ACT Cheat Sheet

Learn More About ACT at https://anl-digr.github.io/ACT/

ACT Introduction

The Atmospheric Commutity Toolkit (ACT) is a package for connecting Atmospheric data users to the data. Has the ability to download, read, and visualize multi-file datasets from multiple data sources. Currently, multi-panel timeseries plots are supported.

Installation

Installing ACT from source is the only way to get the latest updates and enhancement to the software that have no yet made it into a release. The latest source code for ACT can be obtained from the GitHub repository, https://github.com/ARM-DOE/ACT. Either download and unpack the zip file of the source code or use git to checkout the repository:

```
$ git clone https://github.com/ARM-DOE/ACT.git
```

- To install in your home directory, use:

```
$ python setup.py install –user
```

- To install for all users on Unix/Linux:

```
$ python setup.py build
$ sudo python setup.py install
```

Contact Information

ACT GitHub Issues Forum:
https://github.com/ARM-DOE/ACT/issues

Email:
atheisen@anl.gov
rjackson@anl.gov

Contributing

ACT is an open source community software project. Contributions to the package are welcomed from all users.

If you are planning on making changes that you would like included in ACT, forking the repository is highly recommended.

We welcome contributions for all uses of ACT, provided the code can be distributed under the BSD 3-clause license. A copy of this license is available in the LICENSE.txt file found at:

https://github.com/ARM-DOE/ACT/blob/master/LICENSE.txt

Getting Started

`>>> import act`	To import ACT.
`>>> print(act.__version__)`	Check version.

Corrections

```
>>> obj = act.corrections.ceil.correct_ciel(obj)
```

- This procedure corrects celiometer data by filling all zero and negative values of backscatter with fill_value and then converting the backscatter data into logarithmic space.

Discovery

```
>>> act.discovery.download_data(
        username, token, datastream, startdate,
        enddate, output=None)
```

- This programmatic interface allows users to query and automate machine-to-machine downloads of ARM data. This tool uses a REST URL and specific parameters (saveData, query), user ID and access token, a datastream name, a start date, and an end date, and data files matching the criteria will be returned to the user and downloaded.

- This will also eliminate the manual step of following a link in an email to download data. More information about the REST API and tools https://adc.arm.gov/armlive/#scripts

- To login/register for an access token: https://adc.arm.gov/armlive/livedata/home

Input and Output Data

```
>>> act_obj = act.io.armfiles.read_netcdf(
        filenames, concat_dim='time',
        return_None=False, **kwargs)
```

- Returns xarray.Dataset with stored data and metadata from a user-defined query of ARM-standard netCDF files from a single datastream.

```
>>> flag = act.io.armfiles.check_arm_standards(
        act_obj)
```

- Checks to see if an xarray dataset conforms to ARM standards.

```
>>> act.io.dataset.ACTAccessor(act_obj)
```

- The xarray accessor for ACT data structures. This adds functionality that includes storing the times and names of each file in the dataset. In addition, the datastream can be given a name and a site.

```
>>> act.io.csvfiles.read_csv(
        filename, sep=', ', engine='python',
        column_names=None, skipfooter=0,
        **kwargs)
```

- Returns an xarray.Dataset with stored data and metadata from user-defined query of CSV files.

```
>>> clean_dataset = act.io.clean.CleanDataset(
        act_obj)
```

- Class containing functions for cleaning dataset. More on the functions below after defining the clean dataset object.

```
>>> clean_dataset.clean_arm_qc(
        override_cf_flag=True,
        clean_units_string=True,
        correct_valid_min_max=True)
```

- Function to clean up xarray object QC variables.

```
>>> clean_dataset.clean_arm_state_variables(
        variables, override_cf_flag=True,
        clean_units_string=True, integer_flag=True
```

- Function to clean up state variables to use more CF style.

```
>>> clean_dataset.cleanup(
        cleanup_arm_qc=True,
        clean_arm_state_vars=None,
        handle_missing_value=True,
        link_qc_variables=True, **kwargs
```

- Wrapper method to automatically call all the standard methods for obj cleanup.

Plotting

Display

Class that contains the common attributes and routine between the differing Display classes.

```
>>> display = act.plotting.Display(
        obj, subplot_shape=(1, ), ds_name=None,
        subplot_kw=None, **kwargs)
```

```
>>> display.add_colorbar(
        mappable, title=None, subplot_index=(0, ))
```

- Adds a colorbar.

```
>>> display.add_subplots(
        subplot_shape=(1, ), subplot_kw=None,
        **kwargs)
```

- Adds subplot to display object.

```
>>> display.assign_to_figure_axis(fig, ax)
```

- This assigns the Display to a specific figure and axis.

```
>>> display.put_display_in_subplot(
        display, subplot_index))
```

- This will place a Display object into a specific subplot.

TimeSeriesDisplay

This subclass contains routines that are specific to plotting time series plots from data.

```
>>> dis = act.plotting.TimeSeriesDisplay(
        obj, subplot_shape=(1, ), ds_name=None,
        **kwargs)
```

```
>>> dis.plot(field[, ...])
```

- Makes a timeseries plot.

```
>>> display.plot_barbs_from_spd_dir(dir_field[, ...]
```

- This procedure will make a wind barb plot timeseries.

```
>>> dis.plot_barbs_from_u_v(u_field, v_field
        [, ...])
```

- This function will plot a wind barb timeseries from u and v wind data. If pres_field is given, a height a time-height series will be plotted from 1-D wind data.

```
>>> dis.plot_time_height_xsection_from_1d_data(
        data_field, pres_field[, ...])
```

- This will plot a time-height cross section from 1D datasets using nearest neighbor interpolation on a regular time by height grid.

```
>>> dis.time_height_scatter(
        data_field=None[, ...])
```

- Create a time series plot of altitude and data variable with color also indicating value with a color bar.

SkewTDisplay

A class for making Skew-T plots.

```
>>> display = act.plotting.SkewTDisplay(
        obj, subplot_shape=(1, ), ds_name=None,
        **kwargs)
```

```
>>> display.add_subplots(
        subplot_shape=(1, ), **kwargs)
```

- Adds subplots to the Display object. The current figure in the object will be deleted and overwritten.

```
>>> display.plot_from_spd_and_dir(
        spd_field, dir_field, p_field, t_field,
        td_field[, ...])
```

- This plot will make a sounding plot from wind data that is given in speed and direction.

ACT Cheat Sheet

Learn More About ACT at https://anl-digr.github.io/ACT/

Plotting

SkewTDisplay Continued

```
>>> display.plot_from_u_and_v(
        u_field, v_field, p_field, t_field,
        td_field[, ...])
```
- This function will plot a Skew-T from a sounding dataset. The wind data must be given in u and v.

WindRoseDisplay

A class for handing wind rose plots..

```
>>> display = act.plotting.WindRoseDisplay(
        obj, subplot_shape=(1, ), ds_name=None,
        **kwargs)

>>> display.plot(dir_field, spd_field[, ...])
```
- Makes the wind rose plot from the given dataset.

XSectionDisplay

Plots cross sections of multidimensional datasets.

```
>>> display = act.plotting.XSectionDisplay(
        obj, subplot_shape=(1, ), ds_name=None,
        **kwargs)

>>> display.plot_xsection(dsname, varname[, ...])
```
- This function plots a cross section whose x and y coordinates are specified by the variable names either provided by the user or automatically detected by xarray.

```
>>> display.plot_xsection_map(
        dsname, varname[, ...])
```
- Plots a cross section of 2D data on a geographical map.

Retrievals

```
>>> ds = act.retrievals.calculate_stability_indicies(
        ds, temp_name='temperature',
        td_name='dewpoint_temperature',
        p_name='pressure', moving_ave_window=0)
```
- Calculates stability indices and adds it to the data set.

Utilities

```
>>> dates = act.utils.datetime_utils.dates_between(
        sdate, edate)
```
- Ths procedure returns all of the dates between sdate and edate.

```
>>> time, data = act.utils.data_utils.add_in_nan(
        time, data)
```
- This procedure adds in NaNs for given time periods in time when there is no corresponding data available. This is useful for timeseries that have irregular gaps in data.

```
>>> val = act.utils.data_utils.get_missing_value(
        variable, default=-9999,
        add_if_missing_in_obj=False,
        use_FillValue=False, nodefault=False)
```
- Method to get missing value from missing_value or _FillValue attribute.

```
>>> ds = act.utils.data_utils.assign_coordinates(
        ds, coord_list)
```
- This procedure will create a new ACT dataset whose coordinates are designated to be the variables in a given list.