EUFAR
European Facility For Airborne Research

EUFAR FP7

N6SP - Standards and Protocols

EGADS Documentation

Version 0.2.0

Last Updated: February 22, 2011

Contents

Chapter 1

Introduction

The EGADS (EUFAR General Airborne Data-processing Software) package is a suite of processing software designed to analyze a wide range of airborne atmospheric science data. EGADS purpose is to provide a benchmark for airborne data-processing with its community-provided algorithms, and to act as a reference and to provide guidance to researchers with its open-source design and well-documented processing routines.

Python was chosen for development of EGADS for its straightforward syntax and flexiblity between multiple systems. Thus, users interact with data processing algorithms using the Python command-line or via Python scripts. Central to EGADS is a data type that encapsulates data and metadata into one object. This simplifies the housekeeping of data and metadata and allows it to be easily passed between algorithms and data files.

Chapter 2

Installation

The latest version of EGADS can be obtained from http://eufar-egads.googlecode.com

2.1 Prerequisites

Building EGADS requires the following packages:

Python 2.5 or newer. Available at http://www.python.org/.

numpy 1.3.0 or newer. Available at http://numpy.scipy.org/.

scipy 0.6.0 or newer. Available at http://www.scipy.org/.

Python netCDF4 libraries 0.8.2. Available at http://code.google.com/p/netcdf4-python/.

2.2 Optional Packages

The following are useful when using or compiling EGADS :

IPython An optional package which simplifies Python command line usage (http://ipython.scipy.org). IPython is an enhanced interactive Python shell which supports tab-completion, debugging, history, etc.

2.3 Installation

Since EGADS is a pure Python distribution, it does not need to be built. However, to use it, it must be installed to a location on the Python path. To install EGADS , type `python setup.py install` from the command line. To install to a user-specified location, type `python setup.py install --prefix=$MYDIR`.

2.4 Testing

To test EGADS after it is installed, run the following commands in Python:

```
>>> import egads
>>> egads.test()
```

Chapter 3

Tutorial

3.1 Command-line usage

The simplest way to start working with EGADS is to run it from the Python command line. To load egads into the Python namespace, simply import it:

```
>>> import egads
```

You may then begin working with any of the algorithms and functions contained in EGADS .

3.1.1 Exploring egads

There are several useful methods to explore the routines contained in EGADS . The first is using the Python built-in `dir()` command:

```
>>> dir(egads)
```

returns all the classes and subpackages contained in EGADS . EGADS follows the naming conventions from the Python Style Guide (http://www.python.org/dev/peps/pep-0008), so classes are always `MixedCase`, functions and modules are generally `lowercase` or `lowercase_with_underscores`. As a further example,

```
>>> dir(egads.input)
```

would returns all the classes and subpackages of the `egads.input` module.

Another way to explore EGADS is by using tab completion, if supported by your Python installation. Typing

```
>>> egads.
```

then hitting TAB will return a list of all available options.

Python has built-in methods to display documentation on any function known as docstrings. The easiest way to access them is using the `help()` function:

```
>>> help(egads.input.NetCdf)
```

will return all methods and their associated documentation for the NetCdf class.

3.1.2 Working with generic text files

EGADS provides the `egads.input.EgadsFile()` class as a simple wrapper for interacting with generic text files. `EgadsFile()` can read, writeand display data from text files, but does not have any tools for automatically formatting input or output data.

Opening

To open a text file the `EgadsFile()` class, use the `open(pathname, permissions)` method:

```
>>> import egads
>>> f = egads.input.EgadsFile()
>>> f.open('/pathname/filename.txt','r')
```

Valid values for permissions are:

- `r` – Read: opens file for reading only. Default value if nothing is provided.
- `w` – Write: opens file for writing, and overwrites data in file.
- `a` – Append: opens file for appending data.
- `r+` – Read and write: opens file for both reading and writing.

File Manipulation

The following methods are available to control the current position in the file and display more information about the file.

- `f.display_file()` – Prints contents of file out to standard output.
- `f.get_position()` – Returns current position in file as integer.
- `f.seek(location, from_where)` – Seeks to specified location in file. `location` is an integer specifying how far to seek. Valid options for `from_where` are 'b' to seek from beginning of file, 'c' to seek from current position in file and 'e' to seek from the end of the file.
- `f.reset()` – Resets position to beginning of file.

Reading Data

Reading data is done using the `read(size)` method on a file that has been opened with `r` or `r+` permissions:

```
>>> import egads
>>> f = egads.input.EgadsFile()
>>> f.open('myfile.txt','r')
>>> single_char_value = f.read()
>>> multiple_chars = f.read(10)
```

If the `size` parameter is not specified, the `read()` function will input a single character from the open file. Providing an integer value n as the `size` parameter to `read(size)` will return n characters from the open file.

Data can be read line-by-line from text files using `read_line()`:

```
>>> line_in = f.read_line()
```

Writing Data

To write data to a file, use the `write(data)` method on a file that has been opened with `w`, `a` or `r+` permissions:

```
>>> import egads
>>> f = egads.input.EgadsFile()
>>> f.open('myfile.txt','a')
>>> data = 'Testing output data to a file.\n This text will appear on the 2nd line.'
>>> f.write(data)
```

Closing

To close a file, simply call the `close()` method:

```
>>> f.close()
```

3.1.3 Working with CSV files

`egads.input.EgadsCsv()` is designed to easily input or output data in CSV format. Data input using `EgadsCsv()` is separated into a list of arrays, which each column a separate array in the list.

Opening

To open a text file the `EgadsCSV()` class, use the `open(pathname, permissions, delimiter, quotechar)` method:

```
>>> import egads
>>> f = egads.input.EgadsFile()
>>> f.open('/pathname/filename.txt','r',',','"')
```

Valid values for permissions are:

- `r` – Read: opens file for reading only. Default value if nothing is provided.
- `w` – Write: opens file for writing, and overwrites data in file.
- `a` – Append: opens file for appending data.
- `r+` – Read and write: opens file for both reading and writing.

The `delimiter` argument is a one-character string specifying the character used to separate fields in the CSV file to be read; the default value is ','. The `quotechar` argument is a one-character string specifying the character used to quote fields containing special characters in the CSV file to to be read; the default value is '“'.

File Manipulation

The following methods are available to control the current position in the file and display more information about the file.

- `f.display_file()` – Prints contents of file out to standard output.
- `f.get_position()` – Returns current position in file as integer.
- `f.seek(location, from_where)` – Seeks to specified location in file. `location` is an integer specifying how far to seek. Valid options for `from_where` are 'b' to seek from beginning of file, 'c' to seek from current position in file and 'e' to seek from the end of the file.
- `f.reset()` – Resets position to beginning of file.

Reading Data

Reading data is done using the `read(lines, format)` method on a file that has been opened with 'r' or 'r+' permissions:

```
>>> import egads
>>> f = egads.input.EgadsCsv()
>>> f.open('mycsvfile.csv','r')
>>> single_line_as_list = f.read(1)
>>> all_lines_as_list = f.read()
```

`read(lines, format)` returns a list of the items read in from the CSV file. The arguments `lines` and `format` are optional. If `lines` is provided, `read(lines, format)` will read in the specified number of lines, otherwise it will read the whole file. `format` is an optional list of characters used to decompose the elements read in from the CSV files to their proper types. Options are:

- i – int
- f – float
- l – long
- s – string

Thus to read in the line:
`FGBTM,20050105T143523,1.5,21,25` the command to input with proper formatting would look like this:

```
>>> data = f.read(1, ['s','s','f','f'])
```

Writing Data

To write data to a file, use the `write(data)` method on a file that has been opened with 'w', 'a' or 'r+' permissions:

```
>>> import egads
>>> f = egads.input.EgadsCsv()
>>> f.open('mycsvfile.csv','a')
>>> titles = ['Aircraft ID','Timestamp','Value1','Value2','Value3']
>>> f.write(titles)
```

where the `data` parameter is a list of values. This list will be output to the CSV, with each value separated by the delimiter specifed when the file was opened (default is ',').

To write multiple lines out to a file, `writerows(data)` is used:

```
>>> data = [['FGBTM','20050105T143523',1.5,21,25],['FGBTM','20050105T143524',1.6,20,25.6]]
>>> f.writerows(data)
```

Closing

To close a file, simply call the `close()` method:

```
>>> f.close()
```

3.1.4 Working with NetCDF files

EGADS provides two classes to work with NetCDF files. The simplest, `egads.input.NetCdf()`, allows simple read/write operations to NetCDF files. The other, `egads.input.EgadsNetCdf()`, is designed to interface with NetCDF files conforming to the N6SP data and metadata regulations. This class directly reads or writes NetCDF data using instances of the `EgadsData` class.

Opening

To open a NetCDF file, simply create a `egads.input.NetCdf()` instance and then use the `open(pathname, permissions)` command:

```
>>> import egads
>>> f = egads.input.NetCdf()
>>> f.open('/pathname/filename.nc','r')
```

Valid values for permissions are:

- `r` – Read: opens file for reading only. Default value if nothing is provided.
- `w` – Write: opens file for writing, and overwrites data in file.
- `a` – Append: opens file for appending data.
- `r+` – Same as 'a'.

Getting info

- `f.get_dimensions()` – returns list of all dimensions and their sizes
- `f.get_dimensions(var_name)` – returns list of all dimensions for `var_name`
- `f.get_attributes()` – returns a list of all top-level attributes
- `f.get_attributes(var_name)` – returns list of all attributes attached to `var_name`
- `f.get_variables()` – returns list of all variables
- `f.get_filename()` – returns filename for currently opened file

Reading data

To read data from a file, use the `read_variable()` function:

```
>>> data = f.read_variable(var_name, input_range)
```

where `var_name` is the name of the variable to read in, and `input_range` (optional) is a XXXXX

If using the `egads.input.NetCdf()` class, an array of values contained in `var_name` will be returned. IF using the `egads.input.EgadsNetCdf()` class, an instance of the `EgadsData()` class will be returned containing the values and attributes of `var_name`.

Writing data

The following describe how to add dimenisons or attributes to a file.

- `f.add_dim(dim_name, dim_size)` – add dimension to file
- `f.add_attribute(attr_name, attr_value)` – add attribute to file
- `f.add_attribute(attr_name, attr_value, var_name)` – add attribute to `var_name`

Data can be output to variables using the `write_variable()` function as follows:

```
>>> f.write_variable(data, var_name, dims, type)
```

where `var_name` is a string for the variable name to output, `dims` is a tuple of dimension names (not needed if the variable already exists), and `type` is the data type of the variable. The default value is *double*, other valid options are *float*, *int*, *short*, *char* and *byte*.

If using `egads.input.NetCdf()`, values for `data` passed into `write_variable` must be scalar or array. Otherwise, if using `egads.input.EgadsNetCdf()`, an instance of `EgadsData()` must be passed into `write_variable`. In this case, any attributes that are contained within the `EgadsData()` instance are applied to the NetCDF variable as well.

Closing

To close a file, simply use the `close()` method:

```
>>> f.close()
```

3.1.5 Working with algorithms

Algorithms in EGADS are stored in the `egads.algorithms` module, and separated into sub-modules by category (microphysics, thermodynamics, radiation, etc). Each algorithm follows a standard naming scheme, using the algorithm's purpose and source:

```
{calculated-parameter}_{detail}_{source}
```

For example, an algorithm which calculates static temperature, which was provided by CNRM would be named:

```
temp_static_cnrm
```

Getting algorithm information

There are several methods to get information about each algorithm contained in EGADS . The EGADS Algorithm Handbook is available for easy reference outside of Python. In the handbook, each algorithm is described in detail, including a brief algorithm summary, descriptions of algorithm inputs and outputs, the formula used in the algorithm, algorithm source and links to additional references. The handbook also specifies the exact name of the algorithm as defined in EGADS . The handbook can be found on the EGADS website, and in the \doc directory packaged with EGADS .

Within Python, usage information on each algorithm can be found using the `help()` command:

```
>>> help(egads.algorithms.thermodynamics.velocity_tas_cnrm)

>>>Help on function velocity_tas_cnrm in module
   egads.algorithms.thermodynamics.velocity_tas_cnrm:

velocity_tas_cnrm(T_s, P_s, dP, cpa, Racpa)
    FILE        velocity_tas_cnrm.py

    VERSION     $Revision: 38 $

    CATEGORY    Thermodynamics

    PURPOSE     Calculate true airspeed

    DESCRIPTION Calculates true airspeed based on static temperature, static pressure
                and dynamic pressure using St Venant's formula.

    INPUT       T_s         vector  K or C      static temperature
                P_s         vector  hPa         static pressure
```

```
                dP          vector  hPa          dynamic pressure
                cpa         coeff.  J K-1 kg-1   specific heat of air
                                                 (dry air is 1004 J K-1 kg-1)
                Racpa       coeff.  ()           R_a/c_pa

    OUTPUT      V_p         vector  m s-1        true airspeed

    SOURCE      CNRM/GMEI/TRAMM

    REFERENCES  "Mecanique des fluides", by S. Candel, Dunod.

                 Bulletin NCAR/RAF Nr 23, Feb 87, by D. Lenschow and
                 P. Spyers-Duran
```

Calling algorithms

Algorithms in EGADS generally accept and return arguments of `EgadsData` type, unless otherwise noted. This has the advantages of constant typing between algorithms, and allows metadata to be passed along the whole processing chain. To call an algorithm, simply pass in the required arguments, in the order they are described in the algorithm help function. An algorithm call, using the `velocity_tas_cnrm` in the previous section as an example, would therefore be the following:

```
>>> V_p = egads.algorithms.thermodynamics.velocity_tas_cnrm(T_s, P_s, dP, cpa, Racpa)
```

where the arguments `T_s`, `P_s`, `dP`, etc are all assumed to be previously defined in the program scope.

3.2 Scripting

The recommended method for using EGADS is to create script files, which are extremely useful for common or repetitive tasks. This can be done using a text editor of your choice. The example script belows shows the calculation of density for all NetCDF files in a directory.

example.py

```
#!/usr/bin/env python

# import egads package
import egads
# import thermodynamic module and rename to simplify usage
import egads.algorithms.thermodynamics as thermo

# get list of all NetCDF files in 'data' directory
filenames = egads.get_file_list('data/*.nc')

f = egads.input.EgadsNetCdf()   # create EgadsNetCdf instance

for name in filenames:          # loop through files
```

```
    f.open(name, 'a')              # open NetCdf file with append permissions

    T_s = f.read_variable('T_t') # read in static temperature
    P_s = f.read_variable('P_s') # read in static pressure from file

    rho = thermo.density_dry_air_cnrm(P_s, T_s)  # calculate density

    f.write_variable(rho, 'rho', ('Time',))      # output variable

    f.close()                                    # close file
```

3.2.1 Scripting Hints

When scripting in Python, there are several

Importance of white space

Python differs from C++ and FORTRAN in how loops or nested statements are signified. Whereas C++ uses brackets ('{' and '}') or FORTRAN uses `end` statements to signify the end of a nesting, Python uses white space. Thus, for statements to nest properly, they must be set at the proper depth. As long as the document is consistent, the number of spaces used doesn't matter, however, most conventions call for 4 spaces to be used per level. See below for examples:

FORTRAN

```
X = 0
DO I = 1,10
  X = X + I
  PRINT I
END DO
PRINT X
```

Python

```
x = 0
for i in range(1,10):
    x = x + i
    print i
print x
```