buzzard Documentation

Release 0.6.5

Nicolas Goguey, Hervé Nivon

Jun 22, 2022

CONTENTS

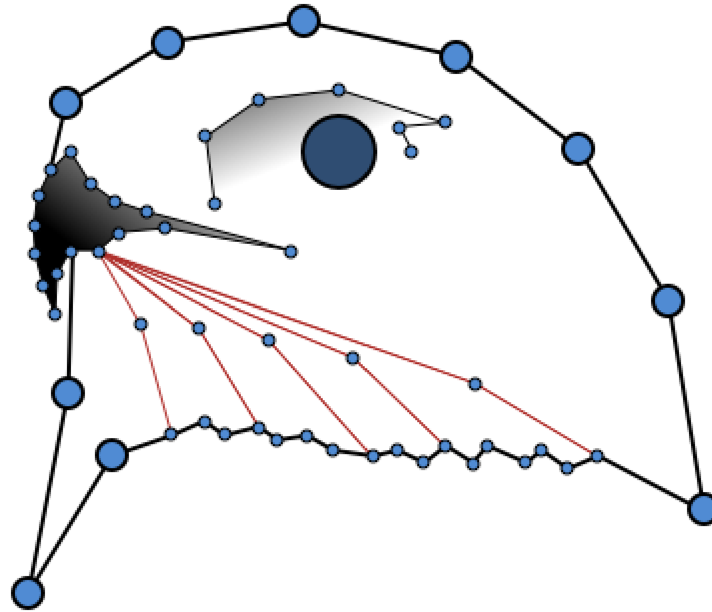
1 API **3**
- 1.1 Dataset . . . 3
- 1.2 Sources . . . 3
- 1.3 Footprint . . . 4
- 1.4 Env . . . 4
- 1.5 Misc. . . . 4

2 Caveats, FAQs and design choices **5**
- 2.1 Caveat List . . . 5
- 2.2 FAQs and design choices . . . 6

3 Indices and tables: **9**

In a nutshell, `buzzard` reads and writes geospatial raster and vector data.

Repository is located here: https://github.com/earthcube-lab/buzzard



CHAPTER

ONE

API

1.1 Dataset

1.1.1 Dataset

1.1.2 Pool Container

1.1.3 Source Constructors

Rasters Sources Using GDAL

Rasters Sources Using NumPy

Rasters Sources Using Recipes

Vectors Sources Using GDAL (OGR)

1.2 Sources

All sources in buzzard can only be constructed from the Dataset methods, see *Source Constructors*

All sources in buzzard inherit from a series of abstract classes:

All the boxes are public classes, called "Facade". They also have a "Back" class associated that is private and can be accessed using the "_back" attribute from the "Facade".

Abstract Classes

Abstract Raster Classes

Abstract Vector Classes

Concrete Classes

APproxyVector
Base class of all vectors, defines a geometry type, fields, and the ability to read geometries

AProxy
Base class of all proxies, defines spatial reference owning

AProxyRaster
Base class of all rasters, defines a location in space, a raster size and a data type, and the ability to read arrays.

AAsyncRaster
Base class for all rasters that are managed by DataSource's scheduler

ARasterRecipe
Base class for all rasters that are computed on the fly

NumpyRaster
Raster wrapping a numpy array

AsyncNumpyRaster
Raster wrapping a numpy array, managed by DataSource's scheduler (NYI in buzzard 0.5.0b1)

AGDALVector
Base class of GDAL vectors

AStoredVector
Base class of all vectors that can be written, defines the ability to insert geometries.

AStored
Base class of all proxies that can be written, defines an opening mode (r/w)

AStoredRaster
Base class of all rasters that can be written, defines the ability to modify arrays

NocacheRasterRecipe
Raster computed on the fly (NYI in buzzard 0.5.0b1)

CachedRasterRecipe
Raster computed on the fly but that caches results on disk

AGDALRaster
Base class of GDAL rasters

GDALMemoryVector
GDAL vector using th 'Memory' driver

AEmissaryVector
Base class of vectors that use a driver

AEmissary
Base class of all proxies that are accessed using a driver (e.g. GDAL 'GeoTiff')

AEmissaryRaster
Base class of rasters that use a driver

GDALMemRaster
GDAL raster using the 'MEM' Driver

GDALFileVector
GDAL vector using a file driver

APooledEmissaryVector
Base class of openable vectors

APooledEmissary
Base class of proxies using a driver that can be opened multiple times (e.g. file descriptor)

APooledEmissaryRaster
Base class of openable rasters

GDALFileRaster
GDAL raster using a file driver

AsyncGDALFileRaster
GDAL raster using a file driver, managed by DataSource's scheduler (NYI in buzzard 0.5.0b1)

1.2.1 GDALFileRaster

1.2.2 GDALMemRaster

1.2.3 NumpyRaster

1.2.4 CachedRasterRecipe

1.2.5 GDALFileVector

1.2.6 GDALMemoryVector

1.3 Footprint

1.4 Env

1.5 Misc.

CHAPTER

TWO

CAVEATS, FAQS AND DESIGN CHOICES

Buzzard has a lot of ambition but is still a young library with several caveats. Are you currently trying to determine if buzzard is the right choice for your project? We got you covered and listed here the use-cases that are currently poorly supported. The rest is a bliss!

2.1 Caveat List

2.1.1 Installation

→ `buzzard` installation is complex because of the `GDAL` and `rtree` dependencies.

→ The anaconda package does not exist

2.1.2 Rasters

→ Reading a raster file is currently internally performed by calls to GDAL drivers, and it might be too slow under certain circumstances. Tweaking the `GDAL_CACHEMAX` variable may improve performances.

→ On-the-fly reprojections is an ambitious feature of buzzard, but this feature only reaches its full potential with vectorial data. On-the-fly raster reprojections are currently partially supported. Those only work if the reprojection preserve angles, if not an exception is raised.

2.1.3 Floating point precision losses

→ The biggest plague of a GIS library is the floating point precision losses. On one hand those losses cannot be avoided (such as in a reprojection operation), and on the other hand certain operations can only be performed with noise-free numbers (such as the floor or ceil operations). The only solution is to round those numbers before critical operations. buzzard has its own way of dealing with this problem: it introduces a global variable to define the number of significant digits that should be considered as noise-less (9 by default).

This way buzzard tries to catch the errors early and raise exceptions. But despite all those efforts some bugs still occur when the noise reaches the significant digits, resulting in strange exceptions being raise.

However those bugs only occur when manipulating very small pixels along with very large coordinates, which is not usual (the ratio `coordinate/pixel-size` should not exceede `10 ** env.significant`).

2.1.4 The Footprint class

→ The Footprint class is long to instanciate (~0.5ms), several use cases involving masses of Footprints are impractical because of this.

→ The Footprint class is the key feature of buzzard, but its specifications are broader that its unit tests: the non-north-up rasters are not fully unit tested. To instanciate such a Footprint the `buzz.env.allow_complex_footprint` should be set to `True`. However those Footprints should work fine in general

→ The Footprint class lack some higher lever constructors to make several common construction schemes easier. However by using the intersection method of a Footprint on itself and tweaking the 3 optional parameters covers most of the missing use-cases.

2.1.5 The async rasters

→ Most of the async rasters as advertised in the doc or the examples are not yet implemented. Only the cached raster recipes are.

→ Using cached raster recipes has a side effect on a file system. Using a single cache directory from two different programs at the same time is an undefined behavior. Although it works fine when the cache files are already instantiated.

→ The scheduler that was written to support the async rasters is not proven to be bug free. Although it is filled with assertions that will most likely catch any remaining bug.

2.2 FAQs and design choices

The following list contains the FAQs or features that are often mistaken as bugs ;)

→ Why buzzard instead of fiona or rasterio that are much more mature and straightforward libraries?

The answer is simple: as soon as you are working with large images, or with geometries alongside images, you can benefit from the higher level abstractions that buzzard provides.

→ Why can't I simply reproject shapely geometries using buzzard?

Because buzzard does not aim to replace pyproj. When using the classic stack, each of osgeo's lib has its own wrapper:

- GEOS -> shapely
- OGR -> fiona
- GDAL -> rio
- OSR -> pyproj

Buzzard is transversal, it wraps enough OGR, GDAL and OSR so that you don't have import those most of the time. Some known exceptions are:

- Raster reprojection that does not preserve angles
- Shapely objects reprojection
- Contour lines generation

It might be the case that someday buzzard provides a transversal feature that replaces pyproj but nothing is planned.

→ In buzzard, all sources (such as raster and vector files) are tied to a Dataset object. This is design choice has several advantages now and even more advantages in the long term. See the Dataset's docstring.

→ buzzard is a binding for GDAL, but all the features that allows editing the attributes of an opened file are not exposed in buzzard. The wish here is to make buzzard as functional as possible.

→ The `with Dataset.close as ds:` syntax is chosen over the `with Dataset as ds:` syntax in order to stay consistent with the `with Source.close as src:` syntax, that itself exist because of the need for disambiguation with this other feature: `with Source.delete as src:`.

→ The Footprint class is an immutable object. This is not a bug.

→ Why is the Footprint class not directly implementing a shapely Polygon?

In the early versions of buzzard, it was the case. But method name conflicts became a big problem. And overall, it was not that useful. You can still use Footprint.poly to convert a Footprint to a shapely Polygon.

→ Why support non-north-up Footprints?

It was harder to design but cleaner in the end. Now that it is (mostly - missing unit tests at the moment) supported there is a hope that it creates new use cases.

→ Why are the get_data and set_data methods of a raster so complex?

Those methods accept **any** Footprint as a parameter, it includes Footprints that don't share alignment/scale/rotation/bounds with the raster source. It allows the user to forget about the file when designing a piece of code. The downside of this feature is that the user is not aware when a resource consuming resampling is performed. To avoid this problem, the Dataset class is by default configured to raise an error when an interpolation occurs.

→ If you ever wander in the buzzard source code you may notice that the Dataset class holds pointers to Source objects and vice versa (through dependency injection). This recursive dependency reveal the design choice of making the Dataset and the Source classes a **single** class. The Source objects should be seen as extensions of a Dataset object.

→ If you ever wander in the buzzard source code you will notice a complex separation of concern scheme in which a class is split between a `facade` and a `back` class.

This separation exists in order to allow garbage collection to be made, even if the Dataset instantiates a scheduler on a separate thread. The facade classes are manipulated by the user and have pointers towards the back classes, and the later have no references to the facade, while the scheduler only have pointers to the `back` classes. This way, when the facade are collected, the back are collected too. This separation also allows us to perform parameter checking only once in the facade classes, and then call the appropriate back implementation using dynamic dispatch.

CHAPTER

THREE

INDICES AND TABLES:

- genindex
- modindex
- search