PyeDNA Documentation

Release 1.01

Eric Strong

Sep 01, 2017

Contents

1 1. Introduction **3**
- 1.1 Disclaimer . . . 3
- 1.2 Package Organization . . . 3
- 1.3 Basic Examples . . . 4

2 2. Getting Started **5**
- 2.1 Installation . . . 5
- 2.2 eDNA Requirements . . . 5
- 2.3 Python Requirements . . . 5
- 2.4 Python Version Support . . . 6
- 2.5 eDNA Version Support . . . 6
- 2.6 Importing PyeDNA . . . 6

3 3. Configuration Information **7**
- 3.1 Service Information . . . 7
- 3.2 Tag Information . . . 7
- 3.3 Tag Picker . . . 8

4 4. Pulling Data **9**
- 4.1 Types of Data Pulls . . . 9
- 4.2 eDNA Data Compression . . . 9
- 4.3 GetHist . . . 10
- 4.4 GetMultipleTags . . . 11

5 5. Pushing Data **13**
- 5.1 Serv Capabilities . . . 13

6 Change Log **15**
- 6.1 Version 0.14 . . . 15
- 6.2 Version 0.15 . . . 15
- 6.3 Version 0.16 . . . 16
- 6.4 Version 0.17 . . . 16
- 6.5 Version 0.17 . . . 16
- 6.6 Version 0.18 . . . 16
- 6.7 Version 1.01 . . . 16

7 Indices and tables **17**

PyeDNA ("pie-dee-en-ay") is a Python wrapper library for the C++ EzDnaApi, written for data analysts who wish to work with eDNA data in the context of a pandas DataFrame. By converting eDNA data into a DataFrame, data analysis can be accomplished using familiar tools like scikit-learn, statsmodels, etc. New functions will be added upon request.

CHAPTER 1

1. Introduction

PyeDNA ("pie-dee-en-ay") is a Python wrapper library for the C++ EzDnaApi, written for data analysts who wish to work with eDNA data in the context of a pandas DataFrame. By converting eDNA data into a DataFrame, data analysis can be accomplished using familiar tools like scikit-learn, statsmodels, etc. New functions will be added upon request.

Disclaimer

PyeDNA is a wrapper library for the API of a data historian called eDNA. eDNA is developed by InStepSoftware, LLC (http://www.instepsoftware.com/), who holds all rights to the eDNA software. PyeDNA does not contain any proprietary code, and is merely a wrapper for functions that must be obtained from a legal, licensed version of EzDnaApi.dll.

This is fan-supported project and is not affiliated in any way with InStepSoftware, LLC. The maintainer enjoys working with eDNA and wishes them the best. :)

Package Organization

PyeDNA is organized into several namespaces, including:

- calc_config
- ezdna
- serv

The namespace of most interest to the typical user will be the *ezdna* namespace, which contains methods that are meant to translate the eDNA API to Pythonic syntax and common libraries, such as pandas. For instance, all the data pulling and configuration information functions are in this namespace.

The other two namespaces, calc_config and serv, contain more specialized functions. Serv contains functions from the EzDNAServAPI that are meant to push data into eDNA. These functions are not entirely converted to familiar syntax, behaving in a more low-level fashion. Calc_config is a namespace meant for parsing a CM.DB file (a sqlite database),

which each eDNA service contains. Calculations defined in eDNA may be difficult to parse, and this class is meant to determine which tags are located in which calculations, to determine a dependency structure.

Basic Examples

All of the core data pulling functions are located in the GetHist function, which will return a pandas DataFrame with the timestamp, value, and status columns. For example, the following code will pull snap data from TESTPNT1 over a 30-second interval:

> import pyedna.ezdna as dna

> tag = "TESTSITE.TESTSERVICE.TESTPNT1" # format site.service.tag

> start = "12/01/16 01:01:01" # format mm/dd/yy hh:mm:ss

> end = "01/03/17 01:01:01" # format mm/dd/yy hh:mm:ss

> period = "00:00:30" # format hh:mm:ss

> df = dna.GetHist(tag, start, end, period=period, mode="snap")

Raw data may be obtained from TESTPNT1 using:

> df = dna.GetHist(tag, start, end, mode="raw")

Other supported pull types include Average, Interpolated, Max, and Min. Please refer to eDNA documentation for more description about these pull types.

Multiple tags can be pulled (in Raw mode) at the same time using:

> tags = ["TESTSITE.TESTSERVICE.TESTPNT1", "TESTSITE.TESTSERVICE.TESTPNT2", "TESTSITE.TESTSERVICE.TESTPNT3", "TESTSITE.TESTSERVICE.TESTPNT4"]

> df = dna.GetMultipleTags(tags, start, end)

A list of connected services may be obtained using GetServices:

> services = dna.GetServices()

A list of point information for a given service can be found using GetPoints:

> points = dna.GetPoints("TESTSITE.TESTSERVICE")

CHAPTER 2

2. Getting Started

Installation

If Python is already installed on your computer, PyeDNA can be installed using PyPI by opening a command window and typing:

pip install pyedna

Upgrading to a new version of pyedna can be accomplished by:

pip install pyedna –upgrade

The source code of pyedna is hosted on GitHub at:

https://github.com/drericstrong/pyedna

eDNA Requirements

PyeDNA currently requires that a legal, licensed version of the EzDnaApi be located in the following directory:

C:Program Files (x86)eDNAEzDnaApi64.dll

If your EzDNAApi is in a different location, each namespace contains a method called LoadDll which can be used to specify the correct location:

LoadDll("CORRECT_LOCATION")

Python Requirements

Required libraries: numba, numpy, pandas

A requirements.txt document is located in the GitHub repository, and all package requirements can be installed using the following line in a command window:

pip install -r requirements.txt

Numba is required to significantly speed up the base-level data pull, and numpy and pandas are used for ease of data processing. It is very unlikely that these requirements will change in the future.

Python Version Support

Currently, PyeDNA only supports Python 3.2+ and is not compatible with Python 2. Testing confirms that PyeDNA will not work on Python 2 without some adjustments to the codebase. If this is important to you, please make a pull request at:

https://github.com/drericstrong/pyedna

The package maintainer welcomes collaboration.

eDNA Version Support

Only the 64-bit version of the eDNA API is supported in the current release. I am having trouble getting the 32-bit version to work. It may be an issue with my particular API file. Again, if 32-bit support is important to you, please contact me for collaboration.

Importing PyeDNA

PyeDNA is usually imported into a script using the following line:

import pyedna.ezdna as dna

Warning- since pyedna is connection-based, importing PyeDNA will always have direct side effects. When the module is imported, PyeDNA will attempt to connect to all available eDNA services. If none are available, a warning will be thrown, and the user's eDNA connection should be checked. If services are available, the number of available services will be printed to the console (the maintainer apologizes for this intrusion, but it was determined to be necessary to provide visibility for connection issues unrelated to PyeDNA behavior).

CHAPTER 3

3. Configuration Information

PyeDNA contains a number of functions which allow the user to pull configuration information from current eDNA services and points. These functions are located in the pyedna.ezdna namespace.

Warning- One of the most common mistakes when using PyeDNA is not to specify the full eDNA tag when using the module. Unless otherwise specified, tags should always be specified by their full Site.Service.Tag designation.

All code in this section assumes that PyeDNA has been imported using:

import pyedna.ezdna as dna

Service Information

eDNA contains a number of services, which each contain many tags. When PyeDNA is imported, it attempts to connect to all available services and will print the number of available eDNA services.

A list of all connection services can be obtained using:

dna.GetServices()

The above function will return a pandas DataFrame with the following columns:

- Name
- Description
- Type
- Status

Tag Information

Each service contains a number of tags, which define a block of time-based data storage. Connecting to a service allows access to all of its tags, which can be found using the following command:

dna.GetPoints("EDNA_SERVICE")

The above function will return a pandas DataFrame with all the points from the eDNA service, also including information such as:

- Tag
- Current Value
- Current Time
- Current Status
- Description
- Units

More specific information about a single point can be obtained using:

dna.GetRTFull("SITE.SERVICE.TAG")

The tag description alone can be found by:

dna.GetTagDescription("SITE.SERVICE.TAG")

Determining if a tag exists in any connected service can be accomplished by:

dna.DoesIDExist("SITE.SERVICE.TAG")

The above function will return either TRUE or FALSE. Ensure that proper spelling and the full Site.Service.Tag format is used.

Tag Picker

A dialog box containing the native eDNA "tag picker" can be brought up using:

dna.SelectPoint()

Unfortunately, only the single point version is supported at this time. Support for multiple tags is expected to be available in the future.

CHAPTER 4

4. Pulling Data

Data from each defined eDNA tag can be obtained by the functions in this section.

Warning- One of the most common mistakes when using PyeDNA is not to specify the full eDNA tag when using the module. Unless otherwise specified, tags should always be specified by their full Site.Service.Tag designation.

Danger- Please read the data compression section to understand what is actually happening when data is pulled in "raw" mode- it **will** affect your data analysis.

All code in this section assumes that PyeDNA has been imported using:

import pyedna.ezdna as dna

Types of Data Pulls

As defined by eDNA, several different types of data pulls may be accomplished:

- Avg: Finds the arithmetic mean of values over a window defined by the time span.
- Interp: Interpolates values over a window defined by the time span.
- Min: Finds the minimum value over a window defined by the time span.
- Max: Finds the maximum value over a window defined by the time span.
- Raw: Pulls data exactly as it is stored in the database (read data compression below)
- Snap: Finds the last data point over a window defined by the time span.

PyeDNA provides functionality for all of these methods.

eDNA Data Compression

The eDNA database only stores data points when either the value or status of the point changes. This allows the data files to be compressed, which is advantageous for transfer over a low-speed or expensive medium. However, this

compression presents some issues for data analysis that the user must be aware of.

First, data gaps may occur over the time period if data transfer is interrupted in some way. These data gaps may be hard to notice in practice, especially if the user is pulling data with the "Snap" method. Since "Snap" will find the last data point at each time window, the "last" data point will be the data point right before the data gap. This causes a "flat-lining" behavior that is usually obvious if the data gap is large enough. It is strongly recommended that the user implement some kind of gap-detection algorithm if gaps are frequent and "Snap" mode is being used.

Second, data pulled using "Raw" mode is not appropriate for many types of statistical analysis. Since "Raw" mode pulls *compressed* data as it is actually stored in the database, the frequency of common data points is reduced compared to uncommon data points. Hence, statistical analysis will be skewed *towards* outliers. It is recommended that the user typically use "Snap" mode to prevent this situation, especially if the data sampling rate is known *a priori*. However, take care about data gaps when using "Snap" mode, as mentioned above.

Please refer to eDNA documentation for more information.

GetHist

The main data pulling functionality is contained in the dna.GetHist function. GetHist will return a pandas DataFrame with the requested data, providing easy access to more advanced data analysis tools in Python.

The start date and end date of the data pull must be specified as input parameters. **Warning**- eDNA prefers the date in this format:

mm/dd/yy hh:mm:ss

While other formats may work, please specify your dates in this format, for safety.

By default, the column label of the DataFrame will be the eDNA tag name, but by specifying the parameter **desc_as_label=True**, the eDNA description can be used instead. Otherwise, a custom label can be specified by **label="CUSTOM_LABEL"**.

Each of the six data pulling methods mentioned above are supported in this function by supplying the parameter **mode="X"**. The default data pulling mode is **raw**:

- avg
- interp
- min
- max
- raw
- snap

By default, the data returned in the pandas DataFrame will use a numpy DateTime as the index. However, if the native eDNA UTC time is requested using **utc=True**, the index will be an integer instead. The speed of the data pull will actually be slightly improved if **utc=True** is selected.

High-speed data can be obtained using the parameter **high_speed=True**. Take care that high speed data is required, because it can significantly slow down the data pull.

Legacy data pulling functions are still available, but have been consolidated into GetHist:

- dna.GetHistAvg
- dna.GetHistInterp
- dna.GetHistMax
- dna.GetHistMin

- dna.GetHistRaw
- dna.GetHistSnap

GetMultipleTags

dna.GetMultipleTags is a convenience function designed to prepare data from multiple tags simultaneously. It may save the user a large amount of time, but it's important to understand what's happening behind the scenes to determine if this function will meet your needs.

The core behavior of GetMultipleTags is to:

1. Pull data from multiple eDNA tags (supplied via a list) using **GetHist(mode="raw")**
2. Remove any duplicated indices (this happens sometimes in eDNA and will cause the concatenation to fail)
3. Concatenate all the DataFrames using an **outer join** (time indices which are not shared will be filled with None)
4. Fill the None values using a "fill-forward" algorithm

If data is to be used for statistical analysis, it is strongly recommended that the user adjust the parameter **sampling_rate="X"**, since data is being pulled using "Raw" mode in this function. The format of the **sampling_rate** parameter uses pandas resampling notation. For instance, "1S" means 1 second, and "5M" means 5 minutes. Refers to pandas documentation for more information.

The parameter **fill_limit** can be used to specify how many data points are filled-forward in step 4 above. If fill_limit is set to 0, the data will not be filled-forward at all.

verify_time=True can be used to ensure that no duplicate time indices exist after the concatenation, which will sometimes occur when more than 10 tags are being concatenated. Unfortunately, this will significantly slow down the data pull.

As with GetHist above, the parameters **desc_as_label** and **utc** may also be specified.

CHAPTER 5

5. Pushing Data

PyeDNA contains the ability to push data *to* an eDNA database. The functions in this section are primarily contained in the "serv" namepace.

Warning- This namespace is still under development, but many of the main functions should be working correctly.

All code in this section assumes that PyeDNA has been imported using:

import pyedna.serv as serv

Serv Capabilities

PyeDNA exposes the following eZDNAServApi functions:

- AddAnalogShortIdRecord
- AddAnalogShortIdRecordNoStatus
- AddDigitalShortIdRecord
- AddAnalogShortIdMsecRecord
- AddAnalogShortIdMsecRecordNoStatus
- AddDigitalShortIdMsecRecord
- FlushShortIdRecords

More information about these functions can be found in eDNA documentation.

CHAPTER 6

Change Log

Version 0.14

- FEATURE- GetServices allows you to get all connected eDNA service information
- FEATURE- GetPoints allows you to get information about all points in a service
- FEATURE- Number of connected services are printed when library is imported
- Better error handling for eDNA connection drops
- Before a data pull, there is now error checking to see if a point exists
- GetMultipleTags no longer automatically resamples and forward-fills data. The user should be in control of this.

Version 0.15

- FEATURE- In the pulling functions, you can now use the desc_as_label parameter to use the point description as the DataFrame column name.
- FEATURE- In the pulling functions, you can now specify a custom column label.
- Better handling of non-ASCII characters in descriptions and units
- GetRTFull never returned a point description- alternative written
- Improved handling of unicode errors- non-Unicode characters are now ignored
- Consistency between ezdna and serv file formatting and dll calls
- Beginnings of a unit test framework
- Miscellaneous code cleanup

Version 0.16

- MAJOR- Refactoring of all GetHistX methods into GetHist. Please use the "mode" parameter to specify the type of history call. Old methods still available.
- New DEPENDENCY- Numba
- Significant speed increase due to JIT compilation
- Bugfix in __init__ header
- Project documentation

Version 0.17

- Bugfix in GetHist related to "switch" statement
- Minor documentation fixes
- Mocking the dna_dll variable so that RTD documentation can be automatically created

Version 0.17

- Bugfix- minor issue with a duplicated last point in every data pull
- Bugfix where GetPoints and GetServices did not return the first entry (due to eDNA apparently treating these functions in a completely different way than the GetHist functions)
- Fixed issue in GetMultipleTags where the pandas function drop_duplicates() removed too many rows. Removed duplicate indices only, instead.

Version 0.18

- Updated logo
- New and improved documentation
- Import function updated
- Minor bugfixes

Version 1.01

- Re-released using the 1.X scheme to fix versioning control (developer mistake)
- No other major updates

CHAPTER 7

Indices and tables

- genindex
- modindex
- search