DriveLink Documentation

Release 0.2.3.1

Chris Dusold

Oct 09, 2017

Contents

1 Indices and tables **3**

2 About **5**

3 Introduction **7**
- 3.1 Interface Base Class 7
- 3.2 Disk Based Dictionary 8
- 3.3 Ordered Access Disk Based Dictionary 9
- 3.4 Disk Based List 9

Python Module Index **11**

Contents:

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

CHAPTER 2

About

A collection of memory conserving data structures designed to give peak performance for on demand data usage while maintaining a constant use of RAM.

Available from PyPI, and easily installed through *pip install DriveLink*. Documentation available at Read The Docs and source available on Github.

CHAPTER 3

Introduction

A library containing storage classes that maintain small RAM usage and original structure access order.

The motivation for this module was to provide constant size RAM usage while maintaining normal use of Python Dictionaries and possibly other structures for semi-big data, where it isn't large enough to warrant more big data centric solutions.

More importantly, this library intends to preserve the usability of Python for rapid prototyping, while enabling larger data access.

Interface Base Class

class drivelink.**Link**(*file_basename*, *size_limit=1024*, *max_pages=16*, *file_location='/home/docs/.DriveLink'*, *compression_ratio=0*)

This abstract base class provides shared functionality for any hard disk linked class required. The other classes in this library use this class, and can be referred to if you need to implement your own version. (Please consider a pull request at https://github.com/cdusold/DriveLink if you make a good general implementation.)

> **Attention:** All classes in DriveLink use this class, so the following applies to each of them.

To be able to implement your own, in addition to implementing the abstract functions, you have to implement self.pages as a dictionary that will work for your class.

This base class provides wrapping that automatically saves to disk, if everything is implemented correctly in inheriting classes. It provides the ability to access implementing classes directly (direct use of Class.close() reccommended) or through a context manager.

Note: This abstract class is not thread safe, nor is it process safe. Any multithreaded or multiprocessed uses of implemented classes hold no guarantees of accuracy.

You can configure how this class stores things in a few ways.

The file_basename parameter allows you to keep multiple different stored objects in the same file_location, which defaults to .DriveLink in the user's home folder. Using a file_basename of the empty string may cause a small slowdown if more than just this object's files are in the folder. Using substrings of other basenames or basenames that end in numbers may cause irregular behavior. Using a file_location of the empty string will result in files being placed in the environment's current location (i.e. what *os.getcwd()* would return).

The size_limit parameter determines how many items are kept in each page, and the max_pages parameter determines how many pages can be kept in memory at the same time. If you use smaller items in the class, increasing either is probably a good idea to get better performance. This setting will only use about 128 MB if standard floats or int32 values. Likely less than 200 MB will ever be in memory, which prevents the RAM from filling up and needing to use swap space. Tuning these values will be project, hardware and usage specific to get the best results. Even with the somewhat low defaults, this will beat out relying on python to use swap space.

In order to speed up disk access, you can specify a compression_ratio. compression is performed using Python's built in ZLib library.

Disk Based Dictionary

class drivelink.**Dict**(*file_basename*, *size_limit=1024*, *max_pages=16*, *file_location='/home/docs/.DriveLink'*, *compression_ratio=0*)

A dictionary class that maintains O(1) look up and write while keeping RAM usage O(1) as well.

This is accomplished through a rudimentary (for now) hashing scheme to page the dictionary into parts.

The object created can be used any way a normal dict would be used, and will clean itself up on python closing. This means saving all the remaining pages to disk. If the file_basename and file_location was used before, it will load the old values back into itself so that the results can be reused.

There are two ways to initialize this object, as a standard object:

```
>>> diskDict = Dict("sampledict")
>>> for i in range(10):
...     diskDict[i] = chr(97+i)
...
>>> diskDict[3]
'd'
>>> 5 in diskDict
True
>>> del diskDict[5]
>>> ", ".join(str(x) for x in diskDict.keys())
'0, 1, 2, 3, 4, 6, 7, 8, 9'
>>> 5 in diskDict
False
```

Or through context:

```
>>> with Dict("testdict") as d:
...     for i in range(10):
...         d[i] = chr(97+i)
...     print(d[3])
d
```

If there is a way to break dict like behavior and you can reproduce it, please report it to the GitHub issues.

Ordered Access Disk Based Dictionary

class drivelink.**OrderedDict**(*file_basename*, *size_limit=1024*, *max_pages=16*, *file_location='/home/docs/.DriveLink'*, *compression_ratio=0*)

A dictionary class that maintains O(1) look up and write while keeping RAM usage O(1) as well.

This is accomplished through a rudimentary (for now) hashing scheme to page the dictionary into parts.

Disk Based List

class drivelink.**List**(*file_basename*, *size_limit=1024*, *max_pages=16*, *file_location='/home/docs/.DriveLink'*, *compression_ratio=0*)

A list class that maintains O(k) look up and O(1) append while keeping RAM usage O(1) as well. Unfortunately, insert is O(n/k).

This is accomplished through paging every size_limit consecutive values together behind the scenes.

The object created can be used any way a normal list would be used, and will clean itself up on python closing. This means saving all the remaining pages to disk. If the file_basename and file_location was used before, it will load the old values back into itself so that the results can be reused.

There are two ways to initialize this object, as a standard object:

```
>>> diskList = List("samplelist")
>>> for i in range(10):
...     diskList.append(i)
...
>>> diskList[3]
3
>>> ", ".join(str(x) for x in diskList)
'0, 1, 2, 3, 4, 5, 6, 7, 8, 9'
>>> del diskList[5]
>>> ", ".join(str(x) for x in diskList)
'0, 1, 2, 3, 4, 6, 7, 8, 9'
```

Or through context:

```
>>> with List("testlist") as d:
...     for i in range(10):
...         d.append(i)
...     print(d[3])
3
```

If there is a way to break list like behavior and you can reproduce it, please report it to the GitHub issues.

Python Module Index

d

drivelink, 7

Index

D

Dict (class in drivelink), 8
drivelink (module), 7

L

Link (class in drivelink), 7
List (class in drivelink), 9

O

OrderedDict (class in drivelink), 9