# ACT Cheat Sheet

Learn More About ACT at https://arm-doe.github.io/ACT/

## ACT Introduction

The Atmospheric data Community Toolkit (ACT) is an open source Python toolkit for working with atmospheric time-series datasets of varying dimensions. The toolkit has functions for every part of the scientific process; discovery, IO, quality control, corrections, retrievals, visualization, and analysis. It is a community platform for sharing code with the goal of reducing duplication of effort and better connecting the science community with programs such as the Atmospheric Radiation Measurement (ARM) User Facility.

## 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, use:

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

To install ACT using Anaconda or Miniconda, create an environment and activate it:

- Then create a conda environment:

```
$ conda create -n act python=3.9
```

- Activate the ACT environment:

```
$ conda activate act
```

- Then install ACT:

```
$ conda install -c conda-forge act-atmos
```

## Contact Information

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

**Email:**

atheisen@anl.gov
mgrover@anl.gov
zsherman@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.correct_ceil(obj)
- This procedure corrects celiometer data.

>>> obj = act.corrections.correct_dl(obj)
- This procedure corrects doppler lidar data.

>>> obj = act.corrections.correct_mpl(obj)
- This procedure corrects MPL data.

>>> obj = act.corrections.correct_rl(obj)
- This procedure corrects raman lidar data.

>>> obj = act.corrections.correct_wind(obj)
- This procedure corrects wind speed and direction. for ship motion based on equations from NOAA tech.

## Discovery

>>> act.discovery.download_data(
username, token, datastream, startdate, enddate[,...])
- 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 can be found on ARM Live: https://adc.arm.gov/armlive/#scripts
- To login/register for an access token: https://adc.arm.gov/armlive/livedata/home

>>> act.discovery.croptype(lat, lon, year)
- Function for working with the CropScape API to get a crop type based on the lat,lon, and year entered.

>>> act.discovery.download_noaa_psl_data(
site, instrument[, ...])
- Function to download data from the NOAA PSL Profiler Network Data Library https://psl.noaa.gov/data/obs/datadisplay/

>>> act.discovery.get_airnow_forecast(token[,...])
- This tool will get current or historical AQI values and categories for a reporting area by either Zip code or Lat/Lon coordinate.

>>> act.discovery.get_airnow_obs(token[,...])
- This tool will get current or historical obs AQI values and categories for a reporting area by either Zip code or Lat/Lon coordinate.

>>> act.discovery.get_asos(time_window[,...])
- Returns all of the station observations from the Iowa Mesonet from either a given latitude and longitude window or a given station code.

>>> act.discovery.get_airnow_bounded_obs(
token[,...])
- Get AQI values or data concentrations for a specific date and time range and set of parameters within a geographic area of intrest https://docs.airnowapi.org/

## Input and Output Data

>>> act_obj = act.io.create_obj_from_arm_dod(
proc, set_dims[,...])
- Queries the ARM DOD api and builds an object based on the ARM DOD and the dimension sizes that are passed in.

>>> flag = act.io.check_arm_standards(act_obj)
- Checks to see if an xarray dataset conforms to ARM standards.

>>> act_obj = act.io.read_netcdf(filenames[,...])
- Returns xarray.Dataset with stored data and metadata from a user-defined query of ARM-standard netCDF files from a single datastream.

>>> act.io.read_csv(filename[,...])
- Returns an xarray.Dataset with stored data and metadata from user-defined query of CSV files.

>>> act.io.read_sigma_mplv5(filename[,...])
- Returns xarray.Dataset with stored data and metadata from a user-defined SIGMA MPL V5 files.

>>> act.io.read_gml(filename[,...])
- Function to call or guess what reading NOAA GML daga routine to use.

>>> act.io.read_gml_co2(filename[,...])
- Function to read carbon dioxide data from NOAA GML.

>>> act.io.read_gml_halo(filename[,...])
- Function to read Halocarbon data from NOAA GML.

>>> act.io.read_gml_met(filename[,...])
- Function to read meteorological data from NOAA GML.

>>> act.io.read_gml_ozone(filename[,...])
- Function to read ozone data from NOAA GML.

>>> act.io.read_gml_radiation(filename)
- Function to read radiation data from NOAA GML.

>>> act.io.read_hk_file(filename)
- This procedure will read in an SP2 housekeeping file

>>> act.io.read_psl_parsivel(filename)
- Returns xarray.Dataset with stored data and metadata from a defined NOAA PSL parsivel.

>>> act.io.read_psl_wind_profiler(filename[,...])
- Returns xarray.Dataset with stored data and metadata from a user-defined NOAA PSL wind profiler file.

>>> act.io.read_psl_wind_profiler_temperature(
filename)
- Returns xarray.Dataset with stored data and metadata from a user-defined NOAA PSL wind profiler temperature file.

>>> act.io.read_sp2(filename[,...])
- Loads a binary SP2 raw data file and returns all of the wave forms into an xarray Dataset.

>>> act.io.read_sp2_dat(filename[,...])
- This reads the .dat files that generate the intermediate parameters used by the Igor processing. Wildcards are supported.

# ACT Cheat Sheet

Learn More About ACT at https://arm-doe.github.io/ACT/

## 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.

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

```
>>> display.plot(field[, ...])
```
- Makes a timeseries plot.

```
>>> display.plot_barbs_from_spd_dir(dir_field[, ...]
```
- This procedure will make a wind barb plot timeseries.

```
>>> display.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.

```
>>> display.time_height_scatter(
        data_field, dsname[, ...])
```
- 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.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.

```
>>> 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.

## Plotting Continued

### 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.

```
>>> display.plot_data(dir_field, spd_field, data_field)
```
- Makes a data rose plot in line or boxplot form from the given data.

### 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.

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

### GeographicPlotDisplay

A class for making geographic tracer plot of aircraft, ship or other moving platform plot.

```
>>> display = act.plotting.GeographicPlotDisplay(
        obj, ds_name=None, **kwargs)
```

```
>>> display.geoplot(data_field[, ...])
```
- Creates a latitude and longitude plot of a time series data set with data values indicated by color.

### HistogramDisplay

Class used to make histogram plots.

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

```
>>> display.plot_heatmap(x_field, y_field[, ...])
```
- This procedure will plot a heatmap of a histogram from 2 variables.

```
>>> display.plot_stacked_bar_graph(field[, ...])
```
- This procedure will plot a stacked bar graph of a histogram.

```
>>> display.plot_stairstep_graph(field[, ...])
```
- This procedure will plot a stairstep plot of a histogram.

## QC

```
>>> act.qc.add_dqr_to_qc(obj[,...])
```
- Function to query the ARM DQR web service control test for reports and add as a new quality to ancillary quality control variable.

```
>>> act.qc.apply_supplemental_qc(obj[,...])
```
- Apply flagging from supplemental QC file by adding new QC tests.

## QC Continued

Classes listed in blue have functions that can be found in ACT's documentation: https://arm-doe.github.io/ACT/API/index.html

### CleanDataset

Class for cleaning up QC variables to standard cf-compliance.

```
>>> act.qc.CleanDataset(obj)
```

### QCFilter

Class for building quality control variables containing arrays for filtering data based on a set of test condition typically based on the values in the data fields.

```
>>> act.qc.QCFilter(obj)
```

### QCTests

Method to perform a time series comparison test between two Xarray Datasets to detect a shift in time based on two similar variables.

```
>>> act.qc.QCTests(obj)
```

## Retrievals

```
>>> act.retrievals.calculate_stability_indicies(
        ds[,...])
```
- Calculates stability indices and adds it to the data set.

```
>>> act.retrievals.aeri2irt(
        aeri_ds[,...])
```
- This function will integrate over the correct wavenumber values to produce the effective IRT temperature.

```
>>> act.retrievals.calculate_pbl_liu_liang(
        ds[,...])
```
- Function for calculating the PBL height from a radiosonde profile using the Liu-Liang 2010 technique.

```
>>> act.retrievals.calc_sp2_diams_masses(
        ds[,...])
```
- Calculates the scattering and incandescence diameters/BC masses for each particle.

```
>>> act.retrievals.calculate_precipitable_water(
        ds[,...])
```
- Function to calculate precipitable water vapor from ARM sondewnpn b1 data.

## Utilities

```
>>> ds = act.utils.create_pyart_obj(obj)
```
- Produces a Py-ART object from an ACT object.

```
>>> ds = act.utils.calculate_dqr_times(obj[,...])
```
- Function to retrieve start and end times of missing or bad data.

```
>>> ds = act.utils.height_adjusted_pressure(obj[,...])
```
- Converts pressure for change in height.

```
>>> ds = act.utils.convert_units(data, in_units
        out_units)
```
- Converts units of a data array.

```
>>> ds = act.utils.accumulate_precip(obj, variable)
```
- Accumulate rain rates from an act object and insert variable back into act object.