio.tools.topology.get_job_diameter` (*jobid*, *nodemap_cache_file=None*, *jobinfo_cache_file=None*)

Calculate the diameter of a job

An extremely crude way to reduce a job's node allocation into a scalar metric. Assumes nodes are equally capable and fall on a 3D network; calculates the center of mass of the job's node positions.

Parameters

- **jobid** (*str*) – A logical job id from which nodes are determined and their topological placement is determined
- **nodemap_cache_file** (*str*) – Full path to the file containing the cached contents to be used to determine the node position map
- **jobinfo_cache_file** (*str*) – Full path to the file containing the cached contents to be used to convert the job id into a node list

Returns

Contains three keys representing three ways in which a job's radius can be expressed. Keys are:

- **job_min_radius**: The smallest distance between the job's center of mass and a job node
- **job_max_radius**: The largest distance between the job's center of mass and a job node
- **job_avg_radius**: The average distance between the job's center of mass and all job nodes

Return type `dict`

1.5.2 Submodules

tokio.common module

Common convenience routines used throughout pytokio

exception `tokio.common.ConfigError`

Bases: `exceptions.RuntimeError`

class `tokio.common.JSONEncoder` (*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

Bases: `json.encoder.JSONEncoder`

Convert common pytokio data types into serializable formats

default (*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement `default` like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

`tokio.common.humanize_bytes` (*bytect*, *base10=False*, *fmt='%i.%f %s'*)

Converts bytes into human-readable units

Parameters

- **bytect** (*int*) – Number of bytes
- **base10** (*bool*) – Convert to base-10 units (MB, GB, etc) if True
- **fmt** (*str* or *None*) – Format of string to return; must contain *%f*/*%d* and *%s* for the quantity and units, respectively.

Returns Quantity and units expressed in a human-readable quantity

Return type *str*

`tokio.common.humanize_bytes_to` (*bytect*, *unit*, *fmt='%i.%f %s'*)

Converts bytes into a specific human-readable unit

Parameters

- **bytect** (*int*) – Number of bytes
- **unit** (*str*) – Unit in which bytes should be expressed
- **fmt** (*str* or *None*) – Format of string to return; must contain *%f*/*%d* and *%s* for the quantity and units, respectively.

Returns Quantity and units expressed in a human-readable quantity

Return type *str*

`tokio.common.isstr` (*obj*)

Determine if an object is a string or string-derivative

Provided for Python2/3 compatibility

Parameters *obj* – object to be tested for stringiness

Returns is it string-like?

Return type *bool*

`tokio.common.recast_string` (*value*)

Converts a string to some type of number or True/False if possible

Parameters *value* (*str*) – A string that may represent an int or float

Returns The most precise numerical or boolean representation of *value* if *value* is a valid string-encoded version of that type. Returns the unchanged string otherwise.

Return type *int*, *float*, *bool*, *str*, or *None*

`tokio.common.to_epoch` (*datetime_obj*, *astype=<type 'int'>*)

Convert *datetime.datetime* into epoch seconds

Currently assumes input datetime is expressed in localtime. Does not handle timezones very well. Once Python2 support is dropped from pytokio, this will be replaced by Python3's `datetime.datetime.timestamp()` method.

Parameters

- **datetime_obj** (*datetime.datetime*) – Datetime to convert to seconds-since-epoch
- **astype** – Whether you want the resulting timestamp as an int or float

Returns Seconds since epoch

Return type `int` or `float`

tokio.config module

Loads the pytokio configuration file.

The pytokio configuration file can be either a JSON or YAML file that contains various site-specific constants, paths, and defaults. pytokio should import correctly and all connectors should be functional in the absence of a configuration file.

If the PyYAML package is available, the configuration file may reference environment variables which get correctly resolved during import. This requires that the configuration file only reference \$VARIABLES in unquoted strings; quoted strings (such as those when loading JSON-formatted YAML) will not be expanded.

A subset of configuration parameters can be overridden by environment variables prefixed with **PYTOKIO_**.

```
tokio.config.CONFIG = {}
```

Global variable containing the configuration

```
tokio.config.DEFAULT_CONFIG_FILE = ''
```

Path of default site configuration file

```
tokio.config.MAGIC_VARIABLES = ['HDF5_FILES', 'ISDCT_FILES', 'LFSSTATUS_FULLNESS_FILES', 'LFSSTATUS_SIZE_FILES']
```

Config parameters that can be overridden using PYTOKIO_* environment variable

```
tokio.config.PYTOKIO_CONFIG_FILE = ''
```

Path to configuration file to load

```
tokio.config.init_config()
```

Loads the global configuration.

Loads the site-wide configuration file, then inspects relevant environment variables for overrides.

```
tokio.config.load_and_expand_yaml(file_handle)
```

Loads YAML config file and expands all environment variables contained therein

Parameters **file_handle** (*file*) – File containing YAML to load

tokio.debug module

```
tokio.debug.debug_print(string)
```

Print debug messages if the module's global debug flag is enabled.

```
tokio.debug.error(string)
```

Handle errors generated within TOKIO. Currently just a passthrough to stderr; should probably provide exceptions later on.

```
tokio.debug.warning(string)
```

Handle warnings generated within TOKIO. Currently just a passthrough to stderr; should probably provide a more rigorous logging/reporting interface later on.

tokio.timeseries module

TimeSeries class to simplify updating and manipulating the in-memory representation of time series data.

```
class tokio.timeseries.TimeSeries (dataset_name=None,      start=None,      end=None,
                                   timestep=None,         num_columns=None,      col-
                                   umn_names=None, timestamp_key=None, sort_hex=False)
```

Bases: `object`

In-memory representation of an HDF5 group in a TokioFile. Can either initialize with no datasets, or initialize against an existing HDF5 group.

add_column (*column_name*)

Add a new column and update the column map

add_rows (*num_rows=1*)

Add additional rows to the end of self.dataset and self.timestamps

convert_to_deltas (*align='l'*)

Converts a matrix of monotonically increasing rows into deltas.

Replaces self.dataset with a matrix with the same number of columns but one fewer row (taken off the bottom of the matrix). Also adjusts the timestamps dataset.

Parameters align (*str*) – “left” or “right”. Determines whether the contents of a cell labeled with timestamp t0 contains the data between t0 and t0 + dt (left) or t0 and t0 - dt (right).

get_insert_pos (*timestamp, column_name, create_col=False, align='l'*)

Determine col and row indices corresponding to timestamp and col name

Parameters

- **timestamp** (*datetime.datetime*) – Timestamp to map to a row index
- **column_name** (*str*) – Name of column to map to a column index
- **create_col** (*bool*) – If column_name does not exist, create it?
- **align** (*str*) – “left” or “right”; governs whether or not the value given for the timestamp argument represents the left or right edge of the bin.

Returns (t_index, c_index) (long or None)

init (*start, end, timestep, num_columns, dataset_name, column_names=None, timestamp_key=None*)

Create a new TimeSeries dataset object

Responsible for setting self.timestep, self.timestamp_key, and self.timestamps

Parameters

- **start** (*datetime*) – timestamp to correspond with the 0th index
- **end** (*datetime*) – timestamp at which timeseries will end (exclusive)
- **timestep** (*int*) – seconds between consecutive timestamp indices
- **num_columns** (*int*) – number of columns to initialize in the numpy.ndarray
- **dataset_name** (*str*) – an HDF5-compatible name for this timeseries
- **column_names** (*list of str, optional*) – strings by which each column should be indexed. Must be less than or equal to num_columns in length; difference remains uninitialized

- **timestamp_key** (*str*, *optional*) – an HDF5-compatible name for this time-series’ timestamp vector. Default is /groupname/timestamps

insert_element (*timestamp*, *column_name*, *value*, *reducer=None*, *align='l'*)

Inserts a value into a (timestamp, column) element

Given a timestamp (datetime.datetime object) and a column name (string), update an element of the dataset. If a reducer function is provided, use that function to reconcile any existing values in the element to be updated.

Parameters

- **timestamp** (*datetime.datetime*) – Determines the row index into which *value* should be inserted
- **column_name** (*str*) – Determines the column into which *value* should be inserted
- **value** – Value to insert into the dataset
- **reducer** (*function or None*) – If a value already exists for the given (timestamp, column_name) coordinate, apply this function to the existing value and the input *value* and store the result. If None, just overwrite the existing value.
- **align** (*str*) – “left” or “right”; governs whether or not the value given for the *timestamp* argument represents the left or right edge of the bin.

Returns True if insertion was successful, False if no action was taken

Return type `bool`

rearrange_columns (*new_order*)

Rearrange the dataset’s columnar data by an arbitrary column order given as an enumerable list

set_columns (*column_names*)

Set the list of column names

set_timestamp_key (*timestamp_key*, *safe=False*)

Set the timestamp key

Parameters

- **timestamp_key** (*str*) – The key for the timestamp dataset.
- **safe** (*bool*) – If true, do not overwrite an existing timestamp key

sort_columns ()

Rearrange the dataset’s column data by sorting them by their headings

swap_columns (*index1*, *index2*)

Swap two columns of the dataset in-place

trim_rows (*num_rows=1*)

Trim some rows off the end of self.dataset and self.timestamps

update_column_map ()

Create the mapping of column names to column indices

`tokio.timeseries.sorted_nodenames` (*nodenames*, *sort_hex=False*)

Garly routine to sort nodenames naturally. Required for nodes named things like ‘bb23’ and ‘bb231’.

`tokio.timeseries.timeseries_deltas` (*dataset*)

Convert monotonically increasing values into deltas

Subtract every row of the dataset from the row that precedes it to convert a matrix of monotonically increasing rows into deltas. This is a lossy process because the deltas for the final measurement of the time series cannot be calculated.

Parameters `dataset` (*numpy.ndarray*) – The dataset to convert from absolute values into deltas. rows should correspond to time, and columns to individual components

Returns

The deltas between each row in the given input dataset. Will have the same number of columns as the input dataset and one fewer rows.

Return type *numpy.ndarray*

1.6 pytokio Release Process

1.6.1 Branching process

General branching model

What are the principal development and release branches?

- `master` contains complete features but is not necessarily bug-free
- `rc` contains stable code
- Version branches (e.g., `0.12`) contain code that is on track for release

Where to commit code?

- All features should land in `master` once they are complete and pass tests
- `rc` should only receive merge or cherry-pick from `master`, no other branches
- Version branches should only receive merge or cherry-pick from `rc`, no other branches

How should commits flow between branches?

- `rc` should *never* be merged back into `master`
- Version branches should *never* be merged into `rc`
- Hotfixes that cannot land in `master` (e.g., because a feature they fix no longer exists) should go directly to the `rc` branch (if appropriate) and/or version branch.

General versioning guidelines

The authoritative version of pytokio is contained in `tokio/__init__.py` and nowhere else.

1. The `master` branch should always be at least one minor number above `rc`
2. Both `master` and `rc` branches should have versions suffixed with `.devX` where `X` is an arbitrary integer
3. Only version branches (`0.11`, `0.12`) should have versions that end in `b1`, `b2`, etc
4. Only version branches should have release versions (`0.11.0`)

Generally, the tip of a version branch should be one beta release ahead of what has actually been released so that subsequent patches automatically have a version reflecting a higher number than the last release.

Feature freezing

This is done by

1. Merge master into rc
2. In master, update version in `tokio/__init__.py` from `0.N.0.devX` to `0.(N+1).0.dev1`
3. Commit to master

Cutting a first beta release

1. Create a branch from rc called `0.N`
2. In that branch, update the version from `0.N.0.devX` to `0.N.0b1`
3. Commit to `0.N`
4. Tag/release `v0.N.0b1` from GitHub's UI from the `0.N` branch
5. Update the version in `0.N` from `0.N.0b1` to `0.N.0b2` to prepare for a hypothetical next release
6. Commit to `0.N`

Applying fixes to a beta release

1. Merge changes into `master` if the fix still applies there. Commit changes to `rc` if the fix still applies there, or commit to the version branch otherwise.
2. Cherry-pick the changes into downstream branches (`rc` if committed to `master`, version branch from `rc`)

Cutting a second beta release

1. Tag the version (`git tag v0.N.0-beta2`) on the `0.N` branch
2. `git push --tags` to send the new tag up to GitHub
3. **Make sure the tag passes all tests in Travis**
4. Build the source tarball using the release process described in the [Releasing pytokio](#) section
5. Release `v0.N.0-beta2` from GitHub's UI and upload the tarball from the previous step
6. Update the version in `0.N` from `0.N.0b2` to `0.N.0b3` (or `b4`, etc) to prepare for a hypothetical next release
7. Commit to `0.N`

1.6.2 Releasing pytokio

Ensure that the `tokio.__version__` (in `tokio/__init__.py`) is correctly set in the version branch from which you would like to cut a release.

Then edit `setup.py` and set `RELEASE = True`.

Build the source distribution:

```
python setup.py sdist
```

The resulting build should be in the `dist/` subdirectory.

It is recommended that you do this all from within a minimal Docker environment for cleanliness.

1.6.3 Testing on Docker

pytokio now includes a `dockerfile` which will can be used to build and test pytokio in a clean environment.

To test pytokio directly from GitHub, you can:

```
docker build -t pytokio_test https://github.com/nersc/pytokio.git
```

Or if you have the repository checked out locally,:

```
docker build -t pytokio_test .
```

Once the image is built, simply:

```
docker run -it pytokio_test
```

to execute the full test suite.

1.6.4 Packaging pytokio

Create `~/.pypirc` with permissions `0600x` and contents:

```
[pypi]
username = <username>
password = <password>
```

Then do a standard `sdist` build:

```
python setup.py sdist
```

and upload it to pypi:

```
twine upload -r testpypi dist/pytokio-0.10.1b2.tar.gz
```

and ensure that `testpypi` is defined in `.pypirc`:

```
[testpypi]
repository = https://test.pypi.org/legacy/
username = <username>
password = <password>
```

1.6.5 More Info

See <https://packaging.python.org/guides/distributing-packages-using-setuptools/>

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

CHAPTER 3

Copyright

Total Knowledge of I/O Copyright (c) 2017, The Regents of the University of California, through Lawrence Berkeley National Laboratory (subject to receipt of any required approvals from the U.S. Dept. of Energy). All rights reserved.

If you have questions about your rights to use or distribute this software, please contact Berkeley Lab's Innovation & Partnerships Office at IPO@lbl.gov.

NOTICE. This Software was developed under funding from the U.S. Department of Energy and the U.S. Government consequently retains certain rights. As such, the U.S. Government has been granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable, worldwide license in the Software to reproduce, distribute copies to the public, prepare derivative works, and perform publicly and display publicly, and to permit other to do so.

t

tokio, 15
tokio.analysis, 15
tokio.analysis.umami, 15
tokio.cli, 16
tokio.cli.archive_collectdes, 16
tokio.cli.archive_esnet_snmp, 19
tokio.cli.archive_lmtdb, 20
tokio.cli.archive_mmperfmon, 22
tokio.cli.cache_collectdes, 25
tokio.cli.cache_darshan, 25
tokio.cli.cache_esnet_snmp, 25
tokio.cli.cache_isdct, 25
tokio.cli.cache_lfsstatus, 25
tokio.cli.cache_lmtdb, 25
tokio.cli.cache_mmperfmon, 26
tokio.cli.cache_nersc_globuslogs, 26
tokio.cli.cache_nersc_jobsdb, 26
tokio.cli.cache_slurm, 26
tokio.cli.cache_topology, 26
tokio.cli.compare_isdct, 26
tokio.cli.darshan_bad_ost, 27
tokio.cli.darshan_scoreboard, 28
tokio.cli.find_darshanlogs, 28
tokio.cli.summarize_h5lmt, 28
tokio.cli.summarize_job, 29
tokio.cli.summarize_tts, 30
tokio.common, 87
tokio.config, 89
tokio.connectors, 31
tokio.connectors._hdf5, 31
tokio.connectors.cachingdb, 33
tokio.connectors.collectd_es, 36
tokio.connectors.common, 37
tokio.connectors.crayfdb, 40
tokio.connectors.darshan, 40
tokio.connectors.es, 44
tokio.connectors.esnet_snmp, 48
tokio.connectors.globuslogs, 52
tokio.connectors.hdf5, 54
tokio.connectors.hpss, 57
tokio.connectors.lfshealth, 63
tokio.connectors.lmtdb, 63
tokio.connectors.mmperfmon, 65
tokio.connectors.nersc_globuslogs, 68
tokio.connectors.nersc_isdct, 69
tokio.connectors.nersc_jobsdb, 72
tokio.connectors.nersc_lfsstate, 73
tokio.connectors.slurm, 75
tokio.debug, 89
tokio.timeseries, 90
tokio.tools, 79
tokio.tools.common, 79
tokio.tools.darshan, 80
tokio.tools.hdf5, 81
tokio.tools.jobinfo, 83
tokio.tools.lfsstatus, 83
tokio.tools.nersc_mmperfmon, 86
tokio.tools.topology, 87

Symbols

- __RECAST_KEY_MAP (in *tokio.connectors.slurm*), 78
 __REX_LFS_DF (in *tokio.connectors.nersc_lfsstate*), 75
 __REX_OST_MAP (in *tokio.connectors.nersc_lfsstate*), 75
 __getitem__() (*tokio.connectors._hdf5.MappedDataset* method), 31
 __getitem__() (*tokio.connectors.hdf5.Hdf5* method), 54
 __init__() (*tokio.cli.archive_esnet_snmp.Archiver* method), 19
 __init__() (*tokio.cli.archive_mmpfmon.Archiver* method), 22
 __init__() (*tokio.connectors._hdf5.MappedDataset* method), 31
 __init__() (*tokio.connectors.cachingdb.CachingDb* method), 33
 __init__() (*tokio.connectors.common.CacheableDict* method), 37
 __init__() (*tokio.connectors.craysdb.CraySdbProc* method), 40
 __init__() (*tokio.connectors.darshan.Darshan* method), 41
 __init__() (*tokio.connectors.es.EsConnection* method), 45
 __init__() (*tokio.connectors.esnet_snmp.EsnetSnm* method), 49
 __init__() (*tokio.connectors.hdf5.Hdf5* method), 54
 __init__() (*tokio.connectors.lmtdb.LmtDb* method), 63
 __init__() (*tokio.connectors.nersc_isdct.NerscIsdct* method), 70
 __init__() (*tokio.connectors.nersc_lfsstate.NerscLfsOstFullness* method), 73
 __init__() (*tokio.connectors.nersc_lfsstate.NerscLfsOstMap* method), 74
 __init__() (*tokio.connectors.slurm.Slurm* method), 76
 __repr__() (*tokio.connectors.craysdb.CraySdbProc* method), 40
 __repr__() (*tokio.connectors.darshan.Darshan* method), 41
 __repr__() (*tokio.connectors.lfshealth.LfsOstFullness* method), 63
 __repr__() (*tokio.connectors.lfshealth.LfsOstMap* method), 63
 __repr__() (*tokio.connectors.mmpfmon.Mmpfmon* method), 66
 __repr__() (*tokio.connectors.nersc_lfsstate.NerscLfsOstFullness* method), 73
 __repr__() (*tokio.connectors.nersc_lfsstate.NerscLfsOstMap* method), 74
 __repr__() (*tokio.connectors.slurm.Slurm* method), 76
 _apply_timestep() (in *tokio.connectors._hdf5*), 32
 _convert_counters() (in *tokio.cli.compare_isdct*), 26
 _darshan_parser() (*tokio.connectors.darshan.Darshan* method), 41
 _decode_nersc_nid() (in *tokio.connectors.nersc_isdct*), 71
 _expand_check_paths() (in *tokio.tools.common*), 79
 _find_columns() (in *tokio.connectors.hpss*), 60
 _get_ascii_resolution() (in *tokio.connectors.hpss*), 60
 _get_columns_h5lmt() (*tokio.connectors.hdf5.Hdf5* method), 55
 _get_interval_result() (in *tokio.connectors.esnet_snmp*), 52
 _get_missing_h5lmt() (*tokio.connectors.hdf5.Hdf5* method), 55
 _get_url() (in *tokio.connectors.esnet_snmp*), 52

_hpss_timedelta_to_secs() (in module *tokio.connectors.hpss*), 61
 _identify_fs_from_path() (in module *tokio.cli.summarize_job*), 29
 _insert_result() (*tokio.connectors.esnet_snmp.EsnetSnmp* dataframe() method), 50
 _listify_ips() (in module *tokio.connectors.globuslogs*), 53
 _load_subprocess() (*tokio.connectors.common.SubprocessOutputDict* method), 38
 _load_subprocess() (*tokio.connectors.common.SubprocessOutputList* method), 39
 _load_subprocess_iter() (*tokio.connectors.darshan.Darshan* method), 41
 _match_files() (in module *tokio.tools.common*), 79
 _merge_parsed_counters() (in module *tokio.connectors.nersc_isdct*), 71
 _normalize_key() (in module *tokio.connectors.nersc_isdct*), 71
 _one_column() (in module *tokio.connectors.hdf5*), 32
 _parse_darshan_parser() (*tokio.connectors.darshan.Darshan* method), 41
 _parse_section() (in module *tokio.connectors.hpss*), 61
 _process_page() (*tokio.connectors.es.EsConnection* method), 45
 _query_mysql() (*tokio.connectors.cachingdb.CachingDb* method), 34
 _query_sqlite3() (*tokio.connectors.cachingdb.CachingDb* method), 34
 _recast_keys() (*tokio.connectors.slurm.Slurm* method), 76
 _rekey_smart_buffer() (in module *tokio.connectors.nersc_isdct*), 71
 _rekey_table() (in module *tokio.connectors.hpss*), 62
 _resolve_schema_key() (*tokio.connectors.hdf5.Hdf5* method), 55
 _save_cache() (*tokio.connectors.common.CacheableDict* method), 37
 _save_cache() (*tokio.connectors.nersc_ufsstate.NerscUfsOstFullness* method), 73
 _save_cache() (*tokio.connectors.nersc_ufsstate.NerscUfsOstMap* method), 74
 _serialize_datetime() (in module *tokio.analysis.umami*), 16
 _summarize_failover() (in module *tokio.tools.ifsstatus*), 83
 _summarize_fullness() (in module *tokio.tools.ifsstatus*), 84
 _synthesize_metrics() (*tokio.connectors.nersc_isdct.NerscIsdct* method), 70
 dataframe() (*tokio.connectors.hdf5.Hdf5* method), 55
 _to_dataframe_h5lmt() (*tokio.connectors.hdf5.Hdf5* method), 55
 _to_dict_for_pandas() (*tokio.analysis.umami.Umami* method), 15

A

add_column() (*tokio.timeseries.TimeSeries* method), 90
 add_rows() (*tokio.timeseries.TimeSeries* method), 90
 append() (*tokio.analysis.umami.UmamiMetric* method), 16
 archive() (*tokio.cli.archive_esnet_snmp.Archiver* method), 19
 archive() (*tokio.cli.archive_mmperfmon.Archiver* method), 23
 archive_esnet_snmp() (in module *tokio.cli.archive_esnet_snmp*), 20
 archive_lmtdb() (in module *tokio.cli.archive_lmtdb*), 22
 archive_mds_data() (*tokio.cli.archive_lmtdb.DatasetDict* method), 21
 archive_mds_ops_data() (*tokio.cli.archive_lmtdb.DatasetDict* method), 21
 archive_mmperfmon() (in module *tokio.cli.archive_mmperfmon*), 24
 archive_oss_data() (*tokio.cli.archive_lmtdb.DatasetDict* method), 21
 archive_ost_data() (*tokio.cli.archive_lmtdb.DatasetDict* method), 21
 Archiver (class in *tokio.cli.archive_esnet_snmp*), 19
 Archiver (class in *tokio.cli.archive_mmperfmon*), 22

B

bin_dataset() (in module *tokio.cli.summarize_h5lmt*), 28
 bin_datasets() (in module *tokio.cli.summarize_h5lmt*), 28
 build_timeseries_query() (in module *tokio.connectors.es*), 47

C

CacheableDict (class in *tokio.connectors.common*), 37
 CachingDb (class in *tokio.connectors.cachingdb*), 33

close() (*tokio.connectors.cachingdb.CachingDb method*), 34

close() (*tokio.connectors.es.EsConnection method*), 46

close_cache() (*tokio.connectors.cachingdb.CachingDb method*), 34

CollectdEs (*class in tokio.connectors.collectd_es*), 36

commit_timeseries() (*tokio.connectors.hdf5.Hdf5 method*), 55

compact_nodelist() (*in module tokio.connectors.slurm*), 78

CONFIG (*in module tokio.config*), 89

ConfigError, 87

connect() (*tokio.connectors.cachingdb.CachingDb method*), 35

connect() (*tokio.connectors.es.EsConnection method*), 46

connect_cache() (*tokio.connectors.cachingdb.CachingDb method*), 35

convert_byte_keys() (*in module tokio.cli.compare_isdct*), 26

convert_counts_rates() (*in module tokio.connectors._hdf5*), 32

convert_deltas() (*tokio.cli.archive_lmtdb.DatasetDict method*), 21

convert_to_deltas() (*tokio.timeseries.TimeSeries method*), 90

correlate_ost_performance() (*in module tokio.cli.darshan_bad_ost*), 27

CraySdbProc (*class in tokio.connectors.craybdb*), 40

D

Darshan (*class in tokio.connectors.darshan*), 41

darshan_parser_base() (*tokio.connectors.darshan.Darshan method*), 42

darshan_parser_perf() (*tokio.connectors.darshan.Darshan method*), 42

darshan_parser_total() (*tokio.connectors.darshan.Darshan method*), 42

darshanlogs_to_ost_dataframe() (*in module tokio.cli.darshan_bad_ost*), 27

dataset2metadataset_key() (*in module tokio.cli.archive_collectdes*), 16

DatasetDict (*class in tokio.cli.archive_lmtdb*), 20

debug_print() (*in module tokio.debug*), 89

default() (*tokio.common.JSONEncoder method*), 87

default() (*tokio.connectors.slurm.SlurmEncoder method*), 77

DEFAULT_CONFIG_FILE (*in module tokio.config*), 89

demux_column() (*in module tokio.connectors._hdf5*), 32

diff() (*tokio.connectors.nersc_isdct.NerscIsdct method*), 70

discover_errors() (*in module tokio.cli.compare_isdct*), 27

drop_cache() (*tokio.connectors.cachingdb.CachingDb method*), 35

drop_cache() (*tokio.connectors.nersc_jobsdb.NerscJobsDb method*), 72

E

endpoint_name() (*in module tokio.cli.archive_esnet_snmp*), 20

enumerate_dated_files() (*in module tokio.tools.common*), 80

enumerate_h5lmts() (*in module tokio.tools.hdf5*), 81

enumerate_hdf5() (*in module tokio.tools.hdf5*), 81

enumerate_mmperfmon_txt() (*in module tokio.tools.nersc_mmperfmon*), 86

error() (*in module tokio.debug*), 89

EsConnection (*class in tokio.connectors.es*), 44

EsnetSnmp (*class in tokio.connectors.esnet_snmp*), 49

estimate_darshan_perf() (*in module tokio.cli.darshan_bad_ost*), 27

expand_nodelist() (*in module tokio.connectors.slurm*), 78

F

finalize() (*tokio.cli.archive_esnet_snmp.Archiver method*), 19

finalize() (*tokio.cli.archive_lmtdb.DatasetDict method*), 21

finalize() (*tokio.cli.archive_mmperfmon.Archiver method*), 23

find_darshanlogs() (*in module tokio.tools.darshan*), 80

from_cache() (*tokio.connectors.collectd_es.CollectdEs class method*), 36

from_cache() (*tokio.connectors.es.EsConnection class method*), 46

from_cache() (*tokio.connectors.nersc_globuslogs.NerscGlobusLogs class method*), 68

from_file() (*tokio.connectors.globuslogs.GlobusLog class method*), 53

from_file() (*tokio.connectors.hpss.FtpLog class method*), 58

from_file() (*tokio.connectors.hpss.HpssDailyReport class method*), 58

from_file() (*tokio.connectors.hpss.HsiLog class method*), 60

from_file() (*tokio.connectors.mmperfmon.Mmperfmon class method*), 66

from_json() (*tokio.connectors.slurm.Slurm method*), 76

- `from_str()` (*tokio.connectors.globuslogs.GlobusLog class method*), 53
- `from_str()` (*tokio.connectors.hpss.FtpLog class method*), 58
- `from_str()` (*tokio.connectors.hpss.HpssDailyReport class method*), 58
- `from_str()` (*tokio.connectors.hpss.HsiLog class method*), 60
- `from_str()` (*tokio.connectors.mmpfmon.Mmpfmon class method*), 66
- `FtpLog` (*class in tokio.connectors.hpss*), 57
- ## G
- `gen_url()` (*tokio.connectors.esnet_snmp.EsnetSnmp method*), 51
- `get_biggest_api()` (*in module tokio.cli.summarize_job*), 29
- `get_biggest_fs()` (*in module tokio.cli.summarize_job*), 29
- `get_col_pos()` (*in module tokio.connectors.mmpfmon*), 67
- `get_columns()` (*tokio.connectors.hdf5.Hdf5 method*), 55
- `get_concurrent_jobs()` (*tokio.connectors.nersc_jobsdb.NerscJobsDb method*), 72
- `get_dataframe_from_time_range()` (*in module tokio.tools.hdf5*), 82
- `get_failovers()` (*tokio.connectors.lfshealth.LfsOstMap method*), 63
- `get_failovers()` (*tokio.connectors.nersc_lfsstate.NerscLfsOstMap method*), 74
- `get_failures()` (*in module tokio.tools.lfsstatus*), 84
- `get_failures_lfsstate()` (*in module tokio.tools.lfsstatus*), 85
- `get_files_and_indices()` (*in module tokio.tools.hdf5*), 82
- `get_fullness()` (*in module tokio.tools.lfsstatus*), 85
- `get_fullness_hdf5()` (*in module tokio.tools.lfsstatus*), 85
- `get_fullness_lfsstate()` (*in module tokio.tools.lfsstatus*), 85
- `get_index()` (*tokio.connectors.hdf5.Hdf5 method*), 55
- `get_insert_indices()` (*in module tokio.connectors.hdf5*), 57
- `get_insert_pos()` (*tokio.timeseries.TimeSeries method*), 90
- `get_interface_counters()` (*tokio.connectors.esnet_snmp.EsnetSnmp method*), 51
- `get_job_diameter()` (*in module tokio.tools.topology*), 87
- `get_job_ids()` (*tokio.connectors.slurm.Slurm method*), 76
- `get_job_nodes()` (*in module tokio.tools.jobinfo*), 83
- `get_job_nodes()` (*tokio.connectors.slurm.Slurm method*), 76
- `get_job_startend()` (*in module tokio.tools.jobinfo*), 83
- `get_job_startend()` (*tokio.connectors.nersc_jobsdb.NerscJobsDb method*), 72
- `get_job_startend()` (*tokio.connectors.slurm.Slurm method*), 76
- `get_lfsstate()` (*in module tokio.tools.lfsstatus*), 86
- `get_mds_data()` (*tokio.connectors.lmtdb.LmtDb method*), 63
- `get_mds_ops_data()` (*tokio.connectors.lmtdb.LmtDb method*), 64
- `get_missing()` (*tokio.connectors.hdf5.Hdf5 method*), 55
- `get_oss_data()` (*tokio.connectors.lmtdb.LmtDb method*), 64
- `get_ost_data()` (*tokio.connectors.lmtdb.LmtDb method*), 64
- `get_paramstyle_symbol()` (*in module tokio.connectors.cachingdb*), 36
- `get_timeseries_data()` (*tokio.connectors.lmtdb.LmtDb method*), 65
- `get_timestamps()` (*in module tokio.connectors._hdf5*), 33
- `get_timestamps()` (*tokio.connectors.hdf5.Hdf5 method*), 56
- `get_timestamps_key()` (*in module tokio.connectors._hdf5*), 33
- `get_timestep()` (*tokio.connectors.hdf5.Hdf5 method*), 56
- `get_ts_ids()` (*tokio.connectors.lmtdb.LmtDb method*), 65
- `get_version()` (*tokio.connectors.hdf5.Hdf5 method*), 56
- `GlobusLog` (*class in tokio.connectors.globuslogs*), 52
- ## H
- `Hdf5` (*class in tokio.connectors.hdf5*), 54
- `HpssDailyReport` (*class in tokio.connectors.hpss*), 58
- `HsiLog` (*class in tokio.connectors.hpss*), 58
- `humanize_bytes()` (*in module tokio.common*), 88
- `humanize_bytes_to()` (*in module tokio.common*), 88
- `humanize_units()` (*in module tokio.cli.summarize_tts*), 30
- ## I
- `init()` (*tokio.timeseries.TimeSeries method*), 90

- init_config()* (in module *tokio.config*), 89
init_dataset() (*tokio.cli.archive_mmperfmon.Archiver* method), 23
init_datasets() (*tokio.cli.archive_esnet_snmp.Archiver* method), 19
init_datasets() (*tokio.cli.archive_lmtdb.DatasetDict* method), 21
init_datasets() (*tokio.cli.archive_mmperfmon.Archiver* method), 23
init_hdf5_file() (in module *tokio.cli.archive_esnet_snmp*), 20
init_hdf5_file() (in module *tokio.cli.archive_lmtdb*), 22
init_hdf5_file() (in module *tokio.cli.archive_mmperfmon*), 24
insert_element() (*tokio.timeseries.TimeSeries* method), 91
isstr() (in module *tokio.common*), 88
- J**
- jobs_running_between()* (in module *tokio.connectors.slurm*), 78
JSONEncoder (class in *tokio.common*), 87
- L**
- LfsOstFullness* (class in *tokio.connectors.lfshealth*), 63
LfsOstMap (class in *tokio.connectors.lfshealth*), 63
LmtDb (class in *tokio.connectors.lmtdb*), 63
load() (*tokio.connectors.common.CacheableDict* method), 37
load() (*tokio.connectors.common.SubprocessOutputDict* method), 38
load() (*tokio.connectors.common.SubprocessOutputList* method), 39
load() (*tokio.connectors.darshan.Darshan* method), 42
load() (*tokio.connectors.mmperfmon.Mmperfmon* method), 66
load() (*tokio.connectors.nersc_isdct.NerscIsdct* method), 70
load() (*tokio.connectors.slurm.Slurm* method), 77
load_and_expand_yaml() (in module *tokio.config*), 89
load_cache() (*tokio.connectors.common.SubprocessOutputDict* method), 38
load_cache() (*tokio.connectors.common.SubprocessOutputList* method), 39
load_cache() (*tokio.connectors.mmperfmon.Mmperfmon* method), 66
load_darshanlogs() (in module *tokio.tools.darshan*), 81
load_json() (*tokio.connectors.common.CacheableDict* method), 37
load_json() (*tokio.connectors.esnet_snmp.EsnetSnmp* method), 51
load_keys() (*tokio.connectors.slurm.Slurm* method), 77
load_multiple() (*tokio.connectors.mmperfmon.Mmperfmon* method), 66
load_native() (*tokio.connectors.common.CacheableDict* method), 38
load_native() (*tokio.connectors.nersc_isdct.NerscIsdct* method), 70
load_ost_fullness_file() (*tokio.connectors.nersc_lfsstate.NerscLfsOstFullness* method), 73
load_ost_map_file() (*tokio.connectors.nersc_lfsstate.NerscLfsOstMap* method), 74
load_str() (*tokio.connectors.common.SubprocessOutputDict* method), 38
load_str() (*tokio.connectors.common.SubprocessOutputList* method), 39
load_str() (*tokio.connectors.crayfdb.CraySdbProc* method), 40
load_str() (*tokio.connectors.darshan.Darshan* method), 42
load_str() (*tokio.connectors.globuslogs.GlobusLog* method), 53
load_str() (*tokio.connectors.hpss.FtpLog* method), 58
load_str() (*tokio.connectors.hpss.HpssDailyReport* method), 58
load_str() (*tokio.connectors.hpss.HsiLog* method), 60
load_str() (*tokio.connectors.lfshealth.LfsOstFullness* method), 63
load_str() (*tokio.connectors.lfshealth.LfsOstMap* method), 63
load_str() (*tokio.connectors.mmperfmon.Mmperfmon* method), 67
load_str() (*tokio.connectors.slurm.Slurm* method), 77
lun_type() (*tokio.cli.archive_mmperfmon.Archiver* method), 23
- M**
- MAGIC_VARIABLES* (in module *tokio.config*), 89
main() (in module *tokio.cli.archive_collectdes*), 17
main() (in module *tokio.cli.archive_esnet_snmp*), 20
main() (in module *tokio.cli.archive_lmtdb*), 22
main() (in module *tokio.cli.archive_mmperfmon*), 24
main() (in module *tokio.cli.cache_collectdes*), 25
main() (in module *tokio.cli.cache_darshan*), 25
main() (in module *tokio.cli.cache_esnet_snmp*), 25
main() (in module *tokio.cli.cache_isdct*), 25
main() (in module *tokio.cli.cache_lfsstatus*), 25

main() (in module *tokio.cli.cache_lmtdb*), 25
 main() (in module *tokio.cli.cache_mmperfmon*), 26
 main() (in module *tokio.cli.cache_nersc_globuslogs*), 26
 main() (in module *tokio.cli.cache_nersc_jobsdb*), 26
 main() (in module *tokio.cli.cache_slurm*), 26
 main() (in module *tokio.cli.cache_topology*), 26
 main() (in module *tokio.cli.compare_isdct*), 27
 main() (in module *tokio.cli.darshan_bad_ost*), 27
 main() (in module *tokio.cli.darshan_scoreboard*), 28
 main() (in module *tokio.cli.find_darshanlogs*), 28
 main() (in module *tokio.cli.summarize_h5lmt*), 29
 main() (in module *tokio.cli.summarize_job*), 29
 main() (in module *tokio.cli.summarize_tts*), 30
 map_dataset() (in module *tokio.connectors.hdf5*), 33
 MappedDataset (class in *tokio.connectors.hdf5*), 31
 merge_dicts() (in module *tokio.cli.summarize_job*), 29
 metadataset2dataset_key() (in module *tokio.cli.archive_collectdes*), 17
 missing_values() (in module *tokio.connectors.hdf5*), 57
 Mmperfmon (class in *tokio.connectors.mmperfmon*), 66
 mutate_query() (in module *tokio.connectors.es*), 48

N

NerscGlobusLogs (class in *tokio.connectors.nersc_globuslogs*), 68
 NerscIsdct (class in *tokio.connectors.nersc_isdct*), 70
 NerscJobsDb (class in *tokio.connectors.nersc_jobsdb*), 72
 NerscLfsOstFullness (class in *tokio.connectors.nersc_lfsstate*), 73
 NerscLfsOstMap (class in *tokio.connectors.nersc_lfsstate*), 74
 normalize_cpu_datasets() (in module *tokio.cli.archive_collectdes*), 17

P

pages_to_hdf5() (in module *tokio.cli.archive_collectdes*), 17
 parse_base_counters() (in module *tokio.connectors.darshan*), 43
 parse_counters_fileobj() (in module *tokio.connectors.nersc_isdct*), 71
 parse_filename_metadata() (in module *tokio.connectors.darshan*), 43
 parse_header() (in module *tokio.connectors.darshan*), 43
 parse_mounts() (in module *tokio.connectors.darshan*), 44
 parse_perf_counters() (in module *tokio.connectors.darshan*), 44

parse_sacct() (in module *tokio.connectors.slurm*), 78
 parse_total_counters() (in module *tokio.connectors.darshan*), 44
 plot() (*tokio.analysis.umami.Umami* method), 15
 pop() (*tokio.analysis.umami.UmamiMetric* method), 16
 print_column_summary() (in module *tokio.cli.summarize_tts*), 30
 print_data_summary() (in module *tokio.cli.summarize_h5lmt*), 29
 print_datum() (in module *tokio.cli.summarize_h5lmt*), 29
 print_summary() (in module *tokio.cli.compare_isdct*), 27
 print_timestep_summary() (in module *tokio.cli.summarize_tts*), 30
 print_top() (in module *tokio.cli.darshan_scoreboard*), 28
 print_tts_hdf5_summary() (in module *tokio.cli.summarize_tts*), 30
 process_page() (in module *tokio.cli.archive_collectdes*), 17
 PYTOKIO_CONFIG_FILE (in module *tokio.config*), 89

Q

query() (*tokio.connectors.cachingdb.CachingDb* method), 35
 query() (*tokio.connectors.es.EsConnection* method), 46
 query() (*tokio.connectors.nersc_globuslogs.NerscGlobusLogs* method), 68
 query() (*tokio.connectors.nersc_jobsdb.NerscJobsDb* method), 73
 query_and_scroll() (*tokio.connectors.es.EsConnection* method), 46
 query_cpu() (*tokio.connectors.collectd_es.CollectdEs* method), 36
 query_disk() (*tokio.connectors.collectd_es.CollectdEs* method), 36
 query_index_db() (in module *tokio.cli.darshan_scoreboard*), 28
 query_memory() (*tokio.connectors.collectd_es.CollectdEs* method), 36
 query_timeseries() (*tokio.connectors.collectd_es.CollectdEs* method), 36
 query_timeseries() (*tokio.connectors.es.EsConnection* method), 46
 query_timeseries() (*tokio.connectors.nersc_globuslogs.NerscGlobusLogs* method), 69
 query_type() (*tokio.connectors.nersc_globuslogs.NerscGlobusLogs* method), 69

query_user() (*tokio.connectors.nersc_globuslogs.NerscGlobusLogs* method), 69

R

rearrange_columns() (*tokio.timeseries.TimeSeries* method), 91

recast_string() (in module *tokio.common*), 88

reduce_dataset_name() (in module *tokio.connectors.hdf5*), 33

reduce_diff() (in module *tokio.cli.compare_isdct*), 27

reset_timeseries() (in module *tokio.cli.archive_collectdes*), 18

retrieve_concurrent_job_data() (in module *tokio.cli.summarize_job*), 29

retrieve_darshan_data() (in module *tokio.cli.summarize_job*), 29

retrieve_jobid() (in module *tokio.cli.summarize_job*), 29

retrieve_lmt_data() (in module *tokio.cli.summarize_job*), 30

retrieve_ost_data() (in module *tokio.cli.summarize_job*), 30

retrieve_tables() (in module *tokio.cli.cache_lmtdb*), 25

retrieve_topology_data() (in module *tokio.cli.summarize_job*), 30

S

save_cache() (*tokio.connectors.cachingdb.CachingDb* method), 35

save_cache() (*tokio.connectors.common.CacheableDict* method), 38

save_cache() (*tokio.connectors.common.SubprocessOutputDict* method), 38

save_cache() (*tokio.connectors.common.SubprocessOutputList* method), 39

save_cache() (*tokio.connectors.es.EsConnection* method), 47

save_cache() (*tokio.connectors.nersc_lfsstate.NerscLfsOstFullness* method), 73

save_cache() (*tokio.connectors.nersc_lfsstate.NerscLfsOstMap* method), 75

scroll() (*tokio.connectors.es.EsConnection* method), 47

serialize_datetime() (in module *tokio.cli.summarize_job*), 30

server_type() (*tokio.cli.archive_mmperfmon.Archiver* method), 23

set_columns() (*tokio.timeseries.TimeSeries* method), 91

set_timeseries_metadata() (*tokio.cli.archive_esnet_snmp.Archiver* method), 19

set_timeseries_metadata() (*tokio.cli.archive_lmtdb.DatasetDict* method), 22

set_timeseries_metadata() (*tokio.cli.archive_mmperfmon.Archiver* method), 24

set_timestamp_key() (*tokio.timeseries.TimeSeries* method), 91

set_version() (*tokio.connectors.hdf5.Hdf5* method), 56

Slurm (class in *tokio.connectors.slurm*), 75

SlurmEncoder (class in *tokio.connectors.slurm*), 77

sort_columns() (*tokio.timeseries.TimeSeries* method), 91

sorted_nodenames() (in module *tokio.timeseries*), 91

SubprocessOutputDict (class in *tokio.connectors.common*), 38

SubprocessOutputList (class in *tokio.connectors.common*), 39

summarize_byterate_df() (in module *tokio.cli.summarize_job*), 30

summarize_columns() (in module *tokio.cli.summarize_tts*), 30

summarize_cpu_df() (in module *tokio.cli.summarize_job*), 30

summarize_darshan() (in module *tokio.cli.summarize_job*), 30

summarize_darshan_perf() (in module *tokio.cli.darshan_bad_ost*), 27

summarize_darshan_posix() (in module *tokio.cli.summarize_job*), 30

summarize_errors() (in module *tokio.cli.compare_isdct*), 27

summarize_mds_ops_df() (in module *tokio.cli.summarize_job*), 30

summarize_missing_df() (in module *tokio.cli.summarize_job*), 30

summarize_reduced_data() (in module *tokio.cli.summarize_h5lmt*), 29

summarize_reduced_diffs() (in module *tokio.cli.compare_isdct*), 27

summarize_timesteps() (in module *tokio.cli.summarize_tts*), 30

summarize_tts_hdf5() (in module *tokio.cli.summarize_tts*), 31

swap_columns() (*tokio.timeseries.TimeSeries* method), 91

T

TimeSeries (class in *tokio.timeseries*), 90

timeseries_deltas() (in module *tokio.timeseries*), 91

- to_dataframe() (*tokio.analysis.umami.Umami method*), 15
 - to_dataframe() (*tokio.connectors.collectdes.Collectdes method*), 37
 - to_dataframe() (*tokio.connectors.es.EsConnection method*), 47
 - to_dataframe() (*tokio.connectors.esnet_snmp.EsnetSnmp method*), 52
 - to_dataframe() (*tokio.connectors.hdf5.Hdf5 method*), 56
 - to_dataframe() (*tokio.connectors.mmpperfmon.Mmperfmon method*), 67
 - to_dataframe() (*tokio.connectors.nersc_globuslogs.NerscGlobusLogs method*), 69
 - to_dataframe() (*tokio.connectors.nersc_isdct.NerscIsdct method*), 70
 - to_dataframe() (*tokio.connectors.slurm.Slurm method*), 77
 - to_dataframe_by_host() (*tokio.connectors.mmpperfmon.Mmperfmon method*), 67
 - to_dataframe_by_metric() (*tokio.connectors.mmpperfmon.Mmperfmon method*), 67
 - to_dict() (*tokio.analysis.umami.Umami method*), 16
 - to_epoch() (*in module tokio.common*), 88
 - to_json() (*tokio.analysis.umami.Umami method*), 16
 - to_json() (*tokio.analysis.umami.UmamiMetric method*), 16
 - to_json() (*tokio.connectors.mmpperfmon.Mmperfmon method*), 67
 - to_json() (*tokio.connectors.slurm.Slurm method*), 77
 - to_timeseries() (*tokio.connectors.hdf5.Hdf5 method*), 56
 - tokio (*module*), 15
 - tokio.analysis (*module*), 15
 - tokio.analysis.umami (*module*), 15
 - tokio.cli (*module*), 16
 - tokio.cli.archive_collectdes (*module*), 16
 - tokio.cli.archive_esnet_snmp (*module*), 19
 - tokio.cli.archive_lmtdb (*module*), 20
 - tokio.cli.archive_mmpperfmon (*module*), 22
 - tokio.cli.cache_collectdes (*module*), 25
 - tokio.cli.cache_darshan (*module*), 25
 - tokio.cli.cache_esnet_snmp (*module*), 25
 - tokio.cli.cache_isdct (*module*), 25
 - tokio.cli.cache_lfsstatus (*module*), 25
 - tokio.cli.cache_lmtdb (*module*), 25
 - tokio.cli.cache_mmpperfmon (*module*), 26
 - tokio.cli.cache_nersc_globuslogs (*module*), 26
 - tokio.cli.cache_nersc_jobsdb (*module*), 26
 - tokio.cli.cache_slurm (*module*), 26
 - tokio.cli.cache_topology (*module*), 26
 - tokio.cli.compare_isdct (*module*), 26
 - tokio.cli.darshan_bad_ost (*module*), 27
 - tokio.cli.darshan_scoreboard (*module*), 28
 - tokio.cli.find_darshanlogs (*module*), 28
 - tokio.cli.summarize_h5lmt (*module*), 28
 - tokio.cli.summarize_job (*module*), 29
 - tokio.cli.summarize_tts (*module*), 30
 - tokio.common (*module*), 87
 - tokio.config (*module*), 89
 - tokio.connectors (*module*), 31
 - tokio.connectors._hdf5 (*module*), 31
 - tokio.connectors.cachingdb (*module*), 33
 - tokio.connectors.collectdes (*module*), 36
 - tokio.connectors.common (*module*), 37
 - tokio.connectors.craysdb (*module*), 40
 - tokio.connectors.darshan (*module*), 40
 - tokio.connectors.es (*module*), 44
 - tokio.connectors.esnet_snmp (*module*), 48
 - tokio.connectors.globuslogs (*module*), 52
 - tokio.connectors.hdf5 (*module*), 54
 - tokio.connectors.hpss (*module*), 57
 - tokio.connectors.lfshealth (*module*), 63
 - tokio.connectors.lmtdb (*module*), 63
 - tokio.connectors.mmpperfmon (*module*), 65
 - tokio.connectors.nersc_globuslogs (*module*), 68
 - tokio.connectors.nersc_isdct (*module*), 69
 - tokio.connectors.nersc_jobsdb (*module*), 72
 - tokio.connectors.nersc_lfsstate (*module*), 73
 - tokio.connectors.slurm (*module*), 75
 - tokio.debug (*module*), 89
 - tokio.timeseries (*module*), 90
 - tokio.tools (*module*), 79
 - tokio.tools.common (*module*), 79
 - tokio.tools.darshan (*module*), 80
 - tokio.tools.hdf5 (*module*), 81
 - tokio.tools.jobinfo (*module*), 83
 - tokio.tools.lfsstatus (*module*), 83
 - tokio.tools.nersc_mmpperfmon (*module*), 86
 - tokio.tools.topology (*module*), 87
 - trim_rows() (*tokio.timeseries.TimeSeries method*), 91
- ## U
- Umami (*class in tokio.analysis.umami*), 15
 - UmamiMetric (*class in tokio.analysis.umami*), 16
 - update_column_map() (*tokio.timeseries.TimeSeries method*), 91
 - update_datasets() (*in module tokio.cli.archive_collectdes*), 18
- ## V
- value_unit_to_bytes() (*in module*)

tokio.connectors.mmpfmon), 68
vprint() (in module *tokio.cli.darshan_scoreboard*), 28

W

walk_file_collection() (in module *tokio.connectors.common*), 39
warning() (in module *tokio.debug*), 89