Miranda Documentation

Release 0.4.0

Trevor James Smith

Mar 30, 2023

CONTENTS

Python utilities for climate data collection, conversion, and management

- Documentation:
- Free Software:

CHAPTER

ONE

FEATURES

Data collection functions for climate and forecast data hosted at:

- ECMWF (ERA5, ERA5-Land, TIGGE)
- ECCC (Canada) (Monthly Climate Summaries, ECCC GEOAPI - In development)
- NCAR (CORDEX-NA on AWS)

Data conversion for Climate and Forecasting (CF) Variable and Metadata compliance:

- ECMWF (ERA5, ERA5-Land, TIGGE - In Development)
- ECCC (Canada) (Flat File Observations, Monthly Climate Summaries, Adjusted and Homogenized Climate Data, ECCC GEOAPI - In Development)
- MELCC (Québec)
- Hydro-Québec (In Development)

Database structuring and facets validation:

- **Simulations:**
  - WCRP (CMIP5, CMIP6, CMIP5-CORDEX, CORDEX-ADJUST, ISIMIP, etc.)
- **Station-Observations:**
  - MELCC (Québec) (Needs mdbtools installed)
  - ECCC (Canada) (In Development)
  - Hydro-Québec (In Development)
- **Gridded-Observations:**
  - NRCAN (Canada) (Future)
  - MELCC (Future)
- **Reanalyses:**
  - ECMWF (ERA5, ERA5-Land, TIGGE)
  - NASA (DayMET, AgMerra/AgCFSR, MERRA2) - In Development
  - NCEP (CFSR/CFSv2) - In Development
  - WFDEI-GEM-CaPa (University of Saskatchewan) - In Development

CHAPTER

TWO

INSTALLATION

miranda can be installed from PyPI:

```
$ pip install miranda
```

Some functionalities require complex-to-install dependencies. In order to gain access to them, we strongly suggest using Anaconda to manage your environment:

```
$ conda env create -f environment.yml
$ conda activate miranda
$ pip install miranda[full]
```

For more information about Anaconda/Miniconda/conda-forge:

- Miniconda: https://docs.conda.io/en/latest/miniconda.html
- conda-forge: https://conda-forge.org/#about

miranda also relies on PyESSV for its climate data controlled vocabulary. This library is optional for users who do not require validation checks, but enabling this feature requires additional installation steps:

```
$ mkdir -p ~/.esdoc
$ git clone git@github.com:ES-DOC/pyessv-archive.git ~/.esdoc/pyessv-archive
```

CHAPTER

THREE

CONTRIBUTING

See the contributing documentation: https://miranda.readthedocs.io/en/latest/contributing.html

3.1 Contents:

3.1.1 Installation

At the command line, enter:

```
$ pip install miranda
```

This is the preferred method to install miranda, as it will always install the most recent stable release.

To make use of remote operations (*miranda.remote*) and some dataset downloading functions (*miranda.ncar miranda.ecmwf*), additional libraries are needed. They can can be installed with the following:

```
$ pip install miranda[full]
```

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for miranda can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/Ouranosinc/miranda
```

Or download the tarball:

```
$ curl -OL https://github.com/Ouranosinc/miranda/tarball/main
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

Alternatively, you can also install a local copy via pip:

```
$ pip install .
```

Creating a Conda environment

To create a conda development environment including all miranda dependencies, enter the following command from within your cloned repo:

```
$ conda create -n my_miranda_env python=3.8 --file=environment.yml
$ conda activate my_miranda_env
$ pip install -e .[dev]
```

3.1.2 Usage

To use Miranda in a project:

```
import miranda
```

3.1.3 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/Ouranosinc/miranda/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

Miranda could always use more documentation, whether as part of the official Miranda docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/Ouranosinc/miranda/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *miranda* for local development.

1. Fork the *miranda* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/miranda.git
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

4. Begin by installing a development build of your branch:

```
# To install miranda with its development environment dependencies
$ pip install -e .[dev]
# To install miranda with its documentation dependencies
$ pip install -e .[docs]
# To install miranda with its remote API dependencies
$ pip install -e .[remote]
```

5. When you're done making changes, check that your changes pass style and unit tests, including testing other Python versions with tox:

```
$ tox
```

To get tox, just pip install it.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7, 3.8, and 3.9. Check https://github.com/Ouranosinc/miranda/actions for active pull request builds or run the `tox` command and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ pytest test/test_miranda.py
```

3.1.4 Credits

Development Lead

- Trevor James Smith <smith.trevorj@ouranos.ca> @Zeitsperre

Co-Developers

- Pascal Bourgault <bourgault.pascal@ouranos.ca> @aulemahal
- Travis Logan <logan.travis@ouranos.ca> @tlogan2000

Contributors

- Sébastien Biner <biner.sebastien@hydroquebec.com> @sbiner
- David Huard <huard.david@ouranos.ca> @huard
- Gabriel Rondeau-Genesse <rondeau-genesse.gabriel@ouranos.ca> @RondeauG

3.1.5 History

v0.4.0 (2023-03-30)

Contributors to this version: Trevor James Smith (@Zeitsperre), Pascal Bourgault (@aulemahal), Travis Logan (@tlogan2000).

New features

- Improvements have been made to the development documentation; Project URLs, ReadTheDocs theming, and other quality of life changes.
- Conversion JSON definitions now support pre-processing to render dimensions and variable names consistent before running corrections/conversions.
- **New datasets with CF-like attributes conversion supported:**
  - RDRS (ECCC)
  - GRNCH (ETS)
- Preliminary `miranda.io` module for organizing output-writing functionality.
- New `miranda.io.fetch_chunk_config` function for "rechunking" datasets according to project presets.
- New `mirands.io.utils.name_output_file` for generating names from Dataset facets or from a dictionary.
- New `mirands.gis.subset_domain` for clipping dataset to a preconfigured region.

Bug fixes

- Many data-related utilities now have more accurate static typing.
- Converted dataset global attributes are now synchronized for consistency.
- ECMWF-based datasets now implement more consistent conversion factors and metadata.
- `miranda.storage.file_size` now handles dictionaries of Pathlib objects.

Internal changes

- Pre-commit version updates.
- Improvements have been made to the development documentation; Project URLs, ReadTheDocs theming, installation methods, and other quality of life changes.
- **Schema and folder structure updates:**
  - *gridded-obs* -> *reconstruction*
  - *bias-adjust-project* is used when present and not just when *level=="biasadjusted"*
- CI now using *tox>=4.0* and *ubuntu-latest* virtual machine images.

v0.3.0 (2022-11-24)

Contributors to this version: Trevor James Smith (@Zeitsperre), Pascal Bourgault (@aulemahal), David Huard (@huard), Travis Logan (@tlogan2000), Gabriel Rondeau-Genesse (@RondeauG), and Sébastien Biner (@sbiner).

Announcements

- First public release on PyPI.

New features

- **Dataset conversion tools (`miranda.convert`) use a JSON-definition file to dynamically populate metadata, run data quality checks, and convert units to CF-compliant standard. Supported datasets are:**
  - ERA5/ERA5-Land (complete)
  - MELCC (stations) (beta)
  - ECCC (stations) (alpha)
  - NASA DayMet (WIP)
  - NASA AgMerra/AgCFSR (WIP)
  - Hydro Québec (stations) (WIP)
  - DEH (stations) (WIP)
  - WFDEI-GEM-CAPA (WIP)
- Module (`miranda.eccc`) for ECCC station data and ECCC Adjusted and Homogenized Canadian Climate Data (AHCCD) conversion (WIP).
- Module (`miranda.ncar`) for fetching interpolated CORDEX-NAM (22i/44i) from NCAR AWS data storage.
- Module (`miranda.ecmwf`) for fetching ECMWF ERA5/-Land (single-levels, pressure-levels, monthly-means) datasets via CDSAPI.
- Module (`miranda.gis`) for setting specific subsetting domains used when converting gridded datasets.
- Modules (`miranda.archive` and `miranda.remote`) for performing data archiving actions locally and remotely (powered by fabric and paramiko) (WIP).
- **Module (`miranda.decode`) for ingesting and parsing dataset metadata based on filename and dataset attributes. Supported datasets are:**
  - *miranda* converted datasets
  - CMIP6
  - CMIP5
  - CMIP5-CORDEX
  - ISIMIP-FT
  - CanDCS-U6 (PCIC)
- Module (`miranda.structure`) for create constructing file-tree databases based on YAML-defined metadata schemas (WIP).
- Modules (`miranda.cv` and `miranda.validators`) for validating metadata using ESGF controlled vocabularies (taken from pyessv-archive) and schema definitions (powered by schema), respectively (WIP).

3.1.6 License

```
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
      and conversions to other media types.

      "Work" shall mean the work of authorship, whether in Source or
      Object form, made available under the License, as indicated by a
      copyright notice that is included in or attached to the work
      (an example is provided in the Appendix below).

      "Derivative Works" shall mean any work, whether in Source or Object
      form, that is based on (or derived from) the Work and for which the
      editorial revisions, annotations, elaborations, or other modifications
      represent, as a whole, an original work of authorship. For the purposes
      of this License, Derivative Works shall not include works that remain
      separable from, or merely link (or bind by name) to the interfaces of,
      the Work and Derivative Works thereof.

      "Contribution" shall mean any work of authorship, including
      the original version of the Work and any modifications or additions
      to that Work or Derivative Works thereof, that is intentionally
      submitted to Licensor for inclusion in the Work by the copyright owner
```

(continues on next page)

(continued from previous page)

```
      or by an individual or Legal Entity authorized to submit on behalf of
      the copyright owner. For the purposes of this definition, "submitted"
      means any form of electronic, verbal, or written communication sent
      to the Licensor or its representatives, including but not limited to
      communication on electronic mailing lists, source code control systems,
      and issue tracking systems that are managed by, or on behalf of, the
      Licensor for the purpose of discussing and improving the Work, but
      excluding communication that is conspicuously marked or otherwise
      designated in writing by the copyright owner as "Not a Contribution."

      "Contributor" shall mean Licensor and any individual or Legal Entity
      on behalf of whom a Contribution has been received by Licensor and
      subsequently incorporated within the Work.

   2. Grant of Copyright License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      copyright license to reproduce, prepare Derivative Works of,
      publicly display, publicly perform, sublicense, and distribute the
      Work and such Derivative Works in Source or Object form.

   3. Grant of Patent License. Subject to the terms and conditions of
      this License, each Contributor hereby grants to You a perpetual,
      worldwide, non-exclusive, no-charge, royalty-free, irrevocable
      (except as stated in this section) patent license to make, have made,
      use, offer to sell, sell, import, and otherwise transfer the Work,
      where such license applies only to those patent claims licensable
      by such Contributor that are necessarily infringed by their
      Contribution(s) alone or by combination of their Contribution(s)
      with the Work to which such Contribution(s) was submitted. If You
      institute patent litigation against any entity (including a
      cross-claim or counterclaim in a lawsuit) alleging that the Work
      or a Contribution incorporated within the Work constitutes direct
      or contributory patent infringement, then any patent licenses
      granted to You under this License for that Work shall terminate
      as of the date such litigation is filed.

   4. Redistribution. You may reproduce and distribute copies of the
      Work or Derivative Works thereof in any medium, with or without
      modifications, and in Source or Object form, provided that You
      meet the following conditions:

      (a) You must give any other recipients of the Work or
          Derivative Works a copy of this License; and

      (b) You must cause any modified files to carry prominent notices
          stating that You changed the files; and

      (c) You must retain, in the Source form of any Derivative Works
          that You distribute, all copyright, patent, trademark, and
          attribution notices from the Source form of the Work,
          excluding those notices that do not pertain to any part of
```

(continues on next page)

(continued from previous page)

```
          the Derivative Works; and

      (d) If the Work includes a "NOTICE" text file as part of its
          distribution, then any Derivative Works that You distribute must
          include a readable copy of the attribution notices contained
          within such NOTICE file, excluding those notices that do not
          pertain to any part of the Derivative Works, in at least one
          of the following places: within a NOTICE text file distributed
          as part of the Derivative Works; within the Source form or
          documentation, if provided along with the Derivative Works; or,
          within a display generated by the Derivative Works, if and
          wherever such third-party notices normally appear. The contents
          of the NOTICE file are for informational purposes only and
          do not modify the License. You may add Your own attribution
          notices within Derivative Works that You distribute, alongside
          or as an addendum to the NOTICE text from the Work, provided
          that such additional attribution notices cannot be construed
          as modifying the License.

      You may add Your own copyright statement to Your modifications and
      may provide additional or different license terms and conditions
      for use, reproduction, or distribution of Your modifications, or
      for any such Derivative Works as a whole, provided Your use,
      reproduction, and distribution of the Work otherwise complies with
      the conditions stated in this License.

   5. Submission of Contributions. Unless You explicitly state otherwise,
      any Contribution intentionally submitted for inclusion in the Work
      by You to the Licensor shall be under the terms and conditions of
      this License, without any additional terms or conditions.
      Notwithstanding the above, nothing herein shall supersede or modify
      the terms of any separate license agreement you may have executed
      with Licensor regarding such Contributions.

   6. Trademarks. This License does not grant permission to use the trade
      names, trademarks, service marks, or product names of the Licensor,
      except as required for reasonable and customary use in describing the
      origin of the Work and reproducing the content of the NOTICE file.

   7. Disclaimer of Warranty. Unless required by applicable law or
      agreed to in writing, Licensor provides the Work (and each
      Contributor provides its Contributions) on an "AS IS" BASIS,
      WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
      implied, including, without limitation, any warranties or conditions
      of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
      PARTICULAR PURPOSE. You are solely responsible for determining the
      appropriateness of using or redistributing the Work and assume any
      risks associated with Your exercise of permissions under this License.

   8. Limitation of Liability. In no event and under no legal theory,
      whether in tort (including negligence), contract, or otherwise,
      unless required by applicable law (such as deliberate and grossly
```

(continues on next page)

(continued from previous page)

```
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

   To apply the Apache License to your work, attach the following
   boilerplate notice, with the fields enclosed by brackets "[]"
   replaced with your own identifying information. (Don't include
   the brackets!)  The text should be enclosed in the appropriate
   comment syntax for the file format. We also recommend that a
   file or class name and description of purpose be included on the
   same "printed page" as the copyright notice for easier
   identification within third-party archives.

Copyright 2019 Trevor James Smith, Ouranos Inc., and contributors

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

3.1.7 miranda

miranda package

class miranda.**DataBase**(*source*, ***, *destination: str | Path | None = None*, *common_path: str | Path | None = None*, *file_pattern: str | List[str] = '*.nc'*, *project_name: str | None = None*, *recursive: bool = True*)

Bases: `object`

archive()

group_by(***, *common_path: Path | str | None = None*, *subdirectories: bool = True*, *dates: bool = True*, *size: int = 10737418240*)

items()

keys()

target(*target: Path | str*)

transfer()

values()

class miranda.**FileMeta**(*path: str*, *size: int = -1*)

Bases: `object`

File path and size.

django = {'path': ['CharField', 'max_length=512'], 'size': ['IntegerField', 'null=True', 'blank=True']}

class miranda.**StorageState**(*base_path*, *capacity=-1*, *used_space=-1*, *free_space=-1*)

Bases: `object`

Information regarding the storage capacity of a disk.

Subpackages

miranda.archive package

miranda.archive.**group_by_deciphered_date**(*files: generator | List[str | Path]*) → Dict[str, List[*Path*]]

Find a common date and groups files based on year and month.

Parameters
files (*Union[GeneratorType, List[Union[str, Path]]]*)

Returns
Dict[str, List[Path]]

miranda.archive.**group_by_length**(*files: generator | List[str | Path], size: int = 10, sort: bool = False*) → List[List[*Path*]]

Group files by an arbitrary number of file entries.

Parameters

- **files** (*Union[GeneratorType, List[Union[str, Path]]]*)
- **size** (*int*)
- **sort** (*bool*)

Returns
List[List[Path]]

miranda.archive.**group_by_size**(*files: generator | List[str | Path], size: int = 10737418240*) → List[List[*Path*]]

Group files up until a desired size and save it as a grouping within a list.

Parameters

- **files** (*Union[GeneratorType, List[Union[str, Path]]]*)
- **size** (*int*)

Returns
List[List[Path]]

miranda.archive.**group_by_subdirectories**(*files: generator | List[str | Path], within: Path | str | None = None*) → Dict[str, List[*Path*]]

This function will group files based on the parent folder that they are located within.

Parameters

- **files** (*Union[GeneratorType, List[Union[str, Path]]]*)
- **within** (*Union[str, Path]*)

Returns
Dict[str, List[Path]]

miranda.archive.**select_by_date_modified**(*source: Path | str, year: int, month: int, day: int, pattern: str | None = None, datetime: date | None = None*) → List

Parameters

- **source**
- **year**
- **month**
- **day**
- **pattern**
- **datetime**

Submodules

miranda.archive._groupings module

miranda.archive._groupings.**group_by_deciphered_date**(*files: generator | List[str | Path]*) → Dict[str, List[*Path*]]

Find a common date and groups files based on year and month.

Parameters
files (*Union[GeneratorType, List[Union[str, Path]]]*)

Returns
Dict[str, List[Path]]

miranda.archive._groupings.**group_by_length**(*files: generator | List[str | Path]*, *size: int = 10*, *sort: bool = False*) → List[List[*Path*]]

Group files by an arbitrary number of file entries.

Parameters

- **files** (*Union[GeneratorType, List[Union[str, Path]]]*)
- **size** (*int*)
- **sort** (*bool*)

Returns
List[List[Path]]

miranda.archive._groupings.**group_by_size**(*files: generator | List[str | Path]*, *size: int = 10737418240*) → List[List[*Path*]]

Group files up until a desired size and save it as a grouping within a list.

Parameters

- **files** (*Union[GeneratorType, List[Union[str, Path]]]*)
- **size** (*int*)

Returns
List[List[Path]]

miranda.archive._groupings.**group_by_subdirectories**(*files: generator | List[str | Path]*, *within: Path | str | None = None*) → Dict[str, List[*Path*]]

This function will group files based on the parent folder that they are located within.

Parameters

- **files** (*Union[GeneratorType, List[Union[str, Path]]]*)
- **within** (*Union[str, Path]*)

Returns
Dict[str, List[Path]]

miranda.archive._selection module

miranda.archive._selection.**select_by_date_modified**(*source: Path | str*, *year: int*, *month: int*, *day: int*, *pattern: str | None = None*, *datetime: date | None = None*) → List

Parameters

- **source**
- **year**
- **month**
- **day**
- **pattern**
- **datetime**

miranda.convert package

miranda.convert.**aggregate**(*ds*, *freq: str = 'day'*) → Dict[str, Dataset]

miranda.convert.**aggregations_possible**(*ds: Dataset*, *freq: str = 'day'*) → Dict[str, Set[str]]

miranda.convert.**dataset_conversion**(*input_files: str | PathLike | Sequence[str | PathLike] | Iterator[PathLike] | Dataset*, *project: str*, *domain: str | None = None*, *mask: Dataset | DataArray | None = None*, *mask_cutoff: float = 0.5*, *add_version_hashes: bool = True*, *preprocess: Callable | str | None = 'auto'*, ***xr_kwargs*) → Dataset | DataArray

Convert an existing Xarray-compatible dataset to another format with variable corrections applied.

Parameters

- **input_files** (*str or os.PathLike or Sequence[str or os.PathLike] or Iterator[os.PathLike] or xr.Dataset*) – Files or objects to be converted. If sent a list or GeneratorType, will open with `xarray.open_mfdataset()` and concatenate files.
- **project** (*{"cordex", "cmip5", "cmip6", "ets-grnch", "isimip-ft", "pcic-candcs-u6", "converted"}*) – Project name for decoding/handling purposes.
- **domain** (*{"global", "nam", "can", "qc", "mtl"}, optional*) – Domain to perform subsetting for. Default: None.
- **mask** (*Optional[Union[xr.Dataset, xr.DataArray]]*) – DataArray or single data_variable dataset containing mask.
- **mask_cutoff** (*float*) – If land_sea_mask supplied, the threshold above which to mask with land_sea_mask. Default: 0.5.
- **add_version_hashes** (*bool*) – If True, version name and sha256sum of source file(s) will be added as a field among the global attributes.
- **preprocess** (*callable or str, optional*) – Preprocessing functions to perform over each Dataset. Default: "auto" - Run preprocessing fixes based on supplied fields from metadata definition. Callable - Runs function over Dataset (single) or supplied to *preprocess* (multifile dataset).
- ****xr_kwargs** – Arguments passed directly to xarray.

Returns
xr.Dataset or xr.DataArray

miranda.convert.**dataset_corrections**(*ds: Dataset*, *project: str*) → Dataset

Convert variables to CF-compliant format

miranda.convert.**dims_conversion**(*d: Dataset*, *p: str*, *m: dict*) → Dataset

miranda.convert.**gather_agcfsr**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather agCFSR source data.

Parameters
path (*str or os.PathLike*)

Returns
dict(str, list[pathlib.Path])

miranda.convert.**gather_agmerra**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather agMERRA source data.

Parameters
path (*str or os.PathLike*)

Returns
dict(str, list[pathlib.Path])

miranda.convert.**gather_ecmwf**(*project: str*, *path: str | PathLike*, *back_extension: bool = False*, *monthly_means: bool = False*) → Dict[str, List[*Path*]]

Parameters

- **project** (*{"era5-single-levels", "era5-pressure-levels", "era5-land"}*)
- **path** (*str or os.PathLike*)
- **back_extension** (*bool*)
- **monthly_means** (*bool*)

Returns
dict(str, list[pathlib.Path])

miranda.convert.**gather_grnch**(*path: str | PathLike*) → Dict[str, List[*Path*]]

miranda.convert.**gather_nrcan_gridded_obs**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather NRCan Gridded Observations source data.

Parameters
path (*str or os.PathLike*)

Returns
dict(str, list[pathlib.Path])

miranda.convert.**gather_raw_rdrs_by_years**(*path: str | PathLike*) → Dict[str, Dict[str, List[*Path*]]]

Gather raw RDRS files for preprocessing.

Parameters
path (*str or os.PathLike*)

Returns
dict(str, dict(str, list[Path])) or None

miranda.convert.**gather_rdrs**(*name: str*, *path: str | PathLike*, *suffix: str*, *key: str*) → Dict[str, Dict[str, List[*Path*]]]

Gather RDRS processed source data.

Parameters

- **name** (*str*)
- **path** (*str or os.PathLike*)
- **suffix** (*str*)
- **key** (*str one of 'raw' or 'cf' indicating which variable name dictionary to search for*)

Returns
dict(str, list[pathlib.Path])

miranda.convert.**gather_sc_earth**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather SC-Earth source data

Parameters
path (*str or os.PathLike*)

Returns
dict(str, list[pathlib.Path])

miranda.convert.**gather_wfdei_gem_capa**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather WFDEI-GEM-CaPa source data.

Parameters
path (*str or os.PathLike*)

Returns
dict(str, list[pathlib.Path])

miranda.convert.**load_json_data_mappings**(*project: str*) → dict

miranda.convert.**metadata_conversion**(*d: Dataset*, *p: str*, *m: Dict*) → Dataset

Update xarray dataset and data_vars with project-specific metadata fields.

Parameters

- **d** (*xarray.Dataset*) – Dataset with metadata to be updated.
- **p** (*str*) – Dataset project name.
- **m** (*dict*) – Metadata definition dictionary for project and variable(s).

Returns
xarray.Dataset

miranda.convert.**threshold_mask**(*ds: Dataset | DataArray*, ***, *mask: Dataset | DataArray*, *mask_cutoff: float | bool = False*) → Dataset | DataArray

Land-Sea mask operations.

Parameters

- **ds** (*Union[xr.Dataset, str, os.PathLike]*)
- **mask** (*Union[xr.Dataset, xr.DataArray]*)
- **mask_cutoff** (*float or bool*)

Returns
Union[xr.Dataset, xr.DataArray]

miranda.convert.**variable_conversion**(*d: Dataset*, *p: str*, *m: dict*) → Dataset

Submodules

miranda.convert._aggregation module

miranda.convert._aggregation.**aggregate**(*ds*, *freq: str = 'day'*) → Dict[str, Dataset]

miranda.convert._aggregation.**aggregations_possible**(*ds: Dataset*, *freq: str = 'day'*) → Dict[str, Set[str]]

miranda.convert._data_corrections module

miranda.convert._data_corrections.**dataset_conversion**(*input_files: str | PathLike | Sequence[str | PathLike] | Iterator[PathLike] | Dataset*, *project: str*, *domain: str | None = None*, *mask: Dataset | DataArray | None = None*, *mask_cutoff: float = 0.5*, *add_version_hashes: bool = True*, *preprocess: Callable | str | None = 'auto'*, ***xr_kwargs*) → Dataset | DataArray

Convert an existing Xarray-compatible dataset to another format with variable corrections applied.

Parameters

- **input_files** (*str or os.PathLike or Sequence[str or os.PathLike] or Iterator[os.PathLike] or xr.Dataset*) – Files or objects to be converted. If sent a list or GeneratorType, will open with `xarray.open_mfdataset()` and concatenate files.
- **project** (*{"cordex", "cmip5", "cmip6", "ets-grnch", "isimip-ft", "pcic-candcs-u6", "converted"}*) – Project name for decoding/handling purposes.
- **domain** (*{"global", "nam", "can", "qc", "mtl"}, optional*) – Domain to perform subsetting for. Default: None.
- **mask** (*Optional[Union[xr.Dataset, xr.DataArray]]*) – DataArray or single data_variable dataset containing mask.
- **mask_cutoff** (*float*) – If land_sea_mask supplied, the threshold above which to mask with land_sea_mask. Default: 0.5.
- **add_version_hashes** (*bool*) – If True, version name and sha256sum of source file(s) will be added as a field among the global attributes.
- **preprocess** (*callable or str, optional*) – Preprocessing functions to perform over each Dataset. Default: "auto" - Run preprocessing fixes based on supplied fields from metadata definition. Callable - Runs function over Dataset (single) or supplied to *preprocess* (multifile dataset).
- ****xr_kwargs** – Arguments passed directly to xarray.

Returns

xr.Dataset or xr.DataArray

miranda.convert._data_corrections.**dataset_corrections**(*ds: Dataset*, *project: str*) → Dataset

Convert variables to CF-compliant format

miranda.convert._data_corrections.**dims_conversion**(*d: Dataset*, *p: str*, *m: dict*) → Dataset

miranda.convert._data_corrections.**load_json_data_mappings**(*project: str*) → dict

miranda.convert._data_corrections.**metadata_conversion**(*d: Dataset*, *p: str*, *m: Dict*) → Dataset

Update xarray dataset and data_vars with project-specific metadata fields.

Parameters

- **d** (*xarray.Dataset*) – Dataset with metadata to be updated.
- **p** (*str*) – Dataset project name.
- **m** (*dict*) – Metadata definition dictionary for project and variable(s).

Returns

xarray.Dataset

miranda.convert._data_corrections.**threshold_mask**(*ds: Dataset | DataArray*, ***, *mask: Dataset | DataArray*, *mask_cutoff: float | bool = False*) → Dataset | DataArray

Land-Sea mask operations.

Parameters

- **ds** (*Union[xr.Dataset, str, os.PathLike]*)
- **mask** (*Union[xr.Dataset, xr.DataArray]*)
- **mask_cutoff** (*float or bool*)

Returns

Union[xr.Dataset, xr.DataArray]

miranda.convert._data_corrections.**variable_conversion**(*d: Dataset*, *p: str*, *m: dict*) → Dataset

miranda.convert._data_definitions module

miranda.convert._data_definitions.**gather_agcfsr**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather agCFSR source data.

Parameters

path (*str or os.PathLike*)

Returns

dict(str, list[pathlib.Path])

miranda.convert._data_definitions.**gather_agmerra**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather agMERRA source data.

Parameters

path (*str or os.PathLike*)

Returns

dict(str, list[pathlib.Path])

miranda.convert._data_definitions.**gather_ecmwf**(*project: str*, *path: str | PathLike*, *back_extension: bool = False*, *monthly_means: bool = False*) → Dict[str, List[*Path*]]

Parameters

- **project** (*{"era5-single-levels", "era5-pressure-levels", "era5-land"}*)

- **path** (*str or os.PathLike*)
- **back_extension** (*bool*)
- **monthly_means** (*bool*)

Returns
dict(str, list[pathlib.Path])

miranda.convert._data_definitions.**gather_grnch**(*path: str | PathLike*) → Dict[str, List[*Path*]]

miranda.convert._data_definitions.**gather_nrcan_gridded_obs**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather NRCan Gridded Observations source data.

Parameters
path (*str or os.PathLike*)

Returns
dict(str, list[pathlib.Path])

miranda.convert._data_definitions.**gather_raw_rdrs_by_years**(*path: str | PathLike*) → Dict[str, Dict[str, List[*Path*]]]

Gather raw RDRS files for preprocessing.

Parameters
path (*str or os.PathLike*)

Returns
dict(str, dict(str, list[Path])) or None

miranda.convert._data_definitions.**gather_rdrs**(*name: str*, *path: str | PathLike*, *suffix: str*, *key: str*) → Dict[str, Dict[str, List[*Path*]]]

Gather RDRS processed source data.

Parameters

- **name** (*str*)
- **path** (*str or os.PathLike*)
- **suffix** (*str*)
- **key** (*str one of 'raw' or 'cf' indicating which variable name dictionary to search for*)

Returns
dict(str, list[pathlib.Path])

miranda.convert._data_definitions.**gather_sc_earth**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather SC-Earth source data

Parameters
path (*str or os.PathLike*)

Returns
dict(str, list[pathlib.Path])

miranda.convert._data_definitions.**gather_wfdei_gem_capa**(*path: str | PathLike*) → Dict[str, List[*Path*]]

Gather WFDEI-GEM-CaPa source data.

Parameters
path (*str or os.PathLike*)

Returns
dict(str, list[pathlib.Path])

miranda.convert._reconstruction module

miranda.convert._reconstruction.**reanalysis_processing**(*data: Dict[str, List[str | PathLike]]*, *output_folder: str | PathLike*, *variables: Sequence[str]*, *aggregate: str | bool = False*, *domains: str | List[str] = '_DEFAULT'*, *start: str | None = None*, *end: str | None = None*, *target_chunks: dict | None = None*, *output_format: str = 'netcdf'*, *overwrite: bool = False*, *engine: str = 'h5netcdf'*, *n_workers: int = 4*, ***dask_kwargs*) → None

Parameters

- **data** (*Dict[str, List[str]]*)
- **output_folder** (*Union[str, os.PathLike]*)
- **variables** (*Sequence[str]*)
- **aggregate** (*{"day", None}*)
- **domains** (*{"QC", "CAN", "AMNO", "NAM", "GLOBAL"}*)
- **start** (*str, optional*)
- **end** (*str, optional*)
- **target_chunks** (*dict, optional*)
- **output_format** (*{"netcdf", "zarr"}*)
- **overwrite** (*bool*)
- **engine** (*{"netcdf4", "h5netcdf"}*)
- **n_workers** (*int*)

Returns
None

miranda.convert.deh module

miranda.convert.deh.**open_txt**(*path: str | Path*, *cf_table: dict | None = {'flag': {'comment': 'See DEH technical information for details.', 'long_name': 'data flag'}, 'q': {'long_name': 'River discharge', 'units': 'm3 s-1'}}*) → Dataset

Extract daily HQ meteorological data and convert to xr.DataArray with CF-Convention attributes.

miranda.convert.eccc module

miranda.convert.eccc.**convert_canswe**(*file: str | Path*, *output: str | Path*)

Convert the CanSWE netCDF files to production-ready CF-compliant netCDFs.

miranda.convert.eccc_rdrs module

miranda.convert.eccc_rdrs.**convert_rdrs**(*project: str*, *input_folder: str | PathLike*, *output_folder: str | PathLike*, *output_format: str = 'zarr'*, *working_folder: str | PathLike | None = None*, *overwrite: bool = False*, ***dask_kwargs*)

Parameters

- **project**
- **input_folder**
- **output_folder**
- **output_format**
- **working_folder**
- **overwrite**
- **dask_kwargs**

miranda.convert.eccc_rdrs.**rdrs_to_daily**(*project: str*, *input_folder: str | PathLike*, *output_folder: str | PathLike*, *working_folder: str | PathLike | None = None*, *overwrite: bool = False*, *output_format: str = 'zarr'*, *year_start: int | None = None*, *year_end: int | None = None*, *process_variables: list | None = None*, ***dask_kwargs*)

Parameters

- **project**
- **input_folder**
- **output_folder**
- **working_folder**
- **overwrite**
- **output_format**
- **year_start**
- **year_end**
- **process_variables**
- **dask_kwargs**

miranda.convert.ecmwf module

miranda.convert.ecmwf.**tigge_convert**(*source: PathLike | None = None*, *target: PathLike | None = None*, *processes: int = 8*) → None

Convert grib2 file to netCDF format.

Parameters

- **source** (*os.PathLike, optional*)
- **target** (*os.PathLike, optional*)
- **processes** (*int*)

Returns

None

miranda.convert.hq module

miranda.convert.hq.**open_csv**(*path: str | Path*, *cf_table: dict | None = {'hurs': {'cell_methods': 'time: point', 'comment': 'The relative humidity with respect to liquid water for T> 0 C, and with respect to ice for T<0 C.', 'frequency': '1h', 'long_name': 'Near-Surface Relative Humidity', 'out_name': 'hurs', 'standard_name': 'relative_humidity', 'type': 'real', 'units': '%'}, 'prlp': {'cell_methods': 'time: mean', 'comment': 'At surface; includes precipitation of all forms of water in the liquid phase.', 'frequency': 'day', 'long_name': 'Rainfall Flux', 'out_name': 'prlp', 'standard_name': 'rainfall_flux', 'type': 'real', 'units': 'kg m-2 s-1'}, 'prsn': {'cell_methods': 'time: mean', 'comment': 'At surface; includes precipitation of all forms of water in the solid phase.', 'frequency': 'day', 'long_name': 'Snowfall Flux', 'out_name': 'prsn', 'standard_name': 'snowfall_flux', 'type': 'real', 'units': 'kg m-2 s-1'}, 'sfcWind': {'cell_methods': 'time: point', 'comment': 'Near-surface (usually, 10 meters) wind speed.', 'frequency': '1h', 'long_name': 'Near-Surface Wind Speed', 'out_name': 'sfcWind', 'standard_name': 'wind_speed', 'type': 'real', 'units': 'm s-1'}, 'sfcWindAz': {'cell_methods': 'time: point', 'comment': 'Near-surface (usually, 10 meters) direction from which wind originates.', 'frequency': '1h', 'long_name': 'Near-Surface Wind Direction', 'out_name': 'sfcWindAz', 'standard_name': 'wind_direction', 'type': 'real', 'units': 'degree'}, 'snd': {'cell_methods': 'time: point', 'comment': 'The thickness of snow.', 'frequency': '1h', 'long_name': 'Snow Depth', 'out_name': 'snd', 'standard_name': 'surface_snow_thickness', 'type': 'real', 'units': 'm'}, 'tasmax_1h': {'cell_methods': 'time: maximum', 'comment': 'Maximum near-surface (usually, 2 meter) air temperature.', 'frequency': '1h', 'long_name': 'Hourly Maximum Near-Surface Air Temperature', 'out_name': 'tasmax', 'standard_name': 'air_temperature', 'type': 'real', 'units': 'K'}, 'tasmax_day': {'cell_methods': 'time: maximum', 'comment': 'Maximum near-surface (usually, 2 meter) air temperature.', 'frequency': 'day', 'long_name': 'Daily Maximum Near-Surface Air Temperature', 'out_name': 'tasmax', 'standard_name': 'air_temperature', 'type': 'real', 'units': 'K'}, 'tasmin_1h': {'cell_methods': 'time: minimum', 'comment': 'Minimum near-surface (usually, 2 meter) air temperature.', 'frequency': '1h', 'long_name': 'Hourly Minimum Near-Surface Air Temperature', 'out_name': 'tasmin', 'standard_name': 'air_temperature', 'type': 'real', 'units': 'K'}, 'tasmin_day': {'cell_methods': 'time: minimum', 'comment': 'Minimum near-surface (usually, 2 meter) air temperature.', 'frequency': 'day', 'long_name': 'Daily Minimum Near-Surface Air Temperature', 'out_name': 'tasmin', 'standard_name': 'air_temperature', 'type': 'real', 'units': 'K'}}*) → DataArray

Extract daily HQ meteo data and convert to xr.DataArray with CF-Convention attributes.

miranda.convert.melcc module

miranda.convert.melcc.**concat**(*files: Sequence[str | Path]*, *output_folder: str | Path*, *overwrite: bool = True*)

miranda.convert.melcc.**convert_mdb**(*database: str | Path*, *stations: Dataset*, *definitions: Dataset*, *output: str | Path*, *overwrite: bool = True*)

miranda.convert.melcc.**convert_melcc_obs**(*metafile: str | Path*, *folder: str | Path*, *output: str | Path | None = None*, *overwrite: bool = True*)

miranda.convert.melcc.**convert_snow_table**(*file: str | Path*, *output: str | Path*)

Convert snow data given through an Excel file.

This private data is not included in the MDB files.

Parameters

- **file** (*path*) – The excel file with sheets: “Stations”, “Périodes standards” and “Données”
- **output** (*path*) – Folder where to put the netCDF files (one for each of snd, sd and snw).

miranda.convert.melcc.**list_tables**(*dbfile*)

List the tables of a MDB file.

miranda.convert.melcc.**parse_var_code**(*vcode*)

miranda.convert.melcc.**read_definitions**(*dbfile*)

miranda.convert.melcc.**read_stations**(*dbfile*)

miranda.convert.melcc.**read_table**(*dbfile*, *table*)

miranda.convert.utils module

miranda.convert.utils.**date_parser**(*date: str*, ***, *end_of_period: bool = False*, *output_type: str = 'str'*, *strftime_format: str = '%Y-%m-%d'*) → str | Timestamp | NaTType

Parses datetime objects from a string representation of a date or both a start and end date.

Parameters

- **date** (*str*) – Date to be converted.
- **end_of_period** (*bool*) – If True, the date will be the end of month or year depending on what’s most appropriate.
- **output_type** (*{“datetime”, “str”}*) – Desired returned object type.
- **strftime_format** (*str*) – If output_type==’str’, this sets the strftime format.

Returns

pd.Timestamp or str or pd.NaT – Parsed date.

Notes

Adapted from code written by Gabriel Rondeau-Genesse (@RondeauG)

miranda.convert.utils.**find_version_hash**(*file: PathLike | str*) → Dict

Parameters
file (*Path or str*)

Returns
dict

miranda.decode package

class miranda.decode.**Decoder**(*project: str | None*)

Bases: object

decode(*files: PathLike | str | List[str | PathLike] | generator*, *chunks: int | None = None*, *raise_error: bool = False*) → None

Decode facets from file or list of files.

Parameters

- **files** (*Union[str, Path, List[Union[str, Path]]]*)
- **chunks** (*int, optional*)
- **raise_error** (*bool*)

static decode_ahccd_obs(*self*, *file: PathLike | str*) → Dict

classmethod decode_cmip5(*file: PathLike | str*) → dict

classmethod decode_cmip6(*file: PathLike | str*) → dict

classmethod decode_converted(*file: PathLike | str*) → dict

classmethod decode_cordex(*file: PathLike | str*) → dict

static decode_eccc_obs(*self*, *file: PathLike | str*) → Dict

classmethod decode_isimip_ft(*file: PathLike | str*) → dict

static decode_melcc_obs(*self*, *file: PathLike | str*) → Dict

classmethod decode_pcic_candcs_u6(*file: PathLike | str*) → dict

facets_table()

file_facets() → Dict[PathLike, Dict]

guess = False

project = None

exception miranda.decode.**DecoderError**

Bases: Exception

miranda.decode.**guess_project**(*file: PathLike | str*) → str

Guess the name of the project

Parameters
file (*str or os.PathLike*)

Returns
str

Submodules

miranda.decode._decoder module

class miranda.decode._decoder.**Decoder**(*project: str | None*)

Bases: object

decode(*files: PathLike | str | List[str | PathLike] | generator*, *chunks: int | None = None*, *raise_error: bool = False*) → None

Decode facets from file or list of files.

Parameters

- **files** (*Union[str, Path, List[Union[str, Path]]]*)
- **chunks** (*int, optional*)
- **raise_error** (*bool*)

static decode_ahccd_obs(*self*, *file: PathLike | str*) → Dict

classmethod decode_cmip5(*file: PathLike | str*) → dict

classmethod decode_cmip6(*file: PathLike | str*) → dict

classmethod decode_converted(*file: PathLike | str*) → dict

classmethod decode_cordex(*file: PathLike | str*) → dict

static decode_eccc_obs(*self*, *file: PathLike | str*) → Dict

classmethod decode_isimip_ft(*file: PathLike | str*) → dict

static decode_melcc_obs(*self*, *file: PathLike | str*) → Dict

classmethod decode_pcic_candcs_u6(*file: PathLike | str*) → dict

facets_table()

file_facets() → Dict[PathLike, Dict]

guess = False

project = None

miranda.decode._decoder.**guess_project**(*file: PathLike | str*) → str

Guess the name of the project

Parameters
file (*str or os.PathLike*)

Returns
str

miranda.decode._time module

exception miranda.decode._time.**DecoderError**
Bases: Exception

miranda.eccc package

miranda.eccc.**aggregate_stations**(*source_files: str | PathLike | None = None*, *output_folder: str | PathLike | None = None*, *time_step: str | None = None*, *variables: str | int | List[int | str] | None = None*, *include_flags: bool = True*, *groupings: int | None = None*, *mf_dataset_freq: str | None = None*, *temp_directory: str | PathLike | None = None*, *n_workers: int = 1*) → None

Parameters

- **source_files** (*Union[str, Path]*)
- **output_folder** (*Union[str, Path]*)
- **variables** (*Optional[Union[str, int, List[Union[str, int]]]]*)
- **time_step** (*{"hourly", "daily"}*)
- **include_flags** (*bool*)
- **groupings** (*int*) – The number of files in each group used for converting to multi-file Datasets.
- **mf_dataset_freq** (*Optional[str]*) – Resampling frequency for creating output multi-file Datasets. E.g. 'YS': 1 year per file, '5YS': 5 years per file.
- **temp_directory** (*Optional[Union[str, Path]]*) – Use another temporary directory location in case default location is not spacious enough.
- **n_workers** (*int*)

Returns
None

miranda.eccc.**convert_ahccd**(*data_source: str | Path*, *output_dir: str | Path*, *variable: str*, *generation: int | None = None*)

miranda.eccc.**convert_ahccd_fwf_files**(*ff*, *metadata*, *variable*, *generation: int | None = None*, *cols_specs: List[Tuple[int, int]] | None = None*, *attrs: dict | None = None*) → Dataset

miranda.eccc.**convert_flat_files**(*source_files: str | PathLike*, *output_folder: str | PathLike | List[int | str]*, *variables: str | int | List[int | str]*, *mode: str = 'hourly'*, *n_workers: int = 4*) → None

Parameters

- **source_files** (*str or Path*)
- **output_folder** (*str or Path*)
- **variables** (*str or List[str]*)

- **mode** (*{"hourly", "daily"}*)
- **n_workers** (*int*)

Returns
None

miranda.eccc.**daily_summaries_to_netcdf**(*station: dict*, *path_output:* Path | *str*) → None

Parameters

- **station** (*dict*) – dict created by using find_and_extract_dly
- **path_output** (*Union[Path, str]*)

Returns
None

miranda.eccc.**extract_daily_summaries**(*path_station:* Path | *str*, *rm_flags: bool = False*, *file_suffix: str = '.csv'*) → dict

Parameters

- **path_station** (*Union[Path, str]*) – PathLike or str to the station's folder containing the csv files.
- **rm_flags** (*bool*) – Removes the 'Flag' and 'Quality' columns of the ECCC files.
- **file_suffix** (*str*) – File suffixes used by the tabular data. Default: ".csv".

Returns
dict – dict containing the station metadata, as well as the data stored within a pandas Dataframe.

miranda.eccc.**merge_converted_variables**(*source_files: str | PathLike*, *output_folder: str | PathLike*, *variables: str | int | List[int | str] | None = None*, *station_metadata: str | PathLike | None = None*, *overwrite: bool = False*, *n_workers: int = 1*) → None

Parameters

- **source_files** (*Union[str, Path]*)
- **output_folder** (*Union[str, Path]*)
- **variables** (*Optional[Union[str, int, List[Union[str, int]]]]*)
- **station_metadata** (*Optional[Union[str, Path]]*)
- **overwrite** (*bool*)
- **n_workers** (*int*)

Returns
None

Submodules

miranda.eccc._homogenized module

miranda.eccc._homogenized.**convert_ahccd**(*data_source: str | Path*, *output_dir: str | Path*, *variable: str*, *generation: int | None = None*)

miranda.eccc._homogenized.**convert_ahccd_fwf_files**(*ff*, *metadata*, *variable*, *generation: int | None = None*, *cols_specs: List[Tuple[int, int]] | None = None*, *attrs: dict | None = None*) → Dataset

miranda.eccc._raw module

miranda.eccc._raw.**aggregate_stations**(*source_files: str | PathLike | None = None*, *output_folder: str | PathLike | None = None*, *time_step: str | None = None*, *variables: str | int | List[int | str] | None = None*, *include_flags: bool = True*, *groupings: int | None = None*, *mf_dataset_freq: str | None = None*, *temp_directory: str | PathLike | None = None*, *n_workers: int = 1*) → None

Parameters

- **source_files** (*Union[str, Path]*)
- **output_folder** (*Union[str, Path]*)
- **variables** (*Optional[Union[str, int, List[Union[str, int]]]]*)
- **time_step** (*{"hourly", "daily"}*)
- **include_flags** (*bool*)
- **groupings** (*int*) – The number of files in each group used for converting to multi-file Datasets.
- **mf_dataset_freq** (*Optional[str]*) – Resampling frequency for creating output multi-file Datasets. E.g. 'YS': 1 year per file, '5YS': 5 years per file.
- **temp_directory** (*Optional[Union[str, Path]]*) – Use another temporary directory location in case default location is not spacious enough.
- **n_workers** (*int*)

Returns

None

miranda.eccc._raw.**convert_flat_files**(*source_files: str | PathLike*, *output_folder: str | PathLike | List[int | str]*, *variables: str | int | List[int | str]*, *mode: str = 'hourly'*, *n_workers: int = 4*) → None

Parameters

- **source_files** (*str or Path*)
- **output_folder** (*str or Path*)
- **variables** (*str or List[str]*)
- **mode** (*{"hourly", "daily"}*)
- **n_workers** (*int*)

Returns
None

miranda.eccc._raw.**merge_converted_variables**(*source_files: str | PathLike, output_folder: str | PathLike, variables: str | int | List[int | str] | None = None, station_metadata: str | PathLike | None = None, overwrite: bool = False, n_workers: int = 1*) → None

Parameters

- **source_files** (*Union[str, Path]*)
- **output_folder** (*Union[str, Path]*)
- **variables** (*Optional[Union[str, int, List[Union[str, int]]]]*)
- **station_metadata** (*Optional[Union[str, Path]]*)
- **overwrite** (*bool*)
- **n_workers** (*int*)

Returns
None

miranda.eccc._summaries module

miranda.eccc._summaries.**daily_summaries_to_netcdf**(*station: dict, path_output: Path | str*) → None

Parameters

- **station** (*dict*) – dict created by using find_and_extract_dly
- **path_output** (*Union[Path, str]*)

Returns
None

miranda.eccc._summaries.**extract_daily_summaries**(*path_station: Path | str, rm_flags: bool = False, file_suffix: str = '.csv'*) → dict

Parameters

- **path_station** (*Union[Path, str]*) – PathLike or str to the station's folder containing the csv files.
- **rm_flags** (*bool*) – Removes the 'Flag' and 'Quality' columns of the ECCC files.
- **file_suffix** (*str*) – File suffixes used by the tabular data. Default: ".csv".

Returns
dict – dict containing the station metadata, as well as the data stored within a pandas Dataframe.

miranda.eccc._support_rvt module

class miranda.eccc._support_rvt.**Path**(**args*, ***kwargs*)

Bases: PurePath

PurePath subclass that can make system calls.

Path represents a filesystem path but unlike PurePath, also offers methods to do system calls on path objects. Depending on your system, instantiating a Path will return either a PosixPath or a WindowsPath object. You can also instantiate a PosixPath or WindowsPath directly, but cannot instantiate a WindowsPath on a POSIX system or vice versa.

absolute()

Return an absolute version of this path. This function works even if the path doesn't point to anything.

No normalization is done, i.e. all '.' and '..' will be kept along. Use resolve() to get the canonical path to a file.

chmod(*mode*, *, *follow_symlinks=True*)

Change the permissions of the path, like os.chmod().

classmethod cwd()

Return a new path pointing to the current working directory (as returned by os.getcwd()).

exists()

Whether this path exists.

expanduser()

Return a new path with expanded ~ and ~user constructs (as returned by os.path.expanduser)

glob(*pattern*)

Iterate over this subtree and yield all existing files (of any kind, including directories) matching the given relative pattern.

group()

Return the group name of the file gid.

hardlink_to(*target*)

Make this path a hard link pointing to the same file as *target*.

Note the order of arguments (self, target) is the reverse of os.link's.

classmethod home()

Return a new path pointing to the user's home directory (as returned by os.path.expanduser('~')).

is_block_device()

Whether this path is a block device.

is_char_device()

Whether this path is a character device.

is_dir()

Whether this path is a directory.

is_fifo()

Whether this path is a FIFO.

is_file()

Whether this path is a regular file (also True for symlinks pointing to regular files).

is_mount()

Check if this path is a POSIX mount point

is_socket()

Whether this path is a socket.

is_symlink()

Whether this path is a symbolic link.

iterdir()

Iterate over the files in this directory. Does not yield any result for the special paths '.' and '..'.

lchmod(*mode*)

Like chmod(), except if the path points to a symlink, the symlink's permissions are changed, rather than its target's.

link_to(*target*)

Make the target path a hard link pointing to this path.

Note this function does not make this path a hard link to *target*, despite the implication of the function and argument names. The order of arguments (target, link) is the reverse of Path.symlink_to, but matches that of os.link.

Deprecated since Python 3.10 and scheduled for removal in Python 3.12. Use *hardlink_to()* instead.

lstat()

Like stat(), except if the path points to a symlink, the symlink's status information is returned, rather than its target's.

mkdir(*mode=511*, *parents=False*, *exist_ok=False*)

Create a new directory at this given path.

open(*mode='r'*, *buffering=-1*, *encoding=None*, *errors=None*, *newline=None*)

Open the file pointed by this path and return a file object, as the built-in open() function does.

owner()

Return the login name of the file owner.

read_bytes()

Open the file in bytes mode, read it, and close the file.

read_text(*encoding=None*, *errors=None*)

Open the file in text mode, read it, and close the file.

readlink()

Return the path to which the symbolic link points.

rename(*target*)

Rename this path to the target path.

The target path may be absolute or relative. Relative paths are interpreted relative to the current working directory, *not* the directory of the Path object.

Returns the new Path instance pointing to the target path.

replace(*target*)

Rename this path to the target path, overwriting if that path exists.

The target path may be absolute or relative. Relative paths are interpreted relative to the current working directory, *not* the directory of the Path object.

Returns the new Path instance pointing to the target path.

resolve(*strict=False*)

Make the path absolute, resolving all symlinks on the way and also normalizing it (for example turning slashes into backslashes under Windows).

rglob(*pattern*)

Recursively yield all existing files (of any kind, including directories) matching the given relative pattern, anywhere in this subtree.

rmdir()

Remove this directory. The directory must be empty.

samefile(*other_path*)

Return whether other_path is the same or not as this file (as returned by os.path.samefile()).

stat(**, follow_symlinks=True*)

Return the result of the stat() system call on this path, like os.stat() does.

symlink_to(*target*, *target_is_directory=False*)

Make this path a symlink pointing to the target path. Note the order of arguments (link, target) is the reverse of os.symlink.

touch(*mode=438*, *exist_ok=True*)

Create this file with the given access mode, if it doesn't exist.

unlink(*missing_ok=False*)

Remove this file or link. If the path is a directory, use rmdir() instead.

write_bytes(*data*)

Open the file in bytes mode, write to it, and close the file.

write_text(*data*, *encoding=None*, *errors=None*, *newline=None*)

Open the file in text mode, write to it, and close the file.

miranda.eccc._support_rvt.**gather_eccc_stations**(*timestep: str*, *start_date: datetime | str | None = None*, *end_date: datetime | str | None = None*, *climate_id: str | None = None*) → DataFrame

miranda.eccc._utils module

miranda.eccc._utils.**cf_ahccd_metadata**(*code: str, gen: int) -> (typing.Mapping[str, typing.Union[int, float, str]], typing.Dict, typing.List[typing.Tuple[int, int]], <class 'int'>)*

Parameters

- **code** (*{"dx", "dn", "dm", "dt", "ds", "dr"}*)
- **gen** (*{1, 2, 3}*)

Returns

Mapping[str, Union[str, float]], Dict, List[Tuple[int, int]], int

miranda.eccc._utils.**cf_station_metadata**(*variable_code: int | str*) → Mapping[str, int | float | str]

Parameters

variable_code (*Union[int, str]*)

Returns

dict

miranda.ecmwf package

miranda.ecmwf.**rename_era5_files**(*path: PathLike | str*) → None

Rename badly named ERA5 files.

Notes

Requires that the proper ERA5 project name is in the filename, separated by underscores. Assumes that the data

Parameters
path (*os.PathLike or str*) – Path to a folder containing netcdf files

Returns
None

miranda.ecmwf.**request_era5**(*projects: str | List[str]*, ***, *variables: str | Sequence[str] | None = None*, *domain: str = 'AMNO'*, *pressure_levels: List[int] | None = None*, *separate_pressure_levels: bool = True*, *output_folder: str | PathLike | None = None*, *year_start: str | int | None = None*, *year_end: str | int | None = None*, *dry_run: bool = False*, *processes: int = 10*, *url: str | None = None*, *key: str | None = None*) → None

Request ERA5/ERA5-Land from Copernicus Data Store in NetCDF4 format.

Parameters

- **projects** (*str or List[str]*) – Allowed keys: {"era5-land", "era5-land-monthly-means", "era5-single-levels", "era5-single-levels-monthly-means", "era5-single-levels-preliminary-back-extension", "era5-single-levels-monthly-means-preliminary-back-extension", "era5-pressure-levels", "era5-pressure-levels-monthly-means", "era5-pressure-levels-preliminary-back-extension", "era5-pressure-levels-monthly-means-preliminary-back-extension"}
- **variables** (*str or Sequence[str]*) – Variable codes requested. If None, will attempt all hard-coded variables supported by miranda converter.
- **domain** (*{"GLOBAL", "AMNO", "NAM", "CAN", "QC", "MTL"}*) – Geographic domain requested. Default: "AMNO" (North America).
- **pressure_levels** (*List[int], optional*) – If set and project requested has pressure levels, will download specific pressure levels.
- **separate_pressure_levels** (*bool*) – Whether to separate files for each pressure level. Default: True
- **output_folder** (*str or os.PathLike, optional*) – Folder to send files to. If None, will create a "downloaded" folder in current working directory.
- **year_start** (*int, optional*) – Starting year for data download. If None, will download from first available year for project.
- **year_end** (*int, optional*) – End year for data download. If None, will download files for current year and two months prior to present day.
- **dry_run** (*bool*) – Do not send request. For debugging purposes.
- **processes** (*int*) – The number of simultaneous download requests. Default: 10.
- **url** (*str, optional*) – URL for Copernicus Data Store API (if not already using .cdsapirc)
- **key** (*str, optional*) – Personal access key for Copernicus Data Store (if not already using .cdsapirc)

Returns
None

miranda.ecmwf.**request_tigge**(*variables: List[str]*, *providers: List[str] | None = None*, ***, *forecast_type: str = 'pf'*, *times: List[str] | None = None*, *dates: List[str] | None = None*, *date_start: str | None = None*, *date_end: str | None = None*, *output_folder: PathLike | None = None*, *processes: int = 4*) → None

Request tigge data from ECMWF in grib format.

Parameters

- **variables** (*List[str]*)
- **providers** (*List[str], optional*)
- **forecast_type** (*{"pf", "cf"}*)
- **times** (*List[str], optional*)
- **dates** (*List[str]. optional*)
- **date_start** (*str, optional*)
- **date_end** (*str, optional*)
- **output_folder** (*os.PathLike, optional*)
- **processes** (*int*)

Returns
None

Submodules

miranda.ecmwf._era5 module

miranda.ecmwf._era5.**rename_era5_files**(*path: PathLike | str*) → None

Rename badly named ERA5 files.

Notes

Requires that the proper ERA5 project name is in the filename, separated by underscores. Assumes that the data

Parameters
path (*os.PathLike or str*) – Path to a folder containing netcdf files

Returns
None

miranda.ecmwf._era5.**request_era5**(*projects: str | List[str]*, ***, *variables: str | Sequence[str] | None = None*, *domain: str = 'AMNO'*, *pressure_levels: List[int] | None = None*, *separate_pressure_levels: bool = True*, *output_folder: str | PathLike | None = None*, *year_start: str | int | None = None*, *year_end: str | int | None = None*, *dry_run: bool = False*, *processes: int = 10*, *url: str | None = None*, *key: str | None = None*) → None

Request ERA5/ERA5-Land from Copernicus Data Store in NetCDF4 format.

Parameters

- **projects** (*str or List[str]*) – Allowed keys: {"era5-land", "era5-land-monthly-means", "era5-single-levels", "era5-single-levels-monthly-means", "era5-single-levels-preliminary-back-extension", "era5-single-levels-monthly-means-preliminary-back-extension", "era5-pressure-levels", "era5-pressure-levels-monthly-means", "era5-pressure-levels-preliminary-back-extension", "era5-pressure-levels-monthly-means-preliminary-back-extension"}
- **variables** (*str or Sequence[str]*) – Variable codes requested. If None, will attempt all hard-coded variables supported by miranda converter.
- **domain** (*{"GLOBAL", "AMNO", "NAM", "CAN", "QC", "MTL"}*) – Geographic domain requested. Default: "AMNO" (North America).
- **pressure_levels** (*List[int], optional*) – If set and project requested has pressure levels, will download specific pressure levels.
- **separate_pressure_levels** (*bool*) – Whether to separate files for each pressure level. Default: True
- **output_folder** (*str or os.PathLike, optional*) – Folder to send files to. If None, will create a "downloaded" folder in current working directory.
- **year_start** (*int, optional*) – Starting year for data download. If None, will download from first available year for project.
- **year_end** (*int, optional*) – End year for data download. If None, will download files for current year and two months prior to present day.
- **dry_run** (*bool*) – Do not send request. For debugging purposes.
- **processes** (*int*) – The number of simultaneous download requests. Default: 10.
- **url** (*str, optional*) – URL for Copernicus Data Store API (if not already using .cdsapirc)
- **key** (*str, optional*) – Personal access key for Copernicus Data Store (if not already using .cdsapirc)

Returns
None

miranda.ecmwf._tigge module

miranda.ecmwf._tigge.**request_tigge**(*variables: List[str]*, *providers: List[str] | None = None*, *, forecast_type: str = 'pf'*, *times: List[str] | None = None*, *dates: List[str] | None = None*, *date_start: str | None = None*, *date_end: str | None = None*, *output_folder: PathLike | None = None*, *processes: int = 4*) → None

Request tigge data from ECMWF in grib format.

Parameters

- **variables** (*List[str]*)
- **providers** (*List[str], optional*)
- **forecast_type** (*{"pf", "cf"}*)
- **times** (*List[str], optional*)
- **dates** (*List[str]. optional*)
- **date_start** (*str, optional*)
- **date_end** (*str, optional*)

- **output_folder** (*os.PathLike, optional*)
- **processes** (*int*)

Returns
None

miranda.gis package

miranda.gis.**add_ar6_regions**(*ds: Dataset*) → Dataset

Add the IPCC AR6 Regions to dataset.

Parameters
ds (*xarray.Dataset*)

Returns
xarray.Dataset

miranda.gis.**subset_domain**(*ds: Dataset | DataArray*, *domain: str*, ***kwargs*) → Dataset | DataArray

miranda.gis.**subsetting_domains**(*domain: str*) → List

Provides the bounding box coordinates for specific domains.

Parameters
domain (*{"global", "nam", "can", "qc", "mtl"}*)

Returns
np.array – North, West, South, and East coordinates

Submodules

miranda.gis._domains module

miranda.gis._domains.**add_ar6_regions**(*ds: Dataset*) → Dataset

Add the IPCC AR6 Regions to dataset.

Parameters
ds (*xarray.Dataset*)

Returns
xarray.Dataset

miranda.gis._domains.**subset_domain**(*ds: Dataset | DataArray*, *domain: str*, ***kwargs*) → Dataset | DataArray

miranda.gis._domains.**subsetting_domains**(*domain: str*) → List

Provides the bounding box coordinates for specific domains.

Parameters
domain (*{"global", "nam", "can", "qc", "mtl"}*)

Returns
np.array – North, West, South, and East coordinates

miranda.io package

miranda.io.**concat_rechunk_zarr**(*project: str*, *freq: str*, *input_folder: str | PathLike*, *output_folder: str | PathLike*, *overwrite: bool = False*, ***dask_kwargs*)

Parameters

- **project**
- **freq**
- **input_folder**
- **output_folder**
- **overwrite**
- **dask_kwargs**

miranda.io.**discover_data**(*input_files: str | PathLike | List[str | PathLike] | generator*, *suffix: str = 'nc'*, *recurse: bool = True*) → List[*Path*] | generator

Discover data.

Parameters

- **input_files** (*str or Path or List[Union[str, Path]] or GeneratorType*) – Path or string to a file, a folder, or a generator of paths.
- **suffix** (*str*) – File-ending suffix to search for. Default: "nc".
- **recurse** (*bool*) – Whether to recurse through folders or not. Default: True.

Returns

list or generator of Path

Warning: Recursion through ".zarr" files is explicitly disabled. Recursive globs and generators will not be expanded/sorted.

miranda.io.**fetch_chunk_config**(*project: str*, *freq: str*, *priority: str = 'files'*) → Dict[str, int]

Parameters

- **project** (*str, optional*) – Supported projects. Used for determining chunk dictionary.
- **freq** (*{"1hr", "day", "month"}*) – The chunking regime for
- **priority** (*{"time", "files"}*) – Specifies whether the chunking regime should prioritize file granularity ("files") or time series ("time").

miranda.io.**find_filepaths**(*source: Path | str | generator | List[str | Path]*, *recursive: bool = True*, *file_suffixes: str | List[str] | None = None*, ***_*) → List[*Path*]

Find all available filepaths at a given source.

Parameters

- **source** (*Union[Path, str, GeneratorType, List[Union[Path, str]]]*)
- **recursive** (*bool*)
- **file_suffixes** (*List[str]*)

Returns

List[Path]

miranda.io.**merge_rechunk_zarrs**(*input_folder: str | PathLike*, *output_folder: str | PathLike*, *project: str | None = None*, *target_chunks: Dict[str, int] | None = None*, *variables: Sequence[str] | None = None*, *freq: str | None = None*, *suffix: str = 'zarr'*, *overwrite: bool = False*)

Parameters

- **input_folder**
- **output_folder**
- **project**
- **target_chunks**
- **variables**
- **freq**
- **suffix**
- **overwrite**

miranda.io.**rechunk_files**(*input_folder: str | PathLike*, *output_folder: str | PathLike*, *project: str | None = None*, *time_step: str | None = None*, *chunking_priority: str = 'auto'*, *target_chunks: Dict[str, int] | None = None*, *variables: Sequence[str] | None = None*, *suffix: str = 'nc'*, *output_format: str = 'netcdf'*, *overwrite: bool = False*) → None

Rechunks dataset for better loading/reading performance.

Warning: Globbing assumes that target datasets to be rechunked have been saved in NetCDF format. File naming requires the following order of facets: *{variable}_{time_step}_{institute}_{project}_reanalysis_*.nc*. Chunking dimensions are assumed to be CF-Compliant (*lat*, *lon*, *rlat*, *rlon*, *time*).

Parameters

- **input_folder** (*str or os.PathLike*) – Folder to be examined. Performs globbing.
- **output_folder** (*str or os.PathLike*) – Target folder.
- **project** (*str, optional*) – Supported projects. Used for determining chunk dictionary. Superseded if *target_chunks* is set.
- **time_step** (*{"1hr", "day"}, optional*) – Time step of the input data. Parsed from dataset attrs if not set. Superseded if *target_chunks* is set.
- **chunking_priority** (*{"time", "files", "auto"}*) – The chunking regime to use. Default: "auto".
- **target_chunks** (*dict, optional*) – Must include "time", optionally "lat" and "lon", depending on dataset structure.
- **variables** (*Sequence[str], optional*) – If no variables set, will attempt to process all variables supported based on project name.
- **suffix** (*{"nc", "zarr"}*) – Suffix used to identify data files. Default: "nc".
- **output_format** (*{"netcdf", "zarr"}*) – Default: "zarr".
- **overwrite** (*bool*) – Will overwrite files. For zarr, existing folders will be removed before writing.

Returns
None

miranda.io.**translate_time_chunk**(*chunks: dict*, *calendar: str*, *timesize: int*) → dict

Translate chunk specification for time into a number.

Notes

-1 translates to *timesize* 'Nyear' translates to N times the number of days in a year of calendar *calendar*.

miranda.io.**write_dataset**(*ds: DataArray | Dataset*, *project: str*, *output_path: str | PathLike*, *output_format: str*, *chunks: dict | None = None*, *overwrite: bool = False*, *compute: bool = True*)

Parameters

- **ds** (*xr.DataArray or xr.Dataset*)
- **project** (*{"cordex", "cmip5", "cmip6", "ets-grnch", "isimip-ft", "pcic-candcs-u6", "converted"}*) – Project name for decoding/handling purposes.
- **output_path** (*str or os.PathLike*) – Output folder path.
- **output_format** (*{"netcdf", "zarr"}*) – Output data container type.
- **chunks** (*dict, optional*) – Chunking layout to be written to new files. If None, chunking will be left to the relevant backend engine.
- **overwrite** (*bool*) – Whether to remove existing files or fail if files already exist.
- **compute** (*bool*) – If True, files will be converted with each call to file conversion. If False, will return a dask.Delayed object that can be computed later. Default: True.

miranda.io.**write_dataset_dict**(*dataset_dict: Dict[str, Dataset]*, *output_folder: str | PathLike*, *temp_folder: str | PathLike*, *output_format: str = 'zarr'*, *overwrite: bool = False*, *chunks: Dict[str, int] | None = None*, ***dask_kwargs*)

Submodules

miranda.io._input module

miranda.io._input.**discover_data**(*input_files: str | PathLike | List[str | PathLike] | generator*, *suffix: str = 'nc'*, *recurse: bool = True*) → List[*Path*] | generator

Discover data.

Parameters

- **input_files** (*str or Path or List[Union[str, Path]] or GeneratorType*) – Path or string to a file, a folder, or a generator of paths.
- **suffix** (*str*) – File-ending suffix to search for. Default: "nc".
- **recurse** (*bool*) – Whether to recurse through folders or not. Default: True.

Returns
list or generator of Path

> **Warning:** Recursion through ".zarr" files is explicitly disabled. Recursive globs and generators will not be expanded/sorted.

miranda.io._input.**find_filepaths**(*source: Path | str | generator | List[str | Path]*, *recursive: bool = True*, *file_suffixes: str | List[str] | None = None*, ***_*) → List[*Path*]

Find all available filepaths at a given source.

Parameters

- **source** (*Union[Path, str, GeneratorType, List[Union[Path, str]]]*)
- **recursive** (*bool*)
- **file_suffixes** (*List[str]*)

Returns
List[Path]

miranda.io._output module

miranda.io._output.**concat_rechunk_zarr**(*project: str*, *freq: str*, *input_folder: str | PathLike*, *output_folder: str | PathLike*, *overwrite: bool = False*, ***dask_kwargs*)

Parameters

- **project**
- **freq**
- **input_folder**
- **output_folder**
- **overwrite**
- **dask_kwargs**

miranda.io._output.**merge_rechunk_zarrs**(*input_folder: str | PathLike*, *output_folder: str | PathLike*, *project: str | None = None*, *target_chunks: Dict[str, int] | None = None*, *variables: Sequence[str] | None = None*, *freq: str | None = None*, *suffix: str = 'zarr'*, *overwrite: bool = False*)

Parameters

- **input_folder**
- **output_folder**
- **project**
- **target_chunks**
- **variables**
- **freq**
- **suffix**
- **overwrite**

miranda.io._output.**write_dataset**(*ds: DataArray | Dataset*, *project: str*, *output_path: str | PathLike*, *output_format: str*, *chunks: dict | None = None*, *overwrite: bool = False*, *compute: bool = True*)

Parameters

- **ds** (*xr.DataArray or xr.Dataset*)
- **project** (*{"cordex", "cmip5", "cmip6", "ets-grnch", "isimip-ft", "pcic-candcs-u6", "converted"}*) – Project name for decoding/handling purposes.
- **output_path** (*str or os.PathLike*) – Output folder path.
- **output_format** (*{"netcdf", "zarr"}*) – Output data container type.
- **chunks** (*dict, optional*) – Chunking layout to be written to new files. If None, chunking will be left to the relevant backend engine.
- **overwrite** (*bool*) – Whether to remove existing files or fail if files already exist.
- **compute** (*bool*) – If True, files will be converted with each call to file conversion. If False, will return a dask.Delayed object that can be computed later. Default: True.

miranda.io._output.**write_dataset_dict**(*dataset_dict: Dict[str, Dataset]*, *output_folder: str | PathLike*, *temp_folder: str | PathLike*, *output_format: str = 'zarr'*, *overwrite: bool = False*, *chunks: Dict[str, int] | None = None*, ***dask_kwargs*)

miranda.io._rechunk module

miranda.io._rechunk.**fetch_chunk_config**(*project: str*, *freq: str*, *priority: str = 'files'*) → Dict[str, int]

Parameters

- **project** (*str, optional*) – Supported projects. Used for determining chunk dictionary.
- **freq** (*{"1hr", "day", "month"}*) – The chunking regime for
- **priority** (*{"time", "files"}*) – Specifies whether the chunking regime should prioritize file granularity ("files") or time series ("time").

miranda.io._rechunk.**rechunk_files**(*input_folder: str | PathLike*, *output_folder: str | PathLike*, *project: str | None = None*, *time_step: str | None = None*, *chunking_priority: str = 'auto'*, *target_chunks: Dict[str, int] | None = None*, *variables: Sequence[str] | None = None*, *suffix: str = 'nc'*, *output_format: str = 'netcdf'*, *overwrite: bool = False*) → None

Rechunks dataset for better loading/reading performance.

Warning: Globbing assumes that target datasets to be rechunked have been saved in NetCDF format. File naming requires the following order of facets: *{variable}_{time_step}_{institute}_{project}_reanalysis_*.nc*. Chunking dimensions are assumed to be CF-Compliant (*lat*, *lon*, *rlat*, *rlon*, *time*).

Parameters

- **input_folder** (*str or os.PathLike*) – Folder to be examined. Performs globbing.
- **output_folder** (*str or os.PathLike*) – Target folder.

- **project** (*str, optional*) – Supported projects. Used for determining chunk dictionary. Superseded if *target_chunks* is set.
- **time_step** (*{"1hr", "day"}, optional*) – Time step of the input data. Parsed from dataset attrs if not set. Superseded if *target_chunks* is set.
- **chunking_priority** (*{"time", "files", "auto"}*) – The chunking regime to use. Default: "auto".
- **target_chunks** (*dict, optional*) – Must include "time", optionally "lat" and "lon", depending on dataset structure.
- **variables** (*Sequence[str], optional*) – If no variables set, will attempt to process all variables supported based on project name.
- **suffix** (*{"nc", "zarr"}*) – Suffix used to identify data files. Default: "nc".
- **output_format** (*{"netcdf", "zarr"}*) – Default: "zarr".
- **overwrite** (*bool*) – Will overwrite files. For zarr, existing folders will be removed before writing.

Returns
None

`miranda.io._rechunk.`**`translate_time_chunk`**`(`*chunks: dict*, *calendar: str*, *timesize: int*`)` → dict

Translate chunk specification for time into a number.

Notes

-1 translates to *timesize* 'Nyear' translates to N times the number of days in a year of calendar *calendar*.

miranda.io.utils module

`miranda.io.utils.`**`creation_date`**`(`*path_to_file:* Path | *str*`)` → float | date

Try to get the date that a file was created, falling back to when it was last modified if that isn't possible.

See https://stackoverflow.com/a/39501288/1709587 for explanation.

Parameters
path_to_file (*Union[Path, str]*)

Returns
Union[float, date]

`miranda.io.utils.`**`delayed_write`**`(`*ds: Dataset*, *outfile: str | PathLike*, *output_format: str*, *overwrite: bool*, *target_chunks: dict | None = None*`)` → delayed

Parameters

- **ds** (*Union[xr.Dataset, str, os.PathLike]*)
- **outfile** (*str or os.PathLike*)
- **target_chunks** (*dict*)
- **output_format** (*{"netcdf", "zarr"}*)
- **overwrite** (*bool*)

Returns
dask.delayed.delayed

miranda.io.utils.**get_chunks_on_disk**(*file: PathLike | str*) → dict

Parameters
file (*str or os.PathLike*) – File to be examined. Supports NetCDF and Zarr.

Returns
dict

miranda.io.utils.**get_global_attrs**(*file_or_dataset: str | PathLike | Dataset*) → Dict[str, str | int]

miranda.io.utils.**get_time_attrs**(*file_or_dataset: str | ~os.PathLike | ~xarray.core.dataset.Dataset) ->*
(<class 'str'>, <class 'int'>)

miranda.io.utils.**name_output_file**(*ds_or_dict: Dataset | Dict[str, str]*, *project: str*, *output_format: str*) →
str

Name an output file based on facets within a Dataset or a dictionary.

Parameters

- **ds_or_dict** (*xr.Dataset or dict*)
- **project** (*str*)
- **output_format** (*{"netcdf", "zarr"}*) – Suffix to be used for filename

Returns
str

Notes

If using a dictionary, the following must be keys: "variable", "frequency", "institution", "time_start", "time_end".

miranda.io.utils.**sort_variables**(*files: List[Path]*, *variables: Sequence[str]*) → Dict[str, List[*Path*]]

miranda.ncar package

miranda.ncar.**cordex_aws_calendar_correction**(*ds*) → Dataset | None

AWS-stored CORDEX datasets are all on the same standard calendar, this converts the data back to the original calendar, removing added NaNs.

Credit: Pascal Bourgault (@aulemahal)

miranda.ncar.**cordex_aws_download**(*target_folder: str | Path*, ***, *search: Dict[str, str | List[str]]*, *correct_times: bool = False*, *domain: str | None = None*)

Submodules

miranda.ncar._aws_cordex module

miranda.ncar._aws_cordex.**cordex_aws_calendar_correction**(*ds*) → Dataset | None

AWS-stored CORDEX datasets are all on the same standard calendar, this converts the data back to the original calendar, removing added NaNs.

Credit: Pascal Bourgault (@aulemahal)

miranda.ncar._aws_cordex.**cordex_aws_download**(*target_folder: str | Path*, *, *search: Dict[str, str | List[str]]*, *correct_times: bool = False*, *domain: str | None = None*)

miranda.remote package

class miranda.remote.**Connection**(*username: str | Path | None = None*, *host: str | Path | None = None*, *protocol: str = 'sftp'*, **args*, ***kwargs*)

Bases: `object`

connect(***kwargs*)

update(***kwargs*)

miranda.remote.**archive_database**(*source: Path | str | List*, *common_path: Path | str*, *destination: Path | str*, *file_suffixes: str = '.nc'*, *server: str | None = None*, *username: str | None = None*, *project_name: str | None = None*, *overwrite: bool = False*, *compression: bool = False*, *recursive: bool = False*, *use_grouping: bool = True*, *use_subdirectories: bool = True*, *dry_run: bool = False*) → None

Given a source, destination, and dependent on file size limit, create tarfile archives and transfer files to another server for backup purposes

miranda.remote.**create_archive**(*source_files: list[str | os.PathLike]*, *destination: str | os.PathLike*, *transport: SCPClient | SFTPClient | fabric.Connection | miranda.remote.Connection = None*, *delete: bool = True*, *compression: bool = False*, *recursive: bool = True*) → None

Parameters

- **source_files** (*List[Union[str, os.PathLike]]*)
- **destination** (*Union[str, os.PathLike]*)
- **transport** (*Union[SCPClient, SFTPClient, fabric.Connection, Connection]*)
- **delete** (*bool*)
- **compression** (*bool*)
- **recursive** (*bool*)

Returns

None

miranda.remote.**create_remote_directory**(*directory: str | os.PathLike*, *transport: SSHClient | fabric.Connection | miranda.remote.Connection*) → None

This calls a "mkdir -p" function to create a folder structure over SFTP/SSH and waits for confirmation before continuing

Parameters

- **directory** (*Union[str, os.PathLike]*)
- **transport** (*Union[SSHClient, fabric.Connection, Connection]*)

Returns
None

miranda.remote.**delete_by_date**(*, *source: str | Path*, *year: int | None = None*, *month: int | None = None*, *day: int | None = None*, *pattern: str | None = None*, *server: str | Path | None = None*, *user: str | None = None*, *password: str | None = None*, *date_object: date | None = None*) → None

Parameters

- **source** (*Union[str, Path]*)
- **year** (*Optional[int]*)
- **month** (*Optional[int]*)
- **day** (*Optional[int]*)
- **pattern** (*Optional[str]*)
- **server** (*Optional[Union[str, Path]]*)
- **user** (*Optional[str]*)
- **password** (*Optional[str]*)
- **date_object** (*Optional[date]*)

Returns
None

miranda.remote.**delete_by_variable**(*, *target: str | Path | List[str | Path] | generator | None = None*, *variables: List[str] | None = None*, *server: str | None*, *user: str | None = None*, *password: str | None = None*, *file_suffix: str | None = None*, *delete: bool = False*) → None

Given target location(s), a list of variables and a server address, perform a glob search
and delete file names starting with the variables identified

Parameters

- **target** (*Union[str, Path, List[Union[str, Path]], GeneratorType]*)
- **variables** (*List[str]*)
- **server** (*Optional[Union[str, Path]]*)
- **user** (*Optional[str]*)
- **password** (*Optional[str]*)
- **file_suffix** (*Optional[str]*)
- **delete** (*bool*)

Returns
None

miranda.remote.**delete_duplicates**(*, *source: str | Path*, *target: str | Path*, *server: str | Path | None*, *user: str | None = None*, *password: str | None = None*, *pattern: str | None = None*, *delete_target_duplicates: bool = False*) → None

Parameters

- **source** (*Union[str, Path]*)
- **target** (*Union[str, Path]*)
- **server** (*Optional[Union[str, Path]]*)
- **user** (*str*)
- **password** (*str*)
- **pattern** (*str*)
- **delete_target_duplicates** (*bool*)

Returns
None

miranda.remote.**file_emptier**(*, *file_list: List[str | Path] | generator*) → None

Provided a list of file paths, will open and overwrite them in order to delete data while preserving the file name.

Parameters
file_list (*List[Union[str, Path]]*) – List of files to be overwritten

Returns
None

miranda.remote.**transfer_file**(*source_file: str | os.PathLike*, *destination_file: str | os.PathLike*, *transport: SCPClient | SFTPClient | fabric.Connection | miranda.remote.Connection = None*) → bool

Parameters

- **source_file** (*Union[str, os.PathLike]*)
- **destination_file** (*Union[str, os.PathLike]*)
- **transport** (*Union[SCPClient, SFTPClient, fabric.Connection, Connection]*)

Returns
bool

Submodules

miranda.remote.archiver module

miranda.remote.archiver.**archive_database**(*source: Path | str | List*, *common_path: Path | str*, *destination: Path | str*, *file_suffixes: str = '.nc'*, *server: str | None = None*, *username: str | None = None*, *project_name: str | None = None*, *overwrite: bool = False*, *compression: bool = False*, *recursive: bool = False*, *use_grouping: bool = True*, *use_subdirectories: bool = True*, *dry_run: bool = False*) → None

Given a source, destination, and dependent on file size limit, create tarfile archives and transfer
files to another server for backup purposes

miranda.remote.connect module

class miranda.remote.connect.**Connection**(*username: str | Path | None = None*, *host: str | Path | None = None*, *protocol: str = 'sftp'*, **args*, ***kwargs*)

Bases: object

connect(***kwargs*)

update(***kwargs*)

miranda.remote.ops module

miranda.remote.ops.**create_archive**(*source_files: list[str | os.PathLike]*, *destination: str | os.PathLike*, *transport: SCPClient | SFTPClient | fabric.Connection | miranda.remote.Connection = None*, *delete: bool = True*, *compression: bool = False*, *recursive: bool = True*) → None

Parameters

- **source_files** (*List[Union[str, os.PathLike]]*)
- **destination** (*Union[str, os.PathLike]*)
- **transport** (*Union[SCPClient, SFTPClient, fabric.Connection, Connection]*)
- **delete** (*bool*)
- **compression** (*bool*)
- **recursive** (*bool*)

Returns
None

miranda.remote.ops.**create_remote_directory**(*directory: str | os.PathLike*, *transport: SSHClient | fabric.Connection | miranda.remote.Connection*) → None

This calls a "mkdir -p" function to create a folder structure over SFTP/SSH and waits for confirmation before continuing

Parameters

- **directory** (*Union[str, os.PathLike]*)
- **transport** (*Union[SSHClient, fabric.Connection, Connection]*)

Returns
None

miranda.remote.ops.**transfer_file**(*source_file: str | os.PathLike*, *destination_file: str | os.PathLike*, *transport: SCPClient | SFTPClient | fabric.Connection | miranda.remote.Connection = None*) → bool

Parameters

- **source_file** (*Union[str, os.PathLike]*)
- **destination_file** (*Union[str, os.PathLike]*)
- **transport** (*Union[SCPClient, SFTPClient, fabric.Connection, Connection]*)

Returns
bool

miranda.remote.remove module

miranda.remote.remove.**delete_by_date**(*, *source: str | Path*, *year: int | None = None*, *month: int | None = None*, *day: int | None = None*, *pattern: str | None = None*, *server: str | Path | None = None*, *user: str | None = None*, *password: str | None = None*, *date_object: date | None = None*) → None

Parameters

- **source** (*Union[str, Path]*)
- **year** (*Optional[int]*)
- **month** (*Optional[int]*)
- **day** (*Optional[int]*)
- **pattern** (*Optional[str]*)
- **server** (*Optional[Union[str, Path]]*)
- **user** (*Optional[str]*)
- **password** (*Optional[str]*)
- **date_object** (*Optional[date]*)

Returns
None

miranda.remote.remove.**delete_by_variable**(*, *target: str | Path | List[str | Path] | generator | None = None*, *variables: List[str] | None = None*, *server: str | None*, *user: str | None = None*, *password: str | None = None*, *file_suffix: str | None = None*, *delete: bool = False*) → None

Given target location(s), a list of variables and a server address, perform a glob search
and delete file names starting with the variables identified

Parameters

- **target** (*Union[str, Path, List[Union[str, Path]], GeneratorType]*)
- **variables** (*List[str]*)
- **server** (*Optional[Union[str, Path]]*)
- **user** (*Optional[str]*)
- **password** (*Optional[str]*)
- **file_suffix** (*Optional[str]*)
- **delete** (*bool*)

Returns
None

miranda.remote.remove.**delete_duplicates**(*, *source: str | Path*, *target: str | Path*, *server: str | Path | None*, *user: str | None = None*, *password: str | None = None*, *pattern: str | None = None*, *delete_target_duplicates: bool = False*) → None

Parameters

- **source** (*Union[str, Path]*)
- **target** (*Union[str, Path]*)
- **server** (*Optional[Union[str, Path]]*)
- **user** (*str*)
- **password** (*str*)
- **pattern** (*str*)
- **delete_target_duplicates** (*bool*)

Returns
None

miranda.remote.remove.**file_emptier**(*, *file_list: List[str | Path] | generator*) → None

Provided a list of file paths, will open and overwrite them in order to delete data while preserving the file name.

Parameters
file_list (*List[Union[str, Path]]*) – List of files to be overwritten

Returns
None

miranda.structure package

miranda.structure.**build_path_from_schema**(*facets: dict*, *output_folder: str | PathLike*, *schema: str | PathLike | dict | None = None*, *top_folder: str = 'datasets'*, *validate: bool = True*) → *Path* | None

Build a filepath based on a valid data schema.

Parameters

- **facets** (*dict*) – Facets for a given dataset.
- **output_folder** (*str or os.PathLike*) – Parent folder on which to extend the filetree structure.
- **schema** (*str or os.PathLike, optional*) – Path to YAML schematic of database structure. If None, will use Ouranos schema.
- **top_folder** (*str*) – Top-level of supplied schema, used for validation purposes. Default: "datasets".
- **validate** (*bool*) – Run facets-validation checks over given file. Default: True.

Returns
Path or None

miranda.structure.**create_version_hash_files**(*input_files: str | PathLike | List[str | PathLike] | generator | None = None*, *facet_dict: Dict | None = None*, *verify_hash: bool = False*) → None

miranda.structure.**structure_datasets**(*input_files: str | PathLike | List[str | PathLike] | generator*, *output_folder: str | PathLike*, ***, *project: str | None = None*, *guess: bool = True*, *dry_run: bool = False*, *method: str = 'copy'*, *make_dirs: bool = False*, *set_version_hashes: bool = False*, *verify_hashes: bool = False*, *suffix: str = 'nc'*) → Dict[*Path*, *Path*]

Parameters

- **input_files** (*str or Path or list of str or Path or GeneratorType*)
- **output_folder** (*str or Path*)
- **project** (*{"cordex", "cmip5", "cmip6", "isimip-ft", "pcic-candcs-u6", "converted"}, optional*) – Project used to parse the facets of all supplied datasets. If not supplied, will attempt parsing with all available data categories for each file (slow) unless *guess* is True.
- **guess** (*bool*) – If project not supplied, suggest to decoder that activity is the same for all input_files. Default: True.
- **dry_run** (*bool*) – Prints changes that would have been made without performing them. Default: False.
- **method** (*{"move", "copy"}*) – Method to transfer files to intended location. Default: "move".
- **make_dirs** (*bool*) – Make folder tree if it does not already exist. Default: False.
- **set_version_hashes** (*bool*) – Make an accompanying file with version in filename and sha256sum in contents. Default: False.
- **verify_hashes** (*bool*) – Ensure that any existing she256sum files correspond with companion file. Raise on error. Default: False.
- **suffix** (*{"nc", "zarr"}*) – If "zarr", will perform a 'glob' with provided pattern. Otherwise, will perform an 'rglob' (recursive) operation.

Returns
Dict[Path, Path]

Submodules

miranda.structure._structure module

miranda.structure._structure.**build_path_from_schema**(*facets: dict*, *output_folder: str | PathLike*, *schema: str | PathLike | dict | None = None*, *top_folder: str = 'datasets'*, *validate: bool = True*) → *Path* | None

Build a filepath based on a valid data schema.

Parameters

- **facets** (*dict*) – Facets for a given dataset.
- **output_folder** (*str or os.PathLike*) – Parent folder on which to extend the filetree structure.
- **schema** (*str or os.PathLike, optional*) – Path to YAML schematic of database structure. If None, will use Ouranos schema.
- **top_folder** (*str*) – Top-level of supplied schema, used for validation purposes. Default: "datasets".
- **validate** (*bool*) – Run facets-validation checks over given file. Default: True.

Returns
Path or None

miranda.structure._structure.**create_version_hash_files**(*input_files: str | PathLike | List[str | PathLike] | generator | None = None*, *facet_dict: Dict | None = None*, *verify_hash: bool = False*) → None

miranda.structure._structure.**structure_datasets**(*input_files: str | PathLike | List[str | PathLike] | generator*, *output_folder: str | PathLike*, ***, *project: str | None = None*, *guess: bool = True*, *dry_run: bool = False*, *method: str = 'copy'*, *make_dirs: bool = False*, *set_version_hashes: bool = False*, *verify_hashes: bool = False*, *suffix: str = 'nc'*) → Dict[*Path*, *Path*]

Parameters

- **input_files** (*str or Path or list of str or Path or GeneratorType*)
- **output_folder** (*str or Path*)
- **project** (*{"cordex", "cmip5", "cmip6", "isimip-ft", "pcic-candcs-u6", "converted"}, optional*) – Project used to parse the facets of all supplied datasets. If not supplied, will attempt parsing with all available data categories for each file (slow) unless *guess* is True.
- **guess** (*bool*) – If project not supplied, suggest to decoder that activity is the same for all input_files. Default: True.
- **dry_run** (*bool*) – Prints changes that would have been made without performing them. Default: False.
- **method** (*{"move", "copy"}*) – Method to transfer files to intended location. Default: "move".
- **make_dirs** (*bool*) – Make folder tree if it does not already exist. Default: False.
- **set_version_hashes** (*bool*) – Make an accompanying file with version in filename and sha256sum in contents. Default: False.
- **verify_hashes** (*bool*) – Ensure that any existing she256sum files correspond with companion file. Raise on error. Default: False.
- **suffix** (*{"nc", "zarr"}*) – If "zarr", will perform a 'glob' with provided pattern. Otherwise, will perform an 'rglob' (recursive) operation.

Returns

Dict[Path, Path]

Submodules

miranda.cv module

miranda.data module

class miranda.data.**DataBase**(*source*, ***, *destination: str | Path | None = None*, *common_path: str | Path | None = None*, *file_pattern: str | List[str] = '*.nc'*, *project_name: str | None = None*, *recursive: bool = True*)

Bases: `object`

archive()

group_by(***, *common_path: Path | str | None = None*, *subdirectories: bool = True*, *dates: bool = True*, *size: int = 10737418240*)

items()

keys()

target(*target:* Path | *str*)

transfer()

values()

miranda.scripting module

miranda.storage module

Disk space management

Classes:

- DiskSpaceError - the exception raised on failure.
- *FileMeta* - file and its size.
- *StorageState* - storage capacity and availability of a medium.

Functions:

- total_size() - get total size of a list of files.
- *size_division()* - divide files based on number and size restrictions.

exception miranda.storage.**DiskSpaceError**

Bases: Exception

class miranda.storage.**FileMeta**(*path: str*, *size: int = -1*)

Bases: object

File path and size.

django = {'path': ['CharField', 'max_length=512'], 'size': ['IntegerField', 'null=True', 'blank=True']}

class miranda.storage.**StorageState**(*base_path*, *capacity=-1*, *used_space=-1*, *free_space=-1*)

Bases: object

Information regarding the storage capacity of a disk.

miranda.storage.**file_size**(*file_path_or_bytes_or_dict:* Path | *str* | *int* | *List[str* | Path*]* | *generator* | *Dict[str,* Path | *List[*Path*]]*) → int

Parameters

file_path_or_bytes_or_dict (*Union[Path, str, int, List[Union[str, Path]], GeneratorType, Dict[str, Union[Path, List[Path]]]]*)

Returns

int

miranda.storage.**report_file_size**(*file_path_or_bytes_or_dict: Path | str | int | List[str | Path] | Dict[str, Path | List[Path]]*, *use_binary: bool = True*, *significant_digits: int = 2*) → str

This function will parse the contents of a list or generator of files and return the size in bytes of a file or a list of files in pretty formatted text.

miranda.storage.**size_division**(*files_to_divide: List | FileMeta | Path*, *size_limit: int = 0*, *file_limit: int = 0*, *check_name_repetition: bool = False*, *preserve_order: bool = False*) → List[list]

Divide files according to size and number limits.

Parameters

- **files_to_divide** (*Union[List, FileMeta, Path]*)
- **size_limit** (*int*) – Size limit of divisions in bytes. Default: 0 (no limit).
- **file_limit** (*int*) – Number of files limit of divisions. Default: 0 (no limit).
- **check_name_repetition** (*bool*) – Flag to prevent file name repetitions. Default: False.
- **preserve_order** (*bool*) – Flag to force files to be restored in the order they are given. Default: False.

Returns
List[list] – list of divisions (each division is a list of FileMeta objects).

miranda.storage.**size_evaluation**(*file_list: List[str | FileMeta | Path]*) → int

Total size of files.

Parameters
file_list (*Union[str, Path, FileMeta]*)

Returns
int – total size of files in bytes.

miranda.units module

miranda.units.**get_time_frequency**(*d: Dataset*, *expected_period: str | None = None*, *minimum_continuous_period: str = '1M'*) → Tuple[List[int | str], str]

Try to understand the Dataset frequency.

If it can't be inferred with `xarray.infer_freq()` it tries to: - look for a "freq" attrs in the global or time variable attributes. - infer monthly frequency if all time steps are between 27 and 32 days

In the event that an *expected_period* is supplied, special handling will be called allowing for determining data that may be internally discontinuous (e.g. discontinuous overall, but continuous for *minimum_continuous_period*). This is provided for instances where input data in a multifile dataset is sparse.

Parameters

- **d** (*xr.Dataset*)
- **expected_period** (*str*) – An xarray-compatible time period (e.g. "1H", "1D", "7D", "1M", "1A") The time period expected of the input dataset. The "1M" period is specially-handled.
- **minimum_continuous_period** (*str*) – An xarray-compatible time period (e.g. "1H", "1D", "7D", "1M", "1A") The minimum expected granular period that data should have continuous values for. The "1M" period is specially-handled.

Returns

- **offset** (*List[Union[int, str]]*) – The offset a list of (multiplier, base)
- **offset_meaning** (*str*) – The offset meaning (single word)

miranda.units.**parse_offset**(*freq: str*) → Sequence[str]

Parse an offset string.

Parse a frequency offset and, if needed, convert to cftime-compatible components.

Parameters

freq (*str*) – Frequency offset.

Returns

- **multiplier** (*int*) – Multiplier of the base frequency. "[n]W" is always replaced with "[7n]D", as xarray doesn't support "W" for cftime indexes.
- **offset_base** (*str*) – Base frequency. "Y" is always replaced with "A".
- **is_start_anchored** (*bool*) – Whether coordinates of this frequency should correspond to the beginning of the period (*True*) or its end (*False*). Can only be False when base is A, Q or M; in other words, xclim assumes frequencies finer than monthly are all start-anchored.
- **anchor** (*str or None*) – Anchor date for bases A or Q. As xarray doesn't support "W", neither does xclim (anchor information is lost when given).

miranda.utils module

class miranda.utils.**HiddenPrints**

Bases: object

miranda.utils.**chunk_iterables**(*iterable: Sequence*, *chunk_size: int*) → Iterable

Generate lists of *chunk_size* elements from *iterable*.

Notes

Adapted from eidord (2012) https://stackoverflow.com/a/12797249/7322852 (https://creativecommons.org/licenses/by-sa/4.0/)

miranda.utils.**generic_extract_archive**(*resources: str | Path | List[bytes | str | Path]*, *output_dir: str | Path | None = None*) → List[*Path*]

Extract archives (tar/zip) to a working directory.

Parameters

- **resources** (*Union[str, Path, List[Union[bytes, str, Path]]]*) – list of archive files (if netCDF files are in list, they are passed and returned as well in the return).
- **output_dir** (*Optional[Union[str, Path]]*) – string or Path to a working location (default: temporary folder).

Returns

list – List of original or of extracted files

miranda.utils.**list_paths_with_elements**(*base_paths: str | List[str]*, *elements: List[str]*) → List[Dict]

List a given path structure.

Parameters

- **base_paths** (*List[str]*) – list of paths from which to start the search.

- **elements** (*List[str]*) – ordered list of the expected elements.

Returns
List[Dict] – The keys are 'path' and each of the members of the given elements, the path is the absolute path.

Notes

Suppose you have the following structure: /base_path/{color}/{shape} The resulting list would look like:

```
[{'path':/base_path/red/square, 'color':'red', 'shape':'square'},
{'path':/base_path/red/circle, 'color':'red', 'shape':'circle'},
{'path':/base_path/blue/triangle, 'color':'blue', 'shape':'triangle'},
...]
```

Obviously, 'path' should not be in the input list of elements.

miranda.utils.**publish_release_notes**(*style: str = 'md'*, *file: PathLike | StringIO | TextIO | None = None*) → str | None

Format release history in Markdown or ReStructuredText.

Parameters

- **style** (*{"rst", "md"}*) – Use ReStructuredText formatting or Markdown. Default: Markdown.
- **file** (*{os.PathLike, StringIO, TextIO}, optional*) – If provided, prints to the given file-like object. Otherwise, returns a string.

Returns
str, optional

Notes

This function is solely for development purposes.

miranda.utils.**single_item_list**(*iterable: Iterable*) → bool

Ascertain whether a list has exactly one entry.

See: https://stackoverflow.com/a/16801605/7322852

Parameters
iterable (*Iterable*)

Returns
bool

miranda.utils.**working_directory**(*directory: str | Path*) → None

Change the working directory within a context object.

This function momentarily changes the working directory within the context and reverts to the file working directory when the code block it is acting upon exits

Parameters
directory (*Union[str, Path]*)

Returns
None

miranda.validators module

miranda.validators.**url_validate**(*target: str*) → Match[str] | None

Validate whether a supplied URL is reliably written.

Parameters
target (*str*)

References

https://stackoverflow.com/a/7160778/7322852

3.2 Feedback

If you have any suggestions or questions about **Miranda** feel free to email me at smith.trevorj@ouranos.ca.

If you encounter any errors or problems with **Miranda**, please let me know! Open an Issue at the GitHub https://github.com/Ouranosinc/miranda main repository.

PYTHON MODULE INDEX

m

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W