Fiona Documentation

Sean Gillies

May 23, 2023

CONTENTS

Fiona streams simple feature data to and from GIS formats like GeoPackage and Shapefile. Simple features are record, or row-like, and have a single geometry attribute. Fiona can read and write real-world simple feature data using multi-layered GIS formats, zipped and in-memory virtual file systems, from files on your hard drive or in cloud storage. This project includes Python modules and a command line interface (CLI).

Here's an example of streaming and filtering features from a zipped dataset on the web and saving them to a new layer in a new Geopackage file.

```
import fiona

with fiona.open(
    "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip"
) as src:
    profile = src.profile
    profile["driver"] = "GPKG"

    with fiona.open("example.gpkg", "w", layer="selection", **profile) as dst:
        dst.writerecords(feat in src.filter(bbox=(-107.0, 37.0, -105.0, 39.0)))
```

The same result can be achieved on the command line using a combination of fio-cat and fio-load.

```
fio cat zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip --bbox "-
↪107.0,37.0,-105.0,39.0" \
| fio load -f GPKG --layer selection example.gpkg
```

CHAPTER

ONE

FIONA

Fiona streams simple feature data to and from GIS formats like GeoPackage and Shapefile.

Fiona can read and write real-world data using multi-layered GIS formats, zipped and in-memory virtual file systems, from files on your hard drive or in cloud storage. This project includes Python modules and a command line interface (CLI).

Fiona depends on GDAL but is different from GDAL's own bindings. Fiona is designed to be highly productive and to make it easy to write code which is easy to read.

1.1 Installation

Fiona has several extension modules which link against libgdal. This complicates installation. Binary distributions (wheels) containing libgdal and its own dependencies are available from the Python Package Index and can be installed using pip.

```
pip install fiona
```

These wheels are mainly intended to make installation easy for simple applications, not so much for production. They are not tested for compatibility with all other binary wheels, conda packages, or QGIS, and omit many of GDAL's optional format drivers. If you need, for example, GML support you will need to build and install Fiona from a source distribution. It is possible to install Fiona from source using pip (version >= 22.3) and the *–no-binary* option. A specific GDAL installation can be selected by setting the GDAL_CONFIG environment variable.

```
pip install -U pip
pip install --no-binary fiona fiona
```

Many users find Anaconda and conda-forge a good way to install Fiona and get access to more optional format drivers (like GML).

Fiona 1.9 requires Python 3.7 or higher and GDAL 3.2 or higher.

1.2 Python Usage

Features are read from and written to file-like `Collection` objects returned from the `fiona.open()` function. Features are data classes modeled on the GeoJSON format. They don't have any spatial methods of their own, so if you want to transform them you will need Shapely or something like it. Here is an example of using Fiona to read some features from one data file, change their geometry attributes using Shapely, and write them to a new data file.

```
import fiona
from fiona import Feature, Geometry
from shapely.geometry import mapping, shape

# Open a file for reading. We'll call this the source.
with fiona.open(
    "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip"
) as src:

    # The file we'll write to must be initialized with a coordinate
    # system, a format driver name, and a record schema. We can get
    # initial values from the open source's profile property and then
    # modify them as we need.
    profile = src.profile
    profile["schema"]["geometry"] = "Point"
    profile["driver"] = "GPKG"

    # Open an output file, using the same format driver and coordinate
    # reference system as the source. The profile mapping fills in the
    # keyword parameters of fiona.open.
    with fiona.open("centroids.gpkg", "w", **profile) as dst:

        # Process only the feature records intersecting a box.
        for feat in src.filter(bbox=(-107.0, 37.0, -105.0, 39.0)):

            # Get the feature's centroid.
            centroid_shp = shape(feat.geometry).centroid
            new_geom = Geometry.from_dict(centroid_shp)

            # Write the feature out.
            dst.write(
                Feature(geometry=new_geom, properties=f.properties)
            )

    # The destination's contents are flushed to disk and the file is
    # closed when its with block ends. This effectively
    # executes ``dst.flush(); dst.close()``.
```

1.3 CLI Usage

Fiona's command line interface, named "fio", is documented at docs/cli.rst. The CLI has a number of different commands. Its `fio cat` command streams GeoJSON features from any dataset.

```
$ fio cat --compact tests/data/coutwildrnp.shp | jq -c '.'
{"geometry":{"coordinates":[[[-111.73527526855469,41.995094299316406],...]]}}
...
```

1.4 Documentation

For more details about this project, please see:

- Fiona home page
- Docs and manual
- Examples
- Main user discussion group
- Developers discussion group

1.5 Changes

All issue numbers are relative to https://github.com/Toblerity/Fiona/issues.

1.5.1 1.9.4.post1 (2023-05-23)

Extraneous files were unintentionally packaged in the 1.9.4 wheels. This post1 release excludes them so that wheel contents are as in version 1.9.3.

1.5.2 1.9.4 (2023-05-16)

- The performance of Feature.from_dict() has been improved (#1267).
- Several sources of meaningless log messages from fiona._geometry about NULL geometries are avoided (#1264).
- The Parquet driver has been added to the list of supported drivers and will be available if your system's GDAL library links libarrow. Note that fiona wheels on PyPI do not include libarrow as it is rather large.
- Ensure that fiona._vendor modules are found and included.
- Bytes type feature properties are now hex encoded when serializing to GeoJSON (#1263).
- Docstrings for listdir and listlayers have been clarified and harmonized.
- Nose style test cases have been converted to unittest.TestCase (#1256).
- The munch package used by fio-filter and fio-calc is now vendored and patched to remove usage of the deprecated pkg_resources module (#1255).

1.5.3 1.9.3 (2023-04-10)

- Rasterio CRS objects are compatible with the Collection constructor and are now accepted (#1248).
- Enable append mode for fio-load (#1237).
- Reading a GeoJSON with an empty array property can result in a segmentation fault since version 1.9.0. This has been fixed (#1228).

1.5.4 1.9.2 (2023-03-20)

- Get command entry points using importlib.metadata (#1220).
- Instead of warning, transform_geom() raises an exception when some points can't be reprojected unless the caller opts in to partial reprojection. This restores the behavior of version 1.8.22.
- Add support for open options to all CLI commands that call fiona.open (#1215).
- Fix a memory leak that can occur when iterating over a dataset using strides (#1205).
- ZipMemoryFile now supports zipped GDB data (#1203).

1.5.5 1.9.1 (2023-02-09)

- Log a warning message when identically named fields are encountered (#1201).
- Avoid dependence on listdir order in tests (#1193).
- Prevent empty geometries arrays from appearing in __geo_interface__ (#1197).
- setuptools added to pyproject.toml. Its pkg_resources module is used by the CLI (#1191).

1.5.6 1.9.0 (2023-01-30)

- CITATION.txt has been replaced by a new CITATION.cff file and the credits have been updated.
- In setup.py the distutils (deprecated) logger is no longer used.

1.5.7 1.9b2 (2023-01-22)

- Add Feature.__geo_interface__ property (#1181).
- Invalid creation options are filtered and ignored (#1180).
- The readme doc has been shortened and freshened up, with a modern example for version 1.9.0 (#1174).
- The Geometry class now provides and looks for __geo_interface__ (#1174).
- The top level fiona module now exports Feature, Geometry, and Properties (#1174).
- Functions that take Feature or Geometry objects will continue to take dicts or objects that provide __geo_interface__ (#1177). This reverses the deprecation introduced in 1.9a2.
- Python ignores SIGPIPE by default. By never catching BrokenPipeError via *except Exception* when, for example, piping the output of rio-shapes to the Unix head program, we avoid getting an unhandled BrokenPipeError message when the interpreter shuts down (#2689).

1.5.8 1.9b1 (2022-12-13)

New features:

- Add listdir and listlayers method to io.MemoryFile (resolving #754).
- Add support for TIN and triangle geometries (#1163).
- Add an allow_unsupported_drivers option to fiona.open() (#1126).
- Added support for the OGR StringList field type (#1141).

Changes and bug fixes:

- Missing and unused imports have been added or removed.
- Make sure that errors aren't lost when a collection can't be saved properly (#1169).
- Ensure that ZipMemoryFile have the proper GDAL name after creation so that we can use listdir() (#1092).
- The fiona._loading module, which supports DLL loading on Windows, has been moved into __init__.py and is no longer used anywhere else (#1168).
- Move project metadata to pyproject.toml (#1165).
- Update drvsupport.py to reflect new format capabilities in GDAL 3.6.0 (#1122).
- Remove debug logging from env and _env modules.

1.5.9 1.9a3 (2022-10-17)

Packaging:

- Builds now require Cython >= 0.29.29 because of
- https://github.com/cython/cython/issues/4609 (see #1143).
- PyPI wheels now include GDAL 3.5.2, PROJ 9.0.1, and GEOS 3.11.0.
- PyPI wheels are now available for Python 3.11.

1.5.10 1.9a2 (2022-06-10)

Deprecations:

- Fiona's API methods will accept feature and geometry dicts in 1.9.0, but this usage is deprecated. Instances of Feature and Geometry will be required in 2.0.
- The precision keyword argument of fiona.transform.transform_geom is deprecated and will be removed in version 2.0.
- Deprecated usage has been eliminated in the project. Fiona's tests pass when run with a -Werror::DeprecationWarning filter.

Changes:

- Fiona's FionaDeprecationWarning now sub-classes DeprecationWarning.
- Some test modules have beeen re-formatted using black.

New features:

- Fiona Collections now carry a context exit stack into which we can push fiona Envs and MemoryFiles (#1059).
- Fiona has a new CRS class, like rasterio's, which is compatible with the CRS dicts of previous versions (#714).

1.5.11 1.9a1 (2022-05-19)

Deprecations:

- The fiona.drivers() function has been deprecated and will be removed in version 2.0. It should be replaced by fiona.Env().
- The new fiona.meta module will be renamed to fiona.drivers in version 2.0.

Packaging:

- Source distributions contain no C source files and require Cython to create them from .pyx files (#1096).

Changes:

- Shims for various versions of GDAL have been removed and are replaced by Cython compilation conditions (#1093).
- Use of CURL_CA_BUNDLE environment variable is replaced by a more specific GDAL/PROJ_CURL_CA_BUNDLE (#1095).
- Fiona's feature accessors now return instances of fiona.model.Feature instead of Python dicts (#787). The Feature class is compatible with code that expects GeoJSON-like dicts but also provides id, geometry, and properties attributes. The last two of these are instances of fiona.model.Geometry and fiona.model.Properties.
- GDAL 3.1.0 is the minimum GDAL version.
- Drop Python 2, and establish Python 3.7 as the minimum version (#1079).
- Remove six and reduce footprint of fiona.compat (#985).

New features:

- The appropriate format driver can be detected from filename in write mode (#948).
- Driver metadata including dataset open and dataset and layer creations options are now exposed through methods of the fiona.meta module (#950).
- CRS WKT format support (#979).
- Add 'where' SQL clause to set attribute filter (#961, #1097).

Bug fixes:

- Env and Session classes have been updated for parity with rasterio and to resolve a credential refresh bug (#1055).

1.5.12 1.8.21 (2022-02-07)

Changes:

- Driver mode support tests have been made more general and less susceptible to driver quirks involving feature fields and coordinate values (#1060).
- OSError is raised on attempts to open a dataset in a Python file object in "a" mode (see #1027).
- Upgrade attrs, cython, etc to open up Python 3.10 support (#1049).

Bug fixes:

- Allow FieldSkipLogFilter to handle exception messages as well as strings (reported in #1035).
- Clean up VSI files left by MemoryFileBase, resolving #1041.
- Hard-coded "utf-8" collection encoding added in #423 has been removed (#1057).

1.5.13 1.8.20 (2021-05-31)

Packaging:

- Wheels include GDAL 3.3.0 and GEOS 3.9.1.

Bug fixes:

- Allow use with click 8 and higher (#1015).

1.5.14 1.8.19 (2021-04-07)

Packaging:

- Wheels include GDAL 3.2.1 and PROJ 7.2.1.

Bug fixes:

- In fiona/env.py the GDAL data path is now configured using set_gdal_config instead by setting the GDAL_DATA environment variable (#1007).
- Spurious iterator reset warnings have been eliminatged (#987).

1.5.15 1.8.18 (2020-11-17)

- The precision option of transform has been fixed for the case of GeometryCollections (#971, #972).
- Added missing –co (creation) option to fio-load (#390).
- If the certifi package can be imported, its certificate store location will be passed to GDAL during import of fiona._env unless CURL_CA_BUNDLE is already set.
- Warn when feature fields named “” are found (#955).

1.5.16 1.8.17 (2020-09-09)

- To fix issue #952 the fio-cat command no longer cuts feature geometries at the anti-meridian by default. A –cut-at-antimeridian option has been added to allow cutting of geometries in a geographic destination coordinate reference system.

1.5.17 1.8.16 (2020-09-04)

- More OGR errors and warnings arising in calls to GDAL C API functions are surfaced (#946).
- A circular import introduced in some cases in 1.8.15 has been fixed (#945).

1.5.18 1.8.15 (2020-09-03)

- Change shim functions to not return tuples (#942) as a solution for the packaging problem reported in #941.
- Raise a Python exception when VSIFOpenL fails (#937).

1.5.19 1.8.14 (2020-08-31)

- When creating a new Collection in a MemoryFile with a default (random) name Fiona will attempt to use a format driver-supported file extension (#934). When initializing a MemoryFile with bytes of data formatted for a vector driver that requires a certain file name or extension, the user should continue to pass an appropriate filename and/or extension.
- Read support for FlatGeobuf has been enabled in the drvsupport module.
- The MemoryFile implementation has been improved so that it can support multi-part S3 downloads (#906). This is largely a port of code from rasterio.
- Axis ordering for results of fiona.transform was wrong when CRS were passed in the "EPSG:dddd" form (#919). This has been fixed by (#926).
- Allow implicit access to the only dataset in a ZipMemoryFile. The path argument of ZipMemoryFile.open() is now optional (#928).
- Improve support for datetime types: support milliseconds (#744), timezones (#914) and improve warnings if type is not supported by driver (#572).
- Fix "Failed to commit transaction" TransactionError for FileGDB driver.
- Load GDAL DLL dependencies on Python 3.8+ / Windows with add_dll_directory() (#851).
- Do not require optional properties (#848).
- Ensure that slice does not overflow available data (#884).
- Resolve issue when "ERROR 4: Unable to open EPSG support file gcs.csv." is raised on importing fiona (#897).
- Resolve issue resulting in possible mixed up fields names (affecting only DXF, GPX, GPSTrackMacker and DGN driver) (#916).
- Ensure crs_wkt is passed when writing to MemoryFile (#907).

1.5.20 1.8.13.post1 (2020-02-21)

- This release is being made to improve binary wheel compatibility with shapely 1.7.0. There have been no changes to the fiona package code since 1.8.13.

1.5.21 1.8.13 (2019-12-05)

- The Python version specs for argparse and ordereddict in 1.8.12 were wrong and have been corrected (#843).

1.5.22 1.8.12 (2019-12-04)

- Specify Python versions for argparse, enum34, and ordereddict requirements (#842).

1.5.23 1.8.11 (2019-11-07)

- Fix an access violation on Windows (#826).

1.5.24 1.8.10 (2019-11-07)

Deprecations:

- Use of vfs keyword argument with open or listlayers has been previously noted as deprecated, but now triggers a deprecation warning.

Bug fixes:

- fiona.open() can now create new datasets using CRS URNs (#823).
- listlayers() now accepts file and Path objects, like open() (#825).
- Use new set_proj_search_path() function to set the PROJ data search path. For GDAL versions before 3.0 this sets the PROJ_LIB environment variable. For GDAL version 3.0 this calls OSRSetPROJSearchPaths(), which overrides PROJ_LIB.
- Remove old and unused _drivers extension module.
- Check for header.dxf file instead of pcs.csv when looking for installed GDAL data. The latter is gone with GDAL 3.0 but the former remains (#818).

1.5.25 1.8.9.post2 (2019-10-22)

- The 1.8.9.post1 release introduced a bug affecting builds of the package from a source distribution using GDAL 2.x. This bug has been fixed in commit 960568d.

1.5.26 1.8.9.post1 (2019-10-22)

- A change has been made to the package setup script so that the shim module for GDAL 3 is used when building the package from a source distribution. There are no other changes to the package.

1.5.27 1.8.9 (2019-10-21)

- A shim module and support for GDAL 3.0 has been added. The package can now be built and used with GDAL 3.0 and PROJ 6.1 or 6.2. Note that the 1.8.9 wheels we will upload to PyPI will contain GDAL 2.4.2 and PROJ 4.9.3 as in the 1.8.8 wheels.

1.5.28 1.8.8 (2019-09-25)

- The schema of geopackage files with a geometry type code of 3000 could not be reported using Fiona 1.8.7. This bug is fixed.

1.5.29 1.8.7 (2019-09-24)

Bug fixes:

- Regression in handling of polygons with M values noted under version 1.8.5 below was in fact not fixed then (see new report #789), but is fixed in version 1.8.7.
- Windows filenames containing "!" are now parsed correctly, fixing issue #742.

Upcoming changes:

- In version 1.9.0, the objects yielded when a Collection is iterated will be mutable mappings but will no longer be instances of Python's dict. Version 1.9 is intended to be backwards compatible with 1.8 except where user code tests *isinstance(feature, dict)*. In version 2.0 the new Feature, Geometry, and Properties classes will become immutable mappings. See https://github.com/Toblerity/fiona-rfc/blob/master/rfc/0001-fiona-2-0-changes.md for more discussion of the upcoming changes for version 2.0.

1.5.30 1.8.6 (2019-03-18)

- The advertisement for JSON driver enablement in 1.8.5 was false (#176), but in this release they are ready for use.

1.5.31 1.8.5 (2019-03-15)

- GDAL seems to work best if GDAL_DATA is set as early as possible. Ideally it is set when building the library or in the environment before importing Fiona, but for wheels we patch GDAL_DATA into os.environ when fiona.env is imported. This resolves #731.
- A combination of bugs which allowed .cpg files to be overlooked has been fixed (#726).
- On entering a collection context (Collection.__enter__) a new anonymous GDAL environment is created if needed and entered. This makes *with fiona.open(…) as collection:* roughly equivalent to *with fiona.open(…) as collection, Env():*. This helps prevent bugs when Collections are created and then used later or in different scopes.
- Missing GDAL support for TopoJSON, GeoJSONSeq, and ESRIJSON has been enabled (#721).
- A regression in handling of polygons with M values (#724) has been fixed.
- Per-feature debug logging calls in OGRFeatureBuilder methods have been eliminated to improve feature writing performance (#718).
- Native support for datasets in Google Cloud Storage identified by "gs" resource names has been added (#709).
- Support has been added for triangle, polyhedral surface, and TIN geometry types (#679).
- Notes about using the MemoryFile and ZipMemoryFile classes has been added to the manual (#674).

1.5.32 1.8.4 (2018-12-10)

- 3D geometries can now be transformed with a specified precision (#523).
- A bug producing a spurious DriverSupportError for Shapefiles with a "time" field (#692) has been fixed.
- Patching of the GDAL_DATA environment variable was accidentally left in place in 1.8.3 and now has been removed.

1.5.33 1.8.3 (2018-11-30)

- The RASTERIO_ENV config environment marker this project picked up from Rasterio has been renamed to FIONA_ENV (#665).
- Options –gdal-data and –proj-data have been added to the fio-env command so that users of Rasterio wheels can get paths to set GDAL_DATA and PROJ_LIB environment variables.
- The unsuccessful attempt to make GDAL and PROJ support file discovery and configuration automatic within collection's crs and crs_wkt properties has been reverted. Users must execute such code inside a *with Env()* block or set the GDAL_DATA and PROJ_LIB environment variables needed by GDAL.

1.5.34 1.8.2 (2018-11-19)

Bug fixes:

- Raise FionaValueError when an iterator's __next__ is called and the session is found to be missing or inactive instead of passing a null pointer to OGR_L_GetNextFeature (#687).

1.5.35 1.8.1 (2018-11-15)

Bug fixes:

- Add checks around OSRGetAuthorityName and OSRGetAuthorityCode calls that will log problems with looking up these items.
- Opened data sources are now released before we raise exceptions in WritingSession.start (#676). This fixes an issue with locked files on Windows.
- We now ensure that an Env instance exists when getting the crs or crs_wkt properties of a Collection (#673, #690). Otherwise, required GDAL and PROJ data files included in Fiona wheels can not be found.
- GDAL and PROJ data search has been refactored to improve testability (#678).
- In the project's Cython code, void* pointers have been replaced with proper GDAL types (#672).
- Pervasive warning level log messages about ENCODING creation options (#668) have been eliminated.

1.5.36 1.8.0 (2018-10-31)

This is the final 1.8.0 release. Thanks, everyone!

Bug fixes:

- We cpdef Session.stop so that it has a C version that can be called safely from __dealloc__, fixing a PyPy issue (#659, #553).

1.5.37 1.8rc1 (2018-10-26)

There are no changes in 1.8rc1 other than more test standardization and the introduction of a temporary test_collection_legacy.py module to support the build of fully tested Python 2.7 macosx wheels on Travis-CI.

1.5.38 1.8b2 (2018-10-23)

Bug fixes:

- The ensure_env_with_credentials decorator will no longer clobber credentials of the outer environment. This fixes a bug reported to the Rasterio project and which also existed in Fiona.
- An unused import of the packaging module and the dependency have been removed (#653).
- The Env class logged to the 'rasterio' hierarchy instead of 'fiona'. This mistake has been corrected (#646).
- The Mapping abstract base class is imported from collections.abc when possible (#647).

Refactoring:

- Standardization of the tests on pytest functions and fixtures continues and is nearing completion (#648, #649, #650, #651, #652).

1.5.39 1.8b1 (2018-10-15)

Deprecations:

- Collection slicing has been deprecated and will be prohibited in a future version.

Bug fixes:

- Rasterio CRS objects passed to transform module methods will be converted to dicts as needed (#590).
- Implicitly convert curve geometries to their linear approximations rather than failing (#617).
- Migrated unittest test cases in test_collection.py and test_layer.py to the use of the standard data_dir and path_coutwildrnp_shp fixtures (#616).
- Root logger configuration has been removed from all test scripts (#615).
- An AWS session is created for the CLI context Env only if explicitly requested, matching the behavior of Rasterio's CLI (#635).
- Dependency on attrs is made explicit.
- Other dependencies are pinned to known good versions in requirements files.
- Unused arguments have been removed from the Env constructor (#637).

Refactoring:

- A with_context_env decorator has been added and used to set up the GDAL environment for CLI commands. The command functions themselves are now simplified.

1.5.40 1.8a3 (2018-10-01)

Deprecations:

- The `fiona.drivers()` context manager is officially deprecated. All users should switch to `fiona.Env()`, which registers format drivers and manages GDAL configuration in a reversible manner.

Bug fixes:

- The Collection class now filters log messages about skipped fields to a maximum of one warning message per field (#627).
- The boto3 module is only imported when needed (#507, #629).
- Compatibility with Click 7.0 is achieved (#633).
- Use of %r instead of %s in a debug() call prevents UnicodeDecodeErrors (#620).

1.5.41 1.8a2 (2018-07-24)

New features:

- 64-bit integers are the now the default for int type fields (#562, #564).
- 'http', 's3', 'zip+http', and 'zip+s3' URI schemes for datasets are now supported (#425, #426).
- We've added a `MemoryFile` class which supports formatted in-memory feature collections (#501).
- Added support for GDAL 2.x boolean field sub-type (#531).
- A new `fio rm` command makes it possible to cleanly remove multi-file datasets (#538).
- The geometry type in a feature collection is more flexible. We can now specify not only a single geometry type, but a sequence of permissible types, or "Any" to permit any geometry type (#539).
- Support for GDAL 2.2+ null fields has been added (#554).
- The new `gdal_open_vector()` function of our internal API provides much improved error handling (#557).

Bug fixes:

- The bug involving OrderedDict import on Python 2.7 has been fixed (#533).
- An `AttributeError` raised when the `--bbox` option of fio-cat is used with more than one input file has been fixed (#543, #544).
- Obsolete and derelict fiona.tool module has been removed.
- Revert the change in 0a2bc7c that discards Z in geometry types when a collection's schema is reported (#541).
- Require six version 1.7 or higher (#550).
- A regression related to "zip+s3" URIs has been fixed.
- Debian's GDAL data locations are now searched by default (#583).

1.5.42 1.8a1 (2017-11-06)

New features:

- Each call of `writerecords()` involves one or more transactions of up to 20,000 features each. This improves performance when writing GeoPackage files as the previous transaction size was only 200 features (#476, #491).

Packaging:

- Fiona's Cython source files have been refactored so that there are no longer separate extension modules for GDAL 1.x and GDAL 2.x. Instead there is a base extension module based on GDAL 2.x and shim modules for installations that use GDAL 1.x.

1.5.43 1.7.11.post1 (2018-01-08)

- This post-release adds missing expat (and thereby GPX format) support to the included GDAL library (still version 2.2.2).

1.5.44 1.7.11 (2017-12-14)

- The `encoding` keyword argument for `fiona.open()`, which is intended to allow a caller to override a data source's own and possibly erroneous encoding, has not been working (#510, #512). The problem is that we weren't always setting GDAL open or config options before opening the data sources. This bug is resolved by a number of commits in the maint-1.7 branch and the fix is demonstrated in tests/test_encoding.py.
- An `--encoding` option has been added to fio-load to enable creation of encoded shapefiles with an accompanying .cpg file (#499, #517).

1.5.45 1.7.10.post1 (2017-10-30)

- A post-release has been made to fix a problem with macosx wheels uploaded to PyPI.

1.5.46 1.7.10 (2017-10-26)

Bug fixes:

- An extraneous printed line from the `rio cat --layers` validator has been removed (#478).

Packaging:

- Official OS X and Manylinux1 wheels (on PyPI) for this release will be compatible with Shapely 1.6.2 and Rasterio 1.0a10 wheels.

1.5.47 1.7.9.post1 (2017-08-21)

This release introduces no changes in the Fiona package. It upgrades GDAL from 2.2.0 to 2.2.1 in wheels that we publish to the Python Package Index.

1.5.48 1.7.9 (2017-08-17)

Bug fixes:

- Acquire the GIL for GDAL error callback functions to prevent crashes when GDAL errors occur when the GIL has been released by user code.
- Sync and flush layers when closing even when the number of features is not precisely known (#467).

1.5.49 1.7.8 (2017-06-20)

Bug fixes:

- Provide all arguments needed by CPLError based exceptions (#456).

1.5.50 1.7.7 (2017-06-05)

Bug fixes:

- Switch logger *warn()* (deprecated) calls to *warning()*.
- Replace all relative imports and cimports in Cython modules with absolute imports (#450).
- Avoid setting *PROJ_LIB* to a non-existent directory (#439).

1.5.51 1.7.6 (2017-04-26)

Bug fixes:

- Fall back to *share/proj* for PROJ_LIB (#440).
- Replace every call to *OSRDestroySpatialReference()* with *OSRRelease()*, fixing the GPKG driver crasher reported in #441 (#443).
- Add a *DriverIOError* derived from *IOError* to use for driver-specific errors such as the GeoJSON driver's refusal to overwrite existing files. Also we now ensure that when this error is raised by *fiona.open()* any created read or write session is deleted, this eliminates spurious exceptions on teardown of broken *Collection* objects (#437, #444).

1.5.52 1.7.5 (2017-03-20)

Bug fixes:

- Opening a data file in read (the default) mode with *fiona.open()* using the the *driver* or *drivers* keyword arguments (to specify certain format drivers) would sometimes cause a crash on Windows due to improperly terminated lists of strings (#428). The fix: Fiona's buggy *string_list()* has been replaced by GDAL's *CSLAddString()*.

1.5.53 1.7.4 (2017-02-20)

Bug fixes:

- OGR's EsriJSON detection fails when certain keys aren't found in the first 6000 bytes of data passed to *BytesCollection* (#422). A .json file extension is now explicitly given to the in-memory file behind *BytesCollection* when the *driver='GeoJSON'* keyword argument is given (#423).

1.5.54 1.7.3 (2017-02-14)

Roses are red. Tan is a pug. Software regression's the most embarrassing bug.

Bug fixes:

- Use __stdcall for GDAL error handling callback on Windows as in Rasterio.
- Turn on latent support for zip:// URLs in rio-cat and rio-info (#421).
- The 1.7.2 release broke support for zip files with absolute paths (#418). This regression has been fixed with tests to confirm.

1.5.55 1.7.2 (2017-01-27)

Future Deprecation:

- *Collection.__next__()* is buggy in that it can lead to duplication of features when used in combination with *Collection.filter()* or *Collection.__iter__()*. It will be removed in Fiona 2.0. Please check for usage of this deprecated feature by running your tests or programs with *PYTHONWARNINGS="always:::fiona"* or *-W"always:::fiona"* and switch from *next(collection)* to *next(iter(collection))* (#301).

Bug fix:

- Zipped streams of bytes can be accessed by *BytesCollection* (#318).

1.5.56 1.7.1.post1 (2016-12-23)

- New binary wheels using version 1.2.0 of sgillies/frs-wheel-builds. See https://github.com/sgillies/frs-wheel-builds/blob/master/CHANGES.txt.

1.5.57 1.7.1 (2016-11-16)

Bug Fixes:

- Prevent Fiona from stumbling over 'Z', 'M', and 'ZM' geometry types introduced in GDAL 2.1 (#384). Fiona 1.7.1 doesn't add explicit support for these types, they are coerced to geometry types 1-7 ('Point', 'LineString', etc.)
- Raise an *UnsupportedGeometryTypeError* when a bogus or unsupported geometry type is encountered in a new collection's schema or elsewhere (#340).
- Enable *–precision 0* for fio-cat (#370).
- Prevent datetime exceptions from unnecessarily stopping collection iteration by yielding *None* (#385)
- Replace log.warn calls with log.warning calls (#379).

- Print an error message if neither gdal-config or *–gdalversion* indicate a GDAL C API version when running *setup.py* (#364).
- Let dict-like subclasses through CRS type checks (#367).

1.5.58 1.7.0post2 (2016-06-15)

Packaging: define extension modules for 'clean' and 'config' targets (#363).

1.5.59 1.7.0post1 (2016-06-15)

Packaging: No files are copied for the 'clean' setup target (#361, #362).

1.5.60 1.7.0 (2016-06-14)

The C extension modules in this library can now be built and used with either a 1.x or 2.x release of the GDAL library. Big thanks to René Buffat for leading this effort.

Refactoring:

- The *ogrext1.pyx* and *ogrext2.pyx* files now use separate C APIs defined in *ogrext1.pxd* and *ogrex2.pxd*. The other extension modules have been refactored so that they do not depend on either of these modules and use subsets of the GDAL/OGR API compatible with both GDAL 1.x and 2.x (#359).

Packaging:

- Source distributions now contain two different sources for the *ogrext* extension module. The *ogrext1.c* file will be used with GDAL 1.x and the *ogrext2.c* file will be used with GDAL 2.x.

1.5.61 1.7b2 (2016-06-13)

- New feature: enhancement of the *–layer* option for fio-cat and fio-dump to allow separate layers of one or more multi-layer input files to be selected (#349).

1.5.62 1.7b1 (2016-06-10)

- New feature: support for GDAL version 2+ (#259).
- New feature: a new fio-calc CLI command (#273).
- New feature: *–layer* options for fio-info (#316) and fio-load (#299).
- New feature: a *–no-parse* option for fio-collect that lets a careful user avoid extra JSON serialization and deserialization (#306).
- Bug fix: *+wktext* is now preserved when serializing CRS from WKT to PROJ.4 dicts (#352).
- Bug fix: a small memory leak when opening a collection has been fixed (#337).
- Bug fix: internal unicode errors now result in a log message and a *UnicodeError* exception, not a *TypeError* (#356).

1.5.63 1.6.4 (2016-05-06)

- Raise ImportError if the active GDAL library version is >= 2.0 instead of failing unpredictably (#338, #341). Support for GDAL>=2.0 is coming in Fiona 1.7.

1.5.64 1.6.3.post1 (2016-03-27)

- No changes to the library in this post-release version, but there is a significant change to the distributions on PyPI: to help make Fiona more compatible with Shapely on OS X, the GDAL shared library included in the macosx (only) binary wheels now statically links the GEOS library. See https://github.com/sgillies/frs-wheel-builds/issues/5.

1.5.65 1.6.3 (2015-12-22)

- Daytime has been decreasing in the Northern Hemisphere, but is now increasing again as it should.
- Non-UTF strings were being passed into OGR functions in some situations and on Windows this would sometimes crash a Python process (#303). Fiona now raises errors derived from UnicodeError when field names or field values can't be encoded.

1.5.66 1.6.2 (2015-09-22)

- Providing only PROJ4 representations in the dataset meta property resulted in loss of CRS information when using the *fiona.open(…, **src.meta) as dst* pattern (#265). This bug has been addressed by adding a crs_wkt item to the` meta property and extending the *fiona.open()* and the collection constructor to look for and prioritize this keyword argument.

1.5.67 1.6.1 (2015-08-12)

- Bug fix: Fiona now deserializes JSON-encoded string properties provided by the OGR GeoJSON driver (#244, #245, #246).
- Bug fix: proj4 data was not copied properly into binary distributions due to a typo (#254).

Special thanks to WFMU DJ Liz Berg for the awesome playlist that's fueling my release sprint. Check it out at https://wfmu.org/playlists/shows/62083. You can't unhear Love Coffin.

1.5.68 1.6.0 (2015-07-21)

- Upgrade Cython requirement to 0.22 (#214).
- New BytesCollection class (#215).
- Add GDAL's OpenFileGDB driver to registered drivers (#221).
- Implement CLI commands as plugins (#228).
- Raise click.abort instead of calling sys.exit, preventing suprising exits (#236).

1.5.69 1.5.1 (2015-03-19)

- Restore test data to sdists by fixing MANIFEST.in (#216).

1.5.70 1.5.0 (2015-02-02)

- Finalize GeoJSON feature sequence options (#174).
- Fix for reading of datasets that don't support feature counting (#190).
- New test dataset (#188).
- Fix for encoding error (#191).
- Remove confusing warning (#195).
- Add data files for binary wheels (#196).
- Add control over drivers enabled when reading datasets (#203).
- Use cligj for CLI options involving GeoJSON (#204).
- Fix fio-info –bounds help (#206).

1.5.71 1.4.8 (2014-11-02)

- Add missing crs_wkt property as in Rasterio (#182).

1.5.72 1.4.7 (2014-10-28)

- Fix setting of CRS from EPSG codes (#149).

1.5.73 1.4.6 (2014-10-21)

- Handle 3D coordinates in bounds() #178.

1.5.74 1.4.5 (2014-10-18)

- Add –bbox option to fio-cat (#163).
- Skip geopackage tests if run from an sdist (#167).
- Add fio-bounds and fio-distrib.
- Restore fio-dump to working order.

1.5.75 1.4.4 (2014-10-13)

- Fix accidental requirement on GDAL 1.11 introduced in 1.4.3 (#164).

1.5.76 1.4.3 (2014-10-10)

- Add support for geopackage format (#160).
- Add -f and –format aliases for –driver in CLI (#162).
- Add –version option and env command to CLI.

1.5.77 1.4.2 (2014-10-03)

- –dst-crs and –src-crs options for fio cat and collect (#159).

1.5.78 1.4.1 (2014-09-30)

- Fix encoding bug in collection's __getitem__ (#153).

1.5.79 1.4.0 (2014-09-22)

- Add fio cat and fio collect commands (#150).
- Return of Python 2.6 compatibility (#148).
- Improved CRS support (#149).

1.5.80 1.3.0 (2014-09-17)

- Add single metadata item accessors to fio inf (#142).
- Move fio to setuptools entry point (#142).
- Add fio dump and load commands (#143).
- Remove fio translate command.

1.5.81 1.2.0 (2014-09-02)

- Always show property width and precision in schema (#123).
- Write datetime properties of features (#125).
- Reset spatial filtering in filter() (#129).
- Accept datetime.date objects as feature properties (#130).
- Add slicing to collection iterators (#132).
- Add geometry object masks to collection iterators (#136).
- Change source layout to match Shapely and Rasterio (#138).

1.5.82 1.1.6 (2014-07-23)

- Implement Collection __getitem__() (#112).
- Leave GDAL finalization to the DLL's destructor (#113).
- Add Collection keys(), values(), items(), __contains__() (#114).
- CRS bug fix (#116).
- Add fio CLI program.

1.5.83 1.1.5 (2014-05-21)

- Addition of cpl_errs context manager (#108).
- Check for NULLs with '==' test instead of 'is' (#109).
- Open auxiliary files with encoding='utf-8' in setup for Python 3 (#110).

1.5.84 1.1.4 (2014-04-03)

- Convert 'long' in schemas to 'int' (#101).
- Carefully map Python schema to the possibly munged internal schema (#105).
- Allow writing of features with geometry: None (#71).

1.5.85 1.1.3 (2014-03-23)

- Always register all GDAL and OGR drivers when entering the DriverManager context (#80, #92).
- Skip unsupported field types with a warning (#91).
- Allow OGR config options to be passed to fiona.drivers() (#90, #93).
- Add a bounds() function (#100).
- Turn on GPX driver.

1.5.86 1.1.2 (2014-02-14)

- Remove collection slice left in dumpgj (#88).

1.5.87 1.1.1 (2014-02-02)

- Add an interactive file inspector like the one in rasterio.
- CRS to_string bug fix (#83).

1.5.88 1.1 (2014-01-22)

- Use a context manager to manage drivers (#78), a backwards compatible but big change. Fiona is now compatible with rasterio and plays better with the osgeo package.

1.5.89 1.0.3 (2014-01-21)

- Fix serialization of +init projections (#69).

1.5.90 1.0.2 (2013-09-09)

- Smarter, better test setup (#65, #66, #67).
- Add type='Feature' to records read from a Collection (#68).
- Skip geometry validation when using GeoJSON driver (#61).
- Dumpgj file description reports record properties as a list (as in dict.items()) instead of a dict.

1.5.91 1.0.1 (2013-08-16)

- Allow ordering of written fields and preservation of field order when reading (#57).

1.5.92 1.0 (2013-07-30)

- Add prop_type() function.
- Allow UTF-8 encoded paths for Python 2 (#51). For Python 3, paths must always be str, never bytes.
- Remove encoding from collection.meta, it's a file creation option only.
- Support for linking GDAL frameworks (#54).

1.5.93 0.16.1 (2013-07-02)

- Add listlayers, open, prop_width to __init__py:__all__.
- Reset reading of OGR layer whenever we ask for a collection iterator (#49).

1.5.94 0.16 (2013-06-24)

- Add support for writing layers to multi-layer files.
- Add tests to reach 100% Python code coverage.

1.5.95 0.15 (2013-06-06)

- Get and set numeric field widths (#42).
- Add support for multi-layer data sources (#17).
- Add support for zip and tar virtual filesystems (#45).
- Add listlayers() function.
- Add GeoJSON to list of supported formats (#47).
- Allow selection of layers by index or name.

1.5.96 0.14 (2013-05-04)

- Add option to add JSON-LD in the dumpgj program.
- Compare values to six.string_types in Collection constructor.
- Add encoding to Collection.meta.
- Document dumpgj in README.

1.5.97 0.13 (2013-04-30)

- Python 2/3 compatibility in a single package. Pythons 2.6, 2.7, 3.3 now supported.

1.5.98 0.12.1 (2013-04-16)

- Fix messed up linking of README in sdist (#39).

1.5.99 0.12 (2013-04-15)

- Fix broken installation of extension modules (#35).
- Log CPL errors at their matching Python log levels.
- Use upper case for encoding names within OGR, lower case in Python.

1.5.100 0.11 (2013-04-14)

- Cythonize .pyx files (#34).
- Work with or around OGR's internal recoding of record data (#35).
- Fix bug in serialization of int/float PROJ.4 params.

1.5.101 0.10 (2013-03-23)

- Add function to get the width of str type properties.
- Handle validation and schema representation of 3D geometry types (#29).
- Return {'geometry': None} in the case of a NULL geometry (#31).

1.5.102 0.9.1 (2013-03-07)

- Silence the logger in ogrext.so (can be overridden).
- Allow user specification of record field encoding (like 'Windows-1252' for Natural Earth shapefiles) to help when OGR can't detect it.

1.5.103 0.9 (2013-03-06)

- Accessing file metadata (crs, schema, bounds) on never inspected closed files returns None without exceptions.
- Add a dict of supported_drivers and their supported modes.
- Raise ValueError for unsupported drivers and modes.
- Remove asserts from ogrext.pyx.
- Add validate_record method to collections.
- Add helpful coordinate system functions to fiona.crs.
- Promote use of fiona.open over fiona.collection.
- Handle Shapefile's mix of LineString/Polygon and multis (#18).
- Allow users to specify width of shapefile text fields (#20).

1.5.104 0.8 (2012-02-21)

- Replaced .opened attribute with .closed (product of collection() is always opened). Also a __del__() which will close a Collection, but still not to be depended upon.
- Added writerecords method.
- Added a record buffer and better counting of records in a collection.
- Manage one iterator per collection/session.
- Added a read-only bounds property.

1.5.105 0.7 (2012-01-29)

- Initial timezone-naive support for date, time, and datetime fields. Don't use these field types if you can avoid them. RFC 3339 datetimes in a string field are much better.

1.5.106 0.6.2 (2012-01-10)

- Diagnose and set the driver property of collection in read mode.
- Fail if collection paths are not to files. Multi-collection workspaces are a (maybe) TODO.

1.5.107 0.6.1 (2012-01-06)

- Handle the case of undefined crs for disk collections.

1.5.108 0.6 (2012-01-05)

- Support for collection coordinate reference systems based on Proj4.
- Redirect OGR warnings and errors to the Fiona log.
- Assert that pointers returned from the ograpi functions are not NULL before using.

1.5.109 0.5 (2011-12-19)

- Support for reading and writing collections of any geometry type.
- Feature and Geometry classes replaced by mappings (dicts).
- Removal of Workspace class.

1.5.110 0.2 (2011-09-16)

- Rename WorldMill to Fiona.

1.5.111 0.1.1 (2008-12-04)

- Support for features with no geometry.

1.6 Credits

Fiona is written by:

- Alan D. Snow <alansnow21@gmail.com>
- Ariel Nunez <ingenieroariel@gmail.com>
- Ariki <Ariki@users.noreply.github.com>
- Bas Couwenberg <sebastic@xs4all.nl>
- Brandon Liu <bdon@bdon.org>
- Brendan Ward <bcward@consbio.org>
- Chris Mutel <cmutel@gmail.com>
- Denis Rykov <rykovd@gmail.com>
- dimlev <dimlev@gmail.com>

- Efrén <chefren@users.noreply.github.com>
- Egor Fedorov <egor.fedorov@emlid.com>
- Elliott Sales de Andrade <quantum.analyst@gmail.com>
- Even Rouault <even.rouault@mines-paris.org>
- Ewout ter Hoeven <E.M.terHoeven@student.tudelft.nl>
- Filipe Fernandes <ocefpaf@gmail.com>
- fredj <frederic.junod@camptocamp.com>
- Géraud <galak75@users.noreply.github.com>
- Hannes Gräuler <hgraeule@uos.de>
- Jacob Wasserman <jwasserman@gmail.com>
- Jesse Crocker <jesse@gaiagps.com>
- Johan Van de Wauw <johan.vandewauw@gmail.com>
- Joris Van den Bossche <jorisvandenbossche@gmail.com>
- Joshua Arnott <josh@snorfalorpagus.net>
- Juan Luis Cano Rodríguez <Juanlu001@users.noreply.github.com>
- Kelsey Jordahl <kjordahl@enthought.com>
- Kevin Wurster <wursterk@gmail.com>
- Ludovic Delauné <ludotux@gmail.com>
- Martijn Visser <mgvisser@gmail.com>
- Matthew Perry <perrygeo@gmail.com>
- Micah Cochran <micahcochran@users.noreply.github.com>
- Michael Weisman <mweisman@gmail.com>
- Michele Citterio <michele@citterio.net>
- Mike Taves <mwtoews@gmail.com>
- Miro Hrončok <miro@hroncok.cz>
- Oliver Tonnhofer <olt@bogosoft.com>
- Patrick Young <patrick.mckendree.young@gmail.com>
- qinfeng <guo.qinfeng+github@gmail.com>
- René Buffat <buffat@gmail.com>
- Ryan Grout <rgrout@continuum.io>
- Sean Gillies <sean.gillies@gmail.com>
- Sid Kapur <sid-kap@users.noreply.github.com>
- Simon Norris <snorris@hillcrestgeo.ca>
- Stefano Costa <steko@iosa.it>
- Stephane Poss <stephposs@gmail.com>
- Tim Tröndle <tim.troendle@usys.ethz.ch>

- wilsaj <wilson.andrew.j+github@gmail.com>

The GeoPandas project (Joris Van den Bossche et al.) has been a major driver for new features in 1.8.0.

Fiona would not be possible without the great work of Frank Warmerdam and other GDAL/OGR developers.

Some portions of this work were supported by a grant (for Pleiades) from the U.S. National Endowment for the Humanities (https://www.neh.gov).

CHAPTER

TWO

INSTALLATION

Installation of the Fiona package is complicated by its dependency on libgdal and other C libraries. There are easy installations paths and an advanced installation path.

2.1 Easy installation

Fiona has several extension modules which link against libgdal. This complicates installation. Binary distributions (wheels) containing libgdal and its own dependencies are available from the Python Package Index and can be installed using pip.

```
pip install fiona
```

These wheels are mainly intended to make installation easy for simple applications, not so much for production. They are not tested for compatibility with all other binary wheels, conda packages, or QGIS, and omit many of GDAL's optional format drivers. If you need, for example, GML support you will need to build and install Fiona from a source distribution.

Many users find Anaconda and conda-forge a good way to install Fiona and get access to more optional format drivers (like GML).

Fiona 1.9 (coming soon) requires Python 3.7 or higher and GDAL 3.2 or higher.

2.2 Advanced installation

Once GDAL and its dependencies are installed on your computer (how to do this is documented at https://gdal.org) Fiona can be built and installed using setuptools or pip. If your GDAL installation provides the `gdal-config` program, the process is simpler.

Without pip:

```
GDAL_CONFIG=/path/to/gdal-config python setup.py install
```

With pip (version >= 22.3 is required):

```
python -m pip install --user -U pip
GDAL_CONFIG=/path/to/gdal-config python -m pip install --user .
```

These are pretty much equivalent. Pip will use setuptools as the build backend. If the gdal-config program is on your executable path, then you don't need to set the environment variable.

Without gdal-config you will need to configure header and library locations for the build in another way. One way to do this is to create a setup.cfg file in the source directory with content like this:

```
[build_ext]
include_dirs = C:/vcpkg/installed/x64-windows/include
libraries = gdal
library_dirs = C:/vcpkg/installed/x64-windows/lib
```

This is the approach taken by Fiona's wheel-building workflow. With this file in place you can run either `python setup.py install` or `python -m pip install --user .`.

You can also pass those three values on the command line following the setuptools documentation. However, the setup.cfg approach is easier.

CHAPTER

THREE

1 THE FIONA USER MANUAL

Author
Sean Gillies, <sean.gillies@gmail.com>

Version

Date
May 23, 2023

Abstract
Fiona is OGR's neat, nimble, no-nonsense API. This document explains how to use the Fiona package for reading and writing geospatial data files. Python 3 is used in examples. See the README for installation and quick start instructions.

3.1 1.1 Introduction

Geographic information systems (GIS) help us plan, react to, and understand changes in our physical, political, economic, and cultural landscapes. A generation ago, GIS was something done only by major institutions like nations and cities, but it's become ubiquitous today thanks to accurate and inexpensive global positioning systems, commoditization of satellite imagery, and open source software.

The kinds of data in GIS are roughly divided into *rasters* representing continuous scalar fields (land surface temperature or elevation, for example) and *vectors* representing discrete entities like roads and administrative boundaries. Fiona is concerned exclusively with the latter. It is a Python wrapper for vector data access functions from the GDAL/OGR library. A very simple wrapper for minimalists. It reads data records from files as GeoJSON-like mappings and writes the same kind of mappings as records back to files. That's it. There are no layers, no cursors, no geometric operations, no transformations between coordinate systems, no remote method calls; all these concerns are left to other Python packages such as `Shapely` and `pyproj` and Python language protocols. Why? To eliminate unnecessary complication. Fiona aims to be simple to understand and use, with no gotchas.

Please understand this: Fiona is designed to excel in a certain range of tasks and is less optimal in others. Fiona trades memory and speed for simplicity and reliability. Where OGR's Python bindings (for example) use C pointers, Fiona copies vector data from the data source to Python objects. These are simpler and safer to use, but more memory intensive. Fiona's performance is relatively more slow if you only need access to a single record field – and of course if you just want to reproject or filter data files, nothing beats the **`ogr2ogr`** program – but Fiona's performance is much better than OGR's Python bindings if you want *all* fields and coordinates of a record. The copying is a constraint, but it simplifies programs. With Fiona, you don't have to track references to C objects to avoid crashes, and you can work with vector data using familiar Python mapping accessors. Less worry, less time spent reading API documentation.

3.1.1 1.1.1 Rules of Thumb

In what cases would you benefit from using Fiona?

- If the features of interest are from or destined for a file in a non-text format like ESRI Shapefiles, Mapinfo TAB files, etc.
- If you're more interested in the values of many feature properties than in a single property's value.
- If you're more interested in all the coordinate values of a feature's geometry than in a single value.
- If your processing system is distributed or not contained to a single process.

In what cases would you not benefit from using Fiona?

- If your data is in or destined for a JSON document you should use Python's `json` or `simplejson` modules.
- If your data is in a RDBMS like PostGIS, use a Python DB package or ORM like `SQLAlchemy` or `GeoAlchemy`. Maybe you're using `GeoDjango` already. If so, carry on.
- If your data is served via HTTP from CouchDB or CartoDB, etc, use an HTTP package (`httplib2`, `Requests`, etc) or the provider's Python API.
- If you can use **`ogr2ogr`**, do so.

3.1.2 1.1.2 Example

The first example of using Fiona is this: copying features (another word for record) from one file to another, adding two attributes and making sure that all polygons are facing "up". Orientation of polygons is significant in some applications, extruded polygons in Google Earth for one. No other library (like `Shapely`) is needed here, which keeps it uncomplicated. There's a `coutwildrnp.zip` file in the Fiona repository for use in this and other examples.

```
import datetime

import fiona
from fiona import Geometry, Feature, Properties


def signed_area(coords):
    """Return the signed area enclosed by a ring using the linear time
    algorithm at http://www.cgafaq.info/wiki/Polygon_Area. A value >= 0
    indicates a counter-clockwise oriented ring.
    """
    xs, ys = map(list, zip(*coords))
    xs.append(xs[1])
    ys.append(ys[1])
    return sum(xs[i] * (ys[i + 1] - ys[i - 1]) for i in range(1, len(coords))) / 2.0


with fiona.open(
    "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip"
) as src:

    # Copy the source schema and add two new properties.
    dst_schema = src.schema
    dst_schema["properties"]["signed_area"] = "float"
    dst_schema["properties"]["timestamp"] = "datetime"
```

(continues on next page)

(continued from previous page)

```
    # Create a sink for processed features with the same format and
    # coordinate reference system as the source.
    with fiona.open(
        "example.gpkg",
        mode="w",
        layer="oriented-ccw",
        crs=src.crs,
        driver="GPKG",
        schema=dst_schema,
    ) as dst:
        for feat in src:
            # If any feature's polygon is facing "down" (has rings
            # wound clockwise), its rings will be reordered to flip
            # it "up".
            geom = feat.geometry
            assert geom.type == "Polygon"
            rings = geom.coordinates
            sa = sum(signed_area(ring) for ring in rings)

            if sa < 0.0:
                rings = [r[::-1] for r in rings]
                geom = Geometry(type=geom.type, coordinates=rings)

            # Add the signed area of the polygon and a timestamp
            # to the feature properties map.
            props = Properties.from_dict(
                **feat.properties,
                signed_area=sa,
                timestamp=datetime.datetime.now().isoformat()
            )

            dst.write(Feature(geometry=geom, properties=props))
```

3.2 1.2 Data Model

Discrete geographic features are usually represented in geographic information systems by *records*. The characteristics of records and their semantic implications are well known [Kent1978]. Among those most significant for geographic data: records have a single type, all records of that type have the same fields, and a record's fields concern a single geographic feature. Different systems model records in different ways, but the various models have enough in common that programmers have been able to create useful abstract data models. The OGR model is one. Its primary entities are *Data Sources*, *Layers*, and *Features*. Features have not fields, but attributes and a *Geometry*. An OGR Layer contains Features of a single type ("roads" or "wells", for example). The GeoJSON model is a bit more simple, keeping Features and substituting *Feature Collections* for OGR Data Sources and Layers. The term "Feature" is thus overloaded in GIS modeling, denoting entities in both our conceptual and data models.

Various formats for record files exist. The *ESRI Shapefile* [ESRI1998] has been, at least in the United States, the most significant of these up to about 2005 and remains popular today. It is a binary format. The shape fields are stored in one .shp file and the other fields in another .dbf file. The GeoJSON [GeoJSON] format, from 2008, proposed a human readable text format in which geometry and other attribute fields are encoded together using *Javascript Object Notation* [JSON]. In GeoJSON, there's a uniformity of data access. Attributes of features are accessed in the same manner as

attributes of a feature collection. Coordinates of a geometry are accessed in the same manner as features of a collection.

The GeoJSON format turns out to be a good model for a Python API. JSON objects and Python dictionaries are semantically and syntactically similar. Replacing object-oriented Layer and Feature APIs with interfaces based on Python mappings provides a uniformity of access to data and reduces the amount of time spent reading documentation. A Python programmer knows how to use a mapping, so why not treat features as dictionaries? Use of existing Python idioms is one of Fiona's major design principles.

TL;DR

Fiona subscribes to the conventional record model of data, but provides GeoJSON-like access to the data via Python file-like and mapping protocols.

3.3 1.3 Reading Vector Data

Reading a GIS vector file begins by opening it in mode `'r'` using Fiona's *open()* function. It returns an opened *Collection* object.

```
>>> import fiona
>>> colxn = fiona.open("zip+https://github.com/Toblerity/Fiona/files/11151652/
↪coutwildrnp.zip", "r")
>>> colxn
<open Collection '/vsizip/vsicurl/https://github.com/Toblerity/Fiona/files/11151652/
↪coutwildrnp.zip:coutwildrnp', mode 'r' at 0x7f9555af8f50>
>>> collection.closed
False
```

Mode `'r'` is the default and will be omitted in following examples.

Fiona's *Collection* is like a Python `file`, but is iterable for records rather than lines.

```
>>> next(iter(colxn))
{'geometry': {'type': 'Polygon', 'coordinates': ...
>>> len(list(colxn))
67
```

Note that `list()` iterates over the entire collection, effectively emptying it as with a Python `file`.

```
>>> next(iter(colxn))
Traceback (most recent call last):
...
StopIteration
>>> len(list(colxn))
0
```

Seeking the beginning of the file is not supported. You must reopen the collection to get back to the beginning.

```
>>> colxn = fiona.open("zip+https://github.com/Toblerity/Fiona/files/11151652/
↪coutwildrnp.zip")
>>> len(list(colxn))
67
```

File Encoding

The format drivers will attempt to detect the encoding of your data, but may fail. In my experience GDAL 1.7.2 (for example) doesn't detect that the encoding of the Natural Earth dataset is Windows-1252. In this case, the proper encoding can be specified explicitly by using the `encoding` keyword parameter of *fiona.open()*: `encoding='Windows-1252'`.

New in version 0.9.1.

3.3.1 1.3.1 Collection indexing

Features of a collection may also be accessed by index.

```
>>> with fiona.open("zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.
↪zip") as colxn:
...     print(colxn[1])
...
<fiona.model.Feature object at 0x7f954bfc5f50>
```

Note that these indices are controlled by GDAL, and do not always follow Python conventions. They can start from 0, 1 (e.g. geopackages), or even other values, and have no guarantee of contiguity. Negative indices will only function correctly if indices start from 0 and are contiguous.

New in version 1.1.6

3.3.2 1.3.2 Closing Files

A *Collection* involves external resources. There's no guarantee that these will be released unless you explicitly *close()* the object or use a `with` statement. When a *Collection* is a context guard, it is closed no matter what happens within the block.

```
>>> try:
...     with fiona.open("zip+https://github.com/Toblerity/Fiona/files/11151652/
↪coutwildrnp.zip") as colxn:
...         print(len(list(colxn)))
...         assert True is False
... except Exception:
...     print(colxn.closed)
...     raise
...
67
True
Traceback (most recent call last):
  ...
AssertionError
```

An exception is raised in the `with` block above, but as you can see from the print statement in the `except` clause `colxn.__exit__()` (and thereby `colxn.close()`) has been called.

Important: Always call *close()* or use `with` and you'll never stumble over tied-up external resources, locked files, etc.

3.4 1.4 Format Drivers, CRS, Bounds, and Schema

In addition to attributes like those of `file` (`name`, `mode`, `closed`), a *Collection* has a read-only *driver* attribute which names the **OGR** *format driver* used to open the vector file.

```
>>> colxn = fiona.open("zip+https://github.com/Toblerity/Fiona/files/11151652/
↪coutwildrnp.zip")
>>> colxn.driver
'ESRI Shapefile'
```

The *coordinate reference system* (CRS) of the collection's vector data is accessed via a read-only *crs* attribute.

```
>>> colxn.crs
CRS.from_epsg(4326)
```

The *fiona.crs* module provides 3 functions to assist with these mappings. *to_string()* converts mappings to PROJ.4 strings:

```
>>> from fiona.crs import to_string
>>> to_string(colxn.crs)
'EPSG:4326'
```

from_string() does the inverse.

```
>>> from fiona.crs import from_string
>>> from_string("+datum=WGS84 +ellps=WGS84 +no_defs +proj=longlat")
CRS.from_epsg(4326)
```

from_epsg() is a shortcut to CRS mappings from EPSG codes.

```
>>> from fiona.crs import from_epsg
>>> from_epsg(3857)
CRS.from_epsg(3857)
```

No Validation

Both *from_epsg()* and *from_string()* simply restructure data, they do not ensure that the resulting mapping is a pre-defined or otherwise valid CRS in any way.

The number of records in the collection's file can be obtained via Python's built in `len()` function.

```
>>> len(colxn)
67
```

The *minimum bounding rectangle* (MBR) or *bounds* of the collection's records is obtained via a read-only *bounds* attribute.

```
>>> colxn.bounds
(-113.56424713134766, 37.0689811706543, -104.97087097167969, 41.99627685546875)
```

Finally, the schema of its record type (a vector file has a single type of record, remember) is accessed via a read-only *schema* attribute. It has 'geometry' and 'properties' items. The former is a string and the latter is an ordered dict with items having the same order as the fields in the data file.

```
  >>> import pprint
>>> pprint.pprint(colxn.schema)
{'geometry': 'Polygon',
  'properties': {'AGBUR': 'str:80',
                 'AREA': 'float:24.15',
                 'FEATURE1': 'str:80',
                'FEATURE2': 'str:80',
                'NAME': 'str:80',
                  'PERIMETER': 'float:24.15',
                'STATE': 'str:80',
                'STATE_FIPS': 'str:80',
                'URL': 'str:101',
                'WILDRNP020': 'int:10'}}
```

3.4.1 1.4.1 Keeping Schemas Simple

Fiona takes a less is more approach to record types and schemas. Data about record types is structured as closely to data about records as can be done. Modulo a record's 'id' key, the keys of a schema mapping are the same as the keys of the collection's record mappings.

```
>>> feat = next(iter(colxn))
>>> set(feat.keys()) - set(colxn.schema.keys())
{'id'}
>>> set(feat['properties'].keys()) == set(colxn.schema['properties'].keys())
True
```

The values of the schema mapping are either additional mappings or field type names like 'Polygon', 'float', and 'str'. The corresponding Python types can be found in a dictionary named `fiona.FIELD_TYPES_MAP`.

```
 >>> pprint.pprint(fiona.FIELD_TYPES_MAP)
 {'List[str]': typing.List[str],
   'bytes': <class 'bytes'>,
   'date': <class 'fiona.rfc3339.FionaDateType'>,
'datetime': <class 'fiona.rfc3339.FionaDateTimeType'>,
'float': <class 'float'>,
'int': <class 'int'>,
'int32': <class 'int'>,
'int64': <class 'int'>,
'str': <class 'str'>,
'time': <class 'fiona.rfc3339.FionaTimeType'>}
```

3.4.2 1.4.2 Field Types

In a nutshell, the types and their names are as near to what you'd expect in Python (or Javascript) as possible. Since Python 3, the 'str' field type may contain Unicode characters.

```
>>> type(feat.properties['NAME'])
<class 'str'>
>>> colxn.schema['properties']['NAME']
'str'
```

(continues on next page)

(continued from previous page)

```
>>> fiona.FIELD_TYPES_MAP[colxn.schema['properties']['NAME']]
<class 'str'>
```

String type fields may also indicate their maximum width. A value of 'str:25' indicates that all values will be no longer than 25 characters. If this value is used in the schema of a file opened for writing, values of that property will be truncated at 25 characters. The default width is 80 chars, which means 'str' and 'str:80' are more or less equivalent.

Fiona provides a function to get the width of a property.

```
>>> from fiona import prop_width
>>> prop_width('str:25')
25
>>> prop_width('str')
80
```

Another function gets the proper Python type of a property.

```
>>> from fiona import prop_type
>>> prop_type('int')
<type 'int'>
>>> prop_type('float')
<type 'float'>
>>> prop_type('str:25')
<class 'str'>
```

3.4.3 1.4.3 Geometry Types

Fiona supports the geometry types in GeoJSON and their 3D variants. This means that the value of a schema's geometry item will be one of the following:

- Point
- LineString
- Polygon
- MultiPoint
- MultiLineString
- MultiPolygon
- GeometryCollection
- 3D Point
- 3D LineString
- 3D Polygon
- 3D MultiPoint
- 3D MultiLineString
- 3D MultiPolygon
- 3D GeometryCollection

The last seven of these, the 3D types, apply only to collection schema. The geometry types of features are always one of the first seven. A '3D Point' collection, for example, always has features with geometry type 'Point'. The coordinates of those geometries will be (x, y, z) tuples.

Note that one of the most common vector data formats, Esri's Shapefile, has no 'MultiLineString' or 'MultiPolygon' schema geometries. However, a Shapefile that indicates 'Polygon' in its schema may yield either 'Polygon' or 'MultiPolygon' features.

3.5 1.5 Features

A record you get from a collection is structured like a GeoJSON Feature. Fiona records are self-describing; the names of its fields are contained within the data structure and the values in the fields are typed properly for the type of record. Numeric field values are instances of type `int` and `float`, for example, not strings.

The record data has no references to the *Collection* from which it originates or to any other external resource. It's entirely independent and safe to use in any way. Closing the collection does not affect the record at all.

Features are mappings, not dicts

In Fiona versions before 1.9.0 features were Python dicts, mutable and JSON serializable. Since 1.9.0 features are mappings and not immediately JSON serializable.

Instances of Feature can be converted to dicts with `fiona.model.to_dict()` or serialized using the json module and `fiona.model.ObjectEncoder`.

3.5.1 1.5.1 Feature Id

A feature has an `id` attribute. As in the GeoJSON specification, its corresponding value is a string unique within the data file.

```
>>> colxn = fiona.open("zip+https://github.com/Toblerity/Fiona/files/11151652/
↪coutwildrnp.zip")
>>> feat = next(iter(colxn))
>>> feat.id
'0'
```

OGR Details

In the **OGR** model, feature ids are long integers. Fiona record ids are therefore usually string representations of integer record indexes.

3.5.2 1.5.2 Feature Properties

A feature has a `properties` attribute. Its value is a mapping. The keys of the properties mapping are the same as the keys of the properties mapping in the schema of the collection the record comes from (see above).

```
>>> for k, v in feat.properties.items():
...     print(k, v)
...
PERIMETER 1.22107
FEATURE2 None
NAME Mount Naomi Wilderness
FEATURE1 Wilderness
URL http://www.wilderness.net/index.cfm?fuse=NWPS&sec=wildView&wname=Mount%20Naomi
AGBUR FS
AREA 0.0179264
STATE_FIPS 49
WILDRNP020 332
STATE UT
```

3.5.3 1.5.3 Feature Geometry

A feature has a `geometry` attribute. Its value is a mapping with `type` and `coordinates` keys.

```
>>> feat.geometry["type"]
'Polygon'
>>> feat.geometry["coordinates"]
[[(-111.73527526855469, 41.995094299316406), ..., (-111.73527526855469, 41.
↪995094299316406)]]
```

Since the coordinates are just tuples, or lists of tuples, or lists of lists of tuples, the `type` tells you how to interpret them.

Type	Coordinates
Point	A single (x, y) tuple
LineString	A list of (x, y) tuple vertices
Polygon	A list of rings (each a list of (x, y) tuples)
MultiPoint	A list of points (each a single (x, y) tuple)
MultiLineString	A list of lines (each a list of (x, y) tuples)
MultiPolygon	A list of polygons (see above)

Fiona, like the GeoJSON format, has both Northern Hemisphere "North is up" and Cartesian "X-Y" biases. The values within a tuple that denoted as `(x, y)` above are either (longitude E of the prime meridian, latitude N of the equator) or, for other projected coordinate systems, (easting, northing).

Long-Lat, not Lat-Long

Even though most of us say "lat, long" out loud, Fiona's `x,y` is always easting, northing, which means `(long, lat)`. Longitude first and latitude second, consistent with the GeoJSON format specification.

3.5.4 1.5.4 Point Set Theory and Simple Features

In a proper, well-scrubbed vector data file the geometry mappings explained above are representations of geometric objects made up of *point sets*. The following

```
{"type": "LineString", "coordinates": [(0.0, 0.0), (0.0, 1.0)]}
```

represents not just two points, but the set of infinitely many points along the line of length 1.0 from `(0.0, 0.0)` to `(0.0, 1.0)`. In the application of point set theory commonly called *Simple Features Access* [SFA] two geometric objects are equal if their point sets are equal whether they are equal in the Python sense or not. If you have Shapely (which implements Simple Features Access) installed, you can see this in by verifying the following.

```
>>> from shapely.geometry import shape
>>> l1 = shape(
...     {'type': 'LineString', 'coordinates': [(0, 0), (2, 2)]})
>>> l2 = shape(
...     {'type': 'LineString', 'coordinates': [(0, 0), (1, 1), (2, 2)]})
>>> l1 == l2
False
>>> l1.equals(l2)
True
```

Dirty data

Some files may contain vectors that are *invalid* from a simple features standpoint due to accident (inadequate quality control on the producer's end), intention ("dirty" vectors saved to a file for special treatment) or discrepancies of the numeric precision models (Fiona can't handle fixed precision models yet). Fiona doesn't sniff for or attempt to clean dirty data, so make sure you're getting yours from a clean source.

3.6 1.6 Writing Vector Data

A vector file can be opened for writing in mode `'a'` (append) or mode `'w'` (write).

Note

The in situ "update" mode of **OGR** is quite format dependent and is therefore not supported by Fiona.

3.6.1 1.6.1 Appending Data to Existing Files

Let's start with the simplest if not most common use case, adding new records to an existing file.

```
$ wget https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip
$ unzip coutwildrnp.zip
```

The coordinate reference system. format, and schema of the file are already defined, so it's opened with just two arguments as for reading, but in `'a'` mode. The new record is written to the end of the file using the *`write()`* method. Accordingly, the length of the file grows from 67 to 68.

```
>>> with fiona.open("coutwildrnp.shp", "a") as dst:
...     print(len(dst))
...     with fiona.open("zip+https://github.com/Toblerity/Fiona/files/11151652/
↪coutwildrnp.zip") as src:
...         feat = src[0]
...         print(feat.id, feat.properties["NAME"])
...         dst.write(feat)
...     print(len(c))
...
67
('0', 'Mount Naomi Wilderness')
68
```

The feature you write must match the file's schema (because a file contains one type of record, remember). You'll get a `ValueError` if it doesn't.

```
>>> with fiona.open("coutwildrnp.shp", "a") as dst:
...     dst.write({'properties': {'foo': 'bar'}})
...
Traceback (most recent call last):
  .}..
ValueError: Record data not match collection schema
```

Now, what about record ids? The id of a record written to a file is ignored and replaced by the next value appropriate for the file. If you read the file just appended to above,

```
>>> with fiona.open("coutwildrnp.shp") as colxn:
...     feat = colxn[-1]
...
>>> feat.id
'67'
>>> feat.properties["NAME"]
'Mount Naomi Wilderness'
```

You'll see that the id of `'0'` which the record had when written is replaced by `'67'`.

The *write()* method writes a single record to the collection's file. Its sibling *writerecords()* writes a sequence (or iterator) of records.

```
>>> with fiona.open("coutwildrnp.shp", "a") as colxn:
...     colxn.writerecords([feat, feat, feat])
...     print(len(colxn))
...
71
```

Duplication

Fiona's collections do not guard against duplication. The code above will write 3 duplicate records to the file, and they will be given unique sequential ids.

Transactions

Fiona uses transactions during write operations to ensure data integrity. `writerecords()` will start and commit one transaction. If there are lots of records, intermediate commits will be performed at reasonable intervals.

Depending on the driver, a transaction can be a very costly operation. Since `write()` is just a thin convenience wrapper that calls `writerecords()` with a single record, you may experience significant performance issue if you write lots of features one by one using this method. Consider preparing your data first and then writing it in a single call to `writerecords()`.

Buffering

Fiona's output is buffered. The records passed to `write()` and `writerecords()` are flushed to disk when the collection is closed. You may also call `flush()` periodically to write the buffer contents to disk.

Format requirements

Format drivers may have specific requirements about what they store. For example, the Shapefile driver may "fix" topologically invalid features.

3.6.2 1.6.2 Creating files of the same structure

Writing a new file is more complex than appending to an existing file because the file CRS, format, and schema have not yet been defined and must be done so by the programmer. Still, it's not very complicated. A schema is just a mapping, as described above. The possible formats are enumerated in the `fiona.supported_drivers` dictionary.

Review the parameters of our demo file.

```
>>> with fiona.open(
...     "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip"
) as src:
...     driver = src.driver
...     crs = src.crs
...     schema = src.schema
...     feat = src[1]
...
>>> driver
'ESRI Shapefile'
>>> crs
CRS.from_epsg(4326)
>>> pprint.pprint(schema)
{'geometry': 'Polygon',
 'properties': {'AGBUR': 'str:80',
                'AREA': 'float:24.15',
                'FEATURE1': 'str:80',
                'FEATURE2': 'str:80',
                'NAME': 'str:80',
                'PERIMETER': 'float:24.15',
                'STATE': 'str:80',
                'STATE_FIPS': 'str:80',
                'URL': 'str:101',
                'WILDRNP020': 'int:10'}}
```

We can create a new file using them.

```
>>> with fiona.open("example.shp", "w", driver=driver, crs=crs, schema=schema) as dst:
...     print(len(dst))
...     dst.write(feat)
...     print(len(dst))
...
0
1
>>> dst.closed
True
>>> len(dst)
1
```

Because the properties of the source schema are ordered and are passed in the same order to the write-mode collection, the written file's fields have the same order as those of the source file.

The *profile* attribute makes duplication of a file's meta properties even easier.

```
>>> src = fiona.open("zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.
↪zip")
>>> dst = fiona.open("example.shp", "w", **src.profile)
```

3.6.3 1.6.3 Writing new files from scratch

To write a new file from scratch we have to define our own specific driver, crs and schema.

To ensure the order of the attribute fields is predictable, in both the schema and the actual manifestation as feature attributes, we will use ordered dictionaries.

```

```

Consider the following record, structured in accordance to the Python geo protocol, representing the Eiffel Tower using a point geometry with UTM coordinates in zone 31N.

```
>>> eiffel_tower =  {
...   'geometry': {
...     'type': 'Point',
...     'coordinates': (448252, 5411935)
...   },
...   'properties': dict([
...     ('name', 'Eiffel Tower'),
...     ('height', 300.01),
...     ('view', 'scenic'),
...     ('year', 1889)
...   ])
... }
```

A corresponding scheme could be:

```
>>> landmarks_schema = {
...   'geometry': 'Point',
...   'properties': dict([
...     ('name', 'str'),
```

(continues on next page)

(continued from previous page)

```
...     ('height', 'float'),
...     ('view', 'str'),
...     ('year', 'int')
...   ])
... }
```

The coordinate reference system of these landmark coordinates is ETRS89 / UTM zone 31N which is referenced in the EPSG database as EPSG:25831.

```
>>> from fiona.crs import CRS
>>> landmarks_crs = CRS.from_epsg(25831)
```

An appropriate driver could be:

```
>>> driver = "GeoJSON"
```

Having specified schema, crs and driver, we are ready to open a file for writing our record:

```
>>> with fiona.open(
...     "landmarks.geojson",
...     "w",
...     driver="GeoJSON",
...     crs=CRS.from_epsg(25831),
...     schema=landmarks_schema
... ) as colxn:
...     colxn.write(eiffel_tower)
...
```

1.6.3.1 Ordering Record Fields

Beginning with Fiona 1.0.1, the 'properties' item of *fiona.open()*'s 'schema' keyword argument may be an ordered dict or a list of (key, value) pairs, specifying an ordering that carries into written files. If an ordinary dict is given, the ordering is determined by the output of that dict's `items()` method.

3.6.4 1.6.4 3D Coordinates and Geometry Types

If you write 3D coordinates, ones having (x, y, z) tuples, to a 2D file ('Point' schema geometry, for example) the z values will be lost.

```
>>> feat = {"geometry": {"type": "Point", "coordinates": (-1, 1, 5)}}
>>> with fiona.open(
...     "example.shp",
...     "w",
...     driver="Shapefile",
...     schema={"geometry": "Point", "properties": {}}
... ) as dst:
...     dst.write(feat)
...
>>> with fiona.open("example.shp") as src:
...     print(src[0].geometry.coordinates)
```

(continues on next page)

(continued from previous page)

```
...
(-1.0, 1.0)
```

If you write 2D coordinates, ones having only (x, y) tuples, to a 3D file ('3D Point' schema geometry, for example) a default z value of 0 will be provided.

```
>>> feat = {"geometry": {"type": "Point", "coordinates": (-1, 1)}}
>>> with fiona.open(
...     "example.shp",
...     "w",
...     driver="Shapefile",
...     schema={"geometry": "3D Point", "properties": {}}
... ) as dst:
...     dst.write(feat)
...
>>> with fiona.open("example.shp") as src:
...     print(src[0].geometry.coordinates)
...
(-1.0, 1.0, 0.0)
```

3.7 1.7 Advanced Topics

3.7.1 1.7.1 OGR configuration options

GDAL/OGR has a large number of features that are controlled by global or thread-local configuration options. Fiona allows you to configure these options using a context manager, `fiona.Env`. This class's constructor takes GDAL/OGR configuration options as keyword arguments. To see debugging information from GDAL/OGR, for example, you may do the following.

```
import logging
import fiona

logging.basicConfig(level=logging.DEBUG)

with fiona.Env(CPL_DEBUG=True):
    fiona.open(
        "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip"
    )
```

The following extra messages will appear in the Python logger's output.

```
DEBUG:fiona._env:CPLE_None in GNM: GNMRegisterAllInternal
DEBUG:fiona._env:CPLE_None in GNM: RegisterGNMFile
DEBUG:fiona._env:CPLE_None in GNM: RegisterGNMdatabase
DEBUG:fiona._env:CPLE_None in GNM: GNMRegisterAllInternal
DEBUG:fiona._env:CPLE_None in GNM: RegisterGNMFile
DEBUG:fiona._env:CPLE_None in GNM: RegisterGNMdatabase
DEBUG:fiona._env:CPLE_None in GDAL: GDALOpen(tests/data/coutwildrnp.shp, this=0x1683930)␣
↪succeeds as ESRI Shapefile.
```

If you call `fiona.open()` with no surrounding `Env` environment, one will be created for you.

When your program exits the environment's `with` block the configuration reverts to its previous state.

3.7.2 1.7.2 Driver configuration options

Drivers can have dataset open, dataset creation, respectively layer creation options. These options can be found on the drivers page on GDAL's homepage. or using the `fiona.meta` module:

```
>>> import fiona.meta
>>> fiona.meta.print_driver_options("GeoJSON")
```

These options can be passed to `fiona.open`:

```
import fiona
fiona.open('tests/data/coutwildrnp.json', ARRAY_AS_STRING="YES")
```

3.7.3 1.7.3 Cloud storage credentials

One of the most important uses of `fiona.Env` is to set credentials for accessing data stored in AWS S3 or another cloud storage system.

```
import fiona
from fiona.session import AWSSession

with fiona.Env(
    session=AWSSession(
        aws_access_key_id="key",
        aws_secret_access_key="secret"
    )
):
    fiona.open("zip+s3://example-bucket/example.zip")
```

The AWSSession class is currently the only credential session manager in Fiona. The source code has an example of how classes for other cloud storage providers may be implemented. AWSSession relies upon boto3 and botocore, which will be installed as extra dependencies of Fiona if you run `pip install fiona[s3]`.

If you call `fiona.open()` with no surrounding `Env` and pass a path to an S3 object, a session will be created for you using code equivalent to the following code.

```
import boto3

from fiona.session import AWSSession
import fiona

with fiona.Env(session=AWSSession(boto3.Session())):
    fiona.open("zip+s3://fiona-testing/coutwildrnp.zip")
```

3.7.4 1.7.4 Slicing and masking iterators

With some vector data formats a spatial index accompanies the data file, allowing efficient bounding box searches. A collection's *items()* method returns an iterator over pairs of FIDs and records that intersect a given `(minx, miny, maxx, maxy)` bounding box or geometry object. Spatial filtering may be inaccurate and returning all features overlapping the envelope of the geometry. The collection's own coordinate reference system (see below) is used to interpret the box's values. If you want a list of the iterator's items, pass it to Python's builtin `list()` as shown below.

```
>>> colxn = fiona.open(
...     "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip"
... )
>>> hits = list(colxn.items(bbox=(-110.0, 36.0, -108.0, 38.0)))
>>> len(hits)
5
```

The iterator method takes the same `stop` or `start, stop[, step]` slicing arguments as `itertools.islice()`. To get just the first two items from that iterator, pass a stop index.

```
>>> hits = colxn.items(2, bbox=(-110.0, 36.0, -108.0, 38.0)))
>>> len(list(hits))
2
```

To get the third through fifth items from that iterator, pass start and stop indexes.

```
>>> hits = colxn.items(2, 5, bbox=(-110.0, 36.0, -108.0, 38.0)))
>>> len(list(hits))
3
```

To filter features by property values, use Python's builtin `filter()` and `lambda` or your own filter function that takes a single feature record and returns `True` or `False`.

```
>>> def pass_positive_area(rec):
...     return rec['properties'].get('AREA', 0.0) > 0.0
...
>>> colxn = fiona.open(
...     "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip"
... )
>>> hits = filter(pass_positive_area, colxn)
>>> len(list(hits))
67
```

3.7.5 1.7.5 Reading Multilayer data

Up to this point, only simple datasets with one thematic layer or feature type per file have been shown and the venerable Esri Shapefile has been the primary example. Other GIS data formats can encode multiple layers or feature types within a single file or directory. GeoPackage is one example of such a format. A more useful example, for the purpose of this manual, is a directory or zipfile comprising multiple shapefiles. The GitHub-hosted zipfile we've been using in these examples is, in fact, such a multilayer dataset.

The layers of a dataset can be listed using *fiona.listlayers()*. In the shapefile format case, layer names match base names of the files.

```
>>> fiona.listlayers(
...     "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip"
... )
['coutwildrnp']
```

Unlike OGR, Fiona has no classes representing layers or data sources. To access the features of a layer, open a collection using the path to the data source and specify the layer by name using the *layer* keyword.

```
dataset_path = "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip"
>>> for name in fiona.listlayers(dataset_path):
...     with fiona.open(dataset_path, layer=name) as colxn:
...         pprint.pprint(colxn.schema)
...
{'geometry': 'Polygon',
 'properties': {'AGBUR': 'str:80',
                'AREA': 'float:24.15',
                'FEATURE1': 'str:80',
                'FEATURE2': 'str:80',
                'NAME': 'str:80',
                'PERIMETER': 'float:24.15',
                'STATE': 'str:80',
                'STATE_FIPS': 'str:80',
                'URL': 'str:101',
                'WILDRNP020': 'int:10'}}
```

Layers may also be specified by their numerical index.

```
>>> for index, name in enumerate(fiona.listlayers(dataset_path)):
...     with fiona.open(dataset_path, layer=index) as colxn:
...         print(len(colxn))
...
67
```

If no layer is specified, *fiona.open()* returns an open collection using the first layer.

```
>>> with fiona.open(dataset_path) as colxn:
...     colxn.name == fiona.listlayers(datasset_path)[0]
...
True
```

We've been relying on this implicit behavior throughout the manual.

The most general way to open a shapefile for reading, using all of the parameters of *fiona.open()*, is to treat it as a data source with a named layer.

```
>>> fiona.open(
...     "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip",
...     mode="r",
...     layer="coutwildrnp"
... )
```

In practice, it is fine to rely on the implicit first layer and default `'r'` mode and open a shapefile like this:

```
>>> fiona.open(
...     "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip",
... )
```

3.7.6 1.7.6 Writing Multilayer data

To write an entirely new layer to a multilayer data source, simply provide a unique name to the *layer* keyword argument.

```
>>> with fiona.open(
...     "zip+https://github.com/Toblerity/Fiona/files/11151652/coutwildrnp.zip",
... ) as src:
...     with fiona.open("example.gpkg", "w", layer="example one", **src.profile) as dst:
...         dst.writerecords(src)
...
>>> fiona.listlayers("example.gpkg")
['example one']
```

In `'w'` mode, existing layers will be overwritten if specified, just as normal files are overwritten by Python's `open()` function.

3.7.7 1.7.7 Unsupported drivers

Fiona maintains a list of OGR drivers in `fiona.supported_drivers` that are tested and known to work together with Fiona. Opening a dataset using an unsupported driver or access mode results in an :py:attr: *DriverError* exception. By passing *allow_unsupported_drivers=True* to *fiona.open* no compatibility checks are performed and unsupported OGR drivers can be used. However, there are no guarantees that Fiona will be able to access or write data correctly using an unsupported driver.

```
import fiona

with fiona.open("file.kmz", allow_unsupported_drivers=True) as collection:
  ...
```

Not all OGR drivers are necessarily enabled in every GDAL distribution. The following code snippet lists the drivers included in the GDAL installation used by Fiona:

```
from fiona.env import Env

with Env() as gdalenv:
    print(gdalenv.drivers().keys())
```

3.7.8 1.7.8 MemoryFile and ZipMemoryFile

fiona.io.MemoryFile and *fiona.io.ZipMemoryFile* allow formatted feature collections, even zipped feature collections, to be read or written in memory, with no filesystem access required. For example, you may have a zipped shapefile in a stream of bytes coming from a web upload or download.

```
>>> data = open('tests/data/coutwildrnp.zip', 'rb').read()
>>> len(data)
154006
```

(continues on next page)

(continued from previous page)

```
>>> data[:20]
b'PK\x03\x04\x14\x00\x00\x00\x00\x00\xaa~VM\xech\xae\x1e\xec\xab'
```

The feature collection in this stream of bytes can be accessed by wrapping it in an instance of ZipMemoryFile.

```
>>> from fiona.io import ZipMemoryFile
>>> with ZipMemoryFile(data) as zip:
...     with zip.open('coutwildrnp.shp') as collection:
...         print(len(collection))
...         print(collection.schema)
...
67
{'properties': OrderedDict([('PERIMETER', 'float:24.15'), ('FEATURE2', 'str:80'), ('NAME
↪', 'str:80'), ('FEATURE1', 'str:80'), ('URL', 'str:101'), ('AGBUR', 'str:80'), ('AREA',
↪ 'float:24.15'), ('STATE_FIPS', 'str:80'), ('WILDRNP020', 'int:10'), ('STATE', 'str:80
↪')]), 'geometry': 'Polygon'}
```

New in 1.8.0

3.8 1.8 Fiona command line interface

Fiona comes with a command line interface called "fio". See the CLI Documentation for detailed usage instructions.

3.9 1.9 Final Notes

This manual is a work in progress and will grow and improve with Fiona. Questions and suggestions are very welcome. Please feel free to use the issue tracker or email the author directly.

Do see the README for installation instructions and information about supported versions of Python and other software dependencies.

Fiona would not be possible without the contributions of other developers, especially Frank Warmerdam and Even Rouault, the developers of GDAL/OGR; and Mike Weisman, who saved Fiona from neglect and obscurity.

3.10 1.10 References

CHAPTER

FOUR

FIONA

4.1 fiona package

4.1.1 Subpackages

fiona.fio package

Submodules

fiona.fio.bounds module

$ fio bounds

fiona.fio.calc module

fiona.fio.cat module

fio-cat

fiona.fio.collect module

fio-collect

fiona.fio.distrib module

$ fio distrib

fiona.fio.dump module

fio-dump

fiona.fio.env module

$ fio env

fiona.fio.filter module

$ fio filter

fiona.fio.helpers module

Helper objects needed by multiple CLI commands.

fiona.fio.helpers.**eval_feature_expression**(*feature*, *expression*)

fiona.fio.helpers.**id_record**(*rec*)

Converts a record's id to a blank node id and returns the record.

fiona.fio.helpers.**make_ld_context**(*context_items*)

Returns a JSON-LD Context object.

See https://json-ld.org/spec/latest/json-ld/.

fiona.fio.helpers.**nullable**(*val*, *cast*)

fiona.fio.helpers.**obj_gen**(*lines*, *object_hook=None*)

Return a generator of JSON objects loaded from `lines`.

fiona.fio.helpers.**recursive_round**(*obj*, *precision*)

Recursively round coordinates.

fiona.fio.info module

$ fio info

fiona.fio.insp module

$ fio insp

fiona.fio.load module

$ fio load

fiona.fio.ls module

$ fiona ls

fiona.fio.main module

Main click group for the CLI. Needs to be isolated for entry-point loading.

fiona.fio.main.**configure_logging**(*verbosity*)

fiona.fio.options module

Common commandline options for *fio*

fiona.fio.options.**cb_key_val**(*ctx*, *param*, *value*)

click callback to validate *–opt KEY1=VAL1 –opt KEY2=VAL2* and collect in a dictionary like the one below, which is what the CLI function receives. If no value or *None* is received then an empty dictionary is returned.

{
'KEY1': 'VAL1', 'KEY2': 'VAL2'
}

Note: ==*VAL* breaks this as *str.split('=', 1)* is used.

fiona.fio.options.**cb_layer**(*ctx*, *param*, *value*)

Let –layer be a name or index.

fiona.fio.options.**cb_multilayer**(*ctx*, *param*, *value*)

Transform layer options from strings ("1:a,1:b", "2:a,2:c,2:z") to { '1': ['a', 'b'], '2': ['a', 'c', 'z'] }

fiona.fio.options.**validate_multilayer_file_index**(*files*, *layerdict*)

Ensure file indexes provided in the –layer option are valid

fiona.fio.rm module

Module contents

Fiona's command line interface

fiona.fio.**with_context_env**(*f*)

Pops the Fiona Env from the passed context and executes the wrapped func in the context of that obj.

Click's pass_context decorator must precede this decorator, or else there will be no context in the wrapper args.

4.1.2 Submodules

4.1.3 fiona.collection module

Collections provide file-like access to feature data.

class fiona.collection.**BytesCollection**(*bytesbuf*, ***kwds*)

Bases: *Collection*

BytesCollection takes a buffer of bytes and maps that to a virtual file that can then be opened by fiona.

close()

Removes the virtual file associated with the class.

class fiona.collection.**Collection**(*path*, *mode='r'*, *driver=None*, *schema=None*, *crs=None*, *encoding=None*, *layer=None*, *vsi=None*, *archive=None*, *enabled_drivers=None*, *crs_wkt=None*, *ignore_fields=None*, *ignore_geometry=False*, *include_fields=None*, *wkt_version=None*, *allow_unsupported_drivers=False*, ***kwargs*)

Bases: `object`

A file-like interface to features of a vector dataset

Python text file objects are iterators over lines of a file. Fiona Collections are similar iterators (not lists!) over features represented as GeoJSON-like mappings.

property bounds

Returns (minx, miny, maxx, maxy).

close()

In append or write mode, flushes data to disk, then ends access.

property closed

`False` if data can be accessed, otherwise `True`.

property crs

The coordinate reference system (CRS) of the Collection.

property crs_wkt

Returns a WKT string.

property driver

Returns the name of the proper OGR driver.

filter(**args*, ***kwds*)

Returns an iterator over records, but filtered by a test for spatial intersection with the provided `bbox`, a (minx, miny, maxx, maxy) tuple or a geometry `mask`. An attribute filter can be set using an SQL `where` clause, which uses the OGR SQL dialect.

Positional arguments `stop` or `start, stop[, step]` allows iteration to skip over items or stop at a specific item.

Note: spatial filtering using `mask` may be inaccurate and returning all features overlapping the envelope of `mask`.

flush()

Flush the buffer.

get(*item*)

get_tag_item(*key*, *ns=None*)

Returns tag item value

Parameters

- **key** (*str*) – The key for the metadata item to fetch.
- **ns** (*str, optional*) – Used to select a namespace other than the default.

Return type
str

guard_driver_mode()

items(**args*, ***kwds*)

Returns an iterator over FID, record pairs, optionally filtered by a test for spatial intersection with the provided `bbox`, a (minx, miny, maxx, maxy) tuple or a geometry `mask`. An attribute filter can be set using an SQL `where` clause, which uses the OGR SQL dialect.

Positional arguments `stop` or `start, stop[, step]` allows iteration to skip over items or stop at a specific item.

Note: spatial filtering using `mask` may be inaccurate and returning all features overlapping the envelope of `mask`.

keys(**args*, ***kwds*)

Returns an iterator over FIDs, optionally filtered by a test for spatial intersection with the provided `bbox`, a (minx, miny, maxx, maxy) tuple or a geometry `mask`. An attribute filter can be set using an SQL `where` clause, which uses the OGR SQL dialect.

Positional arguments `stop` or `start, stop[, step]` allows iteration to skip over items or stop at a specific item.

Note: spatial filtering using `mask` may be inaccurate and returning all features overlapping the envelope of `mask`.

property meta

Returns a mapping with the driver, schema, crs, and additional properties.

next()

Returns next record from iterator.

property profile

Returns a mapping with the driver, schema, crs, and additional properties.

property schema

Returns a mapping describing the data schema.

The mapping has 'geometry' and 'properties' items. The former is a string such as 'Point' and the latter is an ordered mapping that follows the order of fields in the data file.

tags(*ns=None*)

Returns a dict containing copies of the dataset or layers's tags. Tags are pairs of key and value strings. Tags belong to namespaces. The standard namespaces are: default (None) and 'IMAGE_STRUCTURE'. Applications can create their own additional namespaces.

Parameters
ns (*str, optional*) – Can be used to select a namespace other than the default.

Return type
dict

update_tag_item(*key*, *tag*, *ns=None*)

Updates the tag item value

Parameters

- **key** (*str*) – The key for the metadata item to set.
- **tag** (*str*) – The value of the metadata item to set.
- **ns** (*str, optional*) – Used to select a namespace other than the default.

Return type
int

update_tags(*tags*, *ns=None*)

Writes a dict containing the dataset or layers's tags. Tags are pairs of key and value strings. Tags belong to namespaces. The standard namespaces are: default (None) and 'IMAGE_STRUCTURE'. Applications can create their own additional namespaces.

Parameters

- **tags** (*dict*) – The dict of metadata items to set.
- **ns** (*str, optional*) – Used to select a namespace other than the default.

Return type
int

validate_record(*record*)

Compares the record to the collection's schema.

Returns `True` if the record matches, else `False`.

validate_record_geometry(*record*)

Compares the record's geometry to the collection's schema.

Returns `True` if the record matches, else `False`.

values(**args*, ***kwds*)

Returns an iterator over records, but filtered by a test for spatial intersection with the provided `bbox`, a (minx, miny, maxx, maxy) tuple or a geometry `mask`. An attribute filter can be set using an SQL `where` clause, which uses the OGR SQL dialect.

Positional arguments `stop` or `start, stop[, step]` allows iteration to skip over items or stop at a specific item.

Note: spatial filtering using `mask` may be inaccurate and returning all features overlapping the envelope of `mask`.

write(*record*)

Stages a record for writing to disk.

Note: Each call of this method will start and commit a unique transaction with the data source.

writerecords(*records*)

Stages multiple records for writing to disk.

fiona.collection.**get_filetype**(*bytesbuf*)

Detect compression type of bytesbuf.

ZIP only. TODO: add others relevant to GDAL/OGR.

4.1.4 fiona.compat module

fiona.compat.**strencode**(*instr*, *encoding='utf-8'*)

4.1.5 fiona.crs module

Coordinate reference systems, the CRS class and supporting functions.

A coordinate reference system (CRS) defines how a dataset's pixels map to locations on, for example, a globe or the Earth. A CRS may be local or global. The GIS field shares a number of authority files that define CRS. "EPSG:32618" is the name of a regional CRS from the European Petroleum Survey Group authority file. "OGC:CRS84" is the name of a global CRS from the Open Geospatial Consortium authority. Custom CRS can be described in text using several formats. Rasterio's CRS class is our abstraction for coordinate reference systems.

A fiona.Collection's crs property is an instance of CRS. CRS are also used to define transformations between coordinate reference systems. These transformations are performed by the PROJ library. Rasterio does not call PROJ functions directly, but invokes them via calls to GDAL's "OSR*" functions.

class fiona.crs.**CRS**(*initialdata=None*, ***kwargs*)

Bases: object

A geographic or projected coordinate reference system.

New in version 1.9.0.

CRS objects may be created by passing PROJ parameters as keyword arguments to the standard constructor or by passing EPSG codes, PROJ mappings, PROJ strings, or WKT strings to the from_epsg, from_dict, from_string, or from_wkt static methods.

Examples

The from_dict method takes PROJ parameters as keyword arguments.

```
>>> crs = CRS.from_dict(proj="aea")
```

EPSG codes may be used with the from_epsg method.

```
>>> crs = CRS.from_epsg(3005)
```

The from_string method takes a variety of input.

```
>>> crs = CRS.from_string("EPSG:3005")
```

data

A PROJ4 dict representation of the CRS.

static from_authority(*auth_name*, *code*)

Make a CRS from an authority name and code.

New in version 1.9.0.

Parameters

- **auth_name** (*str*) – The name of the authority.
- **code** (*int or str*) – The code used by the authority.

Return type
CRS

Raises
CRSError –

static from_dict(*initialdata=None*, ***kwargs*)

Make a CRS from a dict of PROJ parameters or PROJ JSON.

Parameters

- **initialdata** (*mapping, optional*) – A dictionary or other mapping
- **kwargs** (*mapping, optional*) – Another mapping. Will be overlaid on the initialdata.

Return type
CRS

Raises
CRSError –

static from_epsg(*code*)

Make a CRS from an EPSG code.

Parameters
code (*int or str*) – An EPSG code. Strings will be converted to integers.

Notes

The input code is not validated against an EPSG database.

Return type
CRS

Raises
CRSError –

static from_proj4(*proj*)

Make a CRS from a PROJ4 string.

Parameters
proj (*str*) – A PROJ4 string like "+proj=longlat ..."

Return type
CRS

Raises
CRSError –

static from_string(*value*, *morph_from_esri_dialect=False*)

Make a CRS from an EPSG, PROJ, or WKT string

Parameters

- **value** (*str*) – An EPSG, PROJ, or WKT string.
- **morph_from_esri_dialect** (*bool, optional*) – If True, items in the input using Esri's dialect of WKT will be replaced by OGC standard equivalents.

Return type
CRS

Raises
CRSError –

static from_user_input(*value*, *morph_from_esri_dialect=False*)

Make a CRS from a variety of inputs.

Parameters

- **value** (*object*) – User input of many different kinds.
- **morph_from_esri_dialect** (*bool, optional*) – If True, items in the input using Esri's dialect of WKT will be replaced by OGC standard equivalents.

Return type
CRS

Raises
CRSError –

static from_wkt(*wkt*, *morph_from_esri_dialect=False*)

Make a CRS from a WKT string.

Parameters

- **wkt** (*str*) – A WKT string.
- **morph_from_esri_dialect** (*bool, optional*) – If True, items in the input using Esri's dialect of WKT will be replaced by OGC standard equivalents.

Return type
CRS

Raises
CRSError –

get(*self*, *item*)

is_epsg_code

Test if the CRS is defined by an EPSG code.

Return type
bool

is_geographic

Test if the CRS is a geographic coordinate reference system.

Return type
bool

Raises
CRSError –

is_projected

Test if the CRS is a projected coordinate reference system.

Return type
bool

Raises
CRSError –

`is_valid`

Test that the CRS is a geographic or projected CRS.

Return type
bool

`items`(*self*)

`keys`(*self*)

`linear_units`

Get a short name for the linear units of the CRS.

Returns
units – "m", "ft", etc.

Return type
str

Raises
CRSError –

`linear_units_factor`

Get linear units and the conversion factor to meters of the CRS.

Returns

- **units** (*str*) – "m", "ft", etc.
- **factor** (*float*) – Ratio of one unit to one meter.

Raises
CRSError –

`to_authority`(*self*, *confidence_threshold=70*)

Convert to the best match authority name and code.

For a CRS created using an EPSG code, that same value is returned. For other CRS, including custom CRS, an attempt is made to match it to definitions in authority files. Matches with a confidence below the threshold are discarded.

Parameters
confidence_threshold (*int*) – Percent match confidence threshold (0-100).

Returns

- **name** (*str*) – Authority name.
- **code** (*str*) – Code from the authority file.
- *or None*

`to_dict`(*self*, *projjson=False*)

Convert CRS to a PROJ dict.

Note: If there is a corresponding EPSG code, it will be used when returning PROJ parameter dict.

New in version 1.9.0.

Parameters
projjson (*bool, default=False*) – If True, will convert to PROJ JSON dict (Requites GDAL 3.1+ and PROJ 6.2+). If False, will convert to PROJ parameter dict.

Return type
dict

`to_epsg`(*self*, *confidence_threshold=70*)

Convert to the best match EPSG code.

For a CRS created using an EPSG code, that same value is returned. For other CRS, including custom CRS, an attempt is made to match it to definitions in the EPSG authority file. Matches with a confidence below the threshold are discarded.

Parameters
confidence_threshold (*int*) – Percent match confidence threshold (0-100).

Return type
int or None

Raises
CRSError –

`to_proj4`(*self*)

Convert to a PROJ4 representation.

Return type
str

`to_string`(*self*)

Convert to a PROJ4 or WKT string.

The output will be reduced as much as possible by attempting a match to CRS defined in authority files.

Notes

Mapping keys are tested against the `all_proj_keys` list. Values of `True` are omitted, leaving the key bare: {'no_defs': True} -> "+no_defs" and items where the value is otherwise not a str, int, or float are omitted.

Return type
str

Raises
CRSError –

`to_wkt`(*self*, *morph_to_esri_dialect=False*, *version=None*)

Convert to a OGC WKT representation.

New in version 1.9.0.

Parameters

- **morph_to_esri_dialect** (*bool, optional*) – Whether or not to morph to the Esri dialect of WKT Only applies to GDAL versions < 3. This parameter will be removed in a future version of fiona (2.0.0).
- **version** (*WktVersion or str, optional*) – The version of the WKT output. Defaults to GDAL's default (WKT1_GDAL for GDAL 3).

Return type
str

Raises
CRSError –

units_factor

Get units and the conversion factor of the CRS.

Returns

- **units** (*str*) – "m", "ft", etc.
- **factor** (*float*) – Ratio of one unit to one radian if the CRS is geographic otherwise, it is to one meter.

Raises

CRSError –

values(*self*)

wkt

An OGC WKT representation of the CRS

Return type

str

fiona.crs.**epsg_treats_as_latlong**(*input_crs*)

Test if the CRS is in latlon order

New in version 1.9.0.

From GDAL docs:

> This method returns TRUE if EPSG feels this geographic coordinate system should be treated as having lat/long coordinate ordering.

> Currently this returns TRUE for all geographic coordinate systems with an EPSG code set, and axes set defining it as lat, long.

> FALSE will be returned for all coordinate systems that are not geographic, or that do not have an EPSG code set.

> **Note**

> Important change of behavior since GDAL 3.0. In previous versions, geographic CRS imported with importFromEPSG() would cause this method to return FALSE on them, whereas now it returns TRUE, since importFromEPSG() is now equivalent to importFromEPSGA().

Parameters

input_crs (CRS) – Coordinate reference system, as a fiona CRS object Example: CRS({'init': 'EPSG:4326'})

Return type

bool

fiona.crs.**epsg_treats_as_northingeasting**(*input_crs*)

Test if the CRS should be treated as having northing/easting coordinate ordering

New in version 1.9.0.

From GDAL docs:

> This method returns TRUE if EPSG feels this projected coordinate system should be treated as having northing/easting coordinate ordering.

> Currently this returns TRUE for all projected coordinate systems with an EPSG code set, and axes set defining it as northing, easting.

> FALSE will be returned for all coordinate systems that are not projected, or that do not have an EPSG code set.

> **Note**

> Important change of behavior since GDAL 3.0. In previous versions, projected CRS with northing, easting axis order imported with importFromEPSG() would cause this method to return FALSE on them, whereas now it returns TRUE, since importFromEPSG() is now equivalent to importFromEPSGA().

Parameters
input_crs (CRS) – Coordinate reference system, as a fiona CRS object Example: CRS({'init': 'EPSG:4326'})

Return type
bool

fiona.crs.**from_epsg**(*val*)

Given an integer code, returns an EPSG-like mapping.

Deprecated since version 1.9.0: This function will be removed in version 2.0. Please use CRS.from_epsg() instead.

fiona.crs.**from_string**(*val*)

Turn a PROJ.4 string into a mapping of parameters.

Deprecated since version 1.9.0: This function will be removed in version 2.0. Please use CRS.from_string() instead.

fiona.crs.**to_string**(*val*)

Turn a parameter mapping into a more conventional PROJ.4 string.

Deprecated since version 1.9.0: This function will be removed in version 2.0. Please use CRS.to_string() instead.

4.1.6 fiona.drvsupport module

fiona.drvsupport.**driver_from_extension**(*path*)

Attempt to auto-detect driver based on the extension.

Parameters
path (*str or pathlike object*) – The path to the dataset to write with.

Returns
The name of the driver for the extension.

Return type
str

fiona.drvsupport.**vector_driver_extensions**()

Returns
Map of extensions to the driver.

Return type
dict

4.1.7 fiona.env module

Fiona's GDAL/AWS environment

class fiona.env.**Env**(*session=None*, *aws_unsigned=False*, *profile_name=None*, *session_class=<function Session.aws_or_dummy>*, ***options*)

Bases: object

Abstraction for GDAL and AWS configuration

The GDAL library is stateful: it has a registry of format drivers, an error stack, and dozens of configuration options.

Fiona's approach to working with GDAL is to wrap all the state up using a Python context manager (see PEP 343, https://www.python.org/dev/peps/pep-0343/). When the context is entered GDAL drivers are registered, error handlers are configured, and configuration options are set. When the context is exited, drivers are removed from the registry and other configurations are removed.

Example

with fiona.Env(GDAL_CACHEMAX=512) as env:
All drivers are registered, GDAL's raster block cache # size is set to 512MB. # Commence processing... ... # End of processing.

At this point, configuration options are set to their # previous (possible unset) values.

A boto3 session or boto3 session constructor arguments *aws_access_key_id*, *aws_secret_access_key*, *aws_session_token* may be passed to Env's constructor. In the latter case, a session will be created as soon as needed. AWS credentials are configured for GDAL as needed.

credentialize()

Get credentials and configure GDAL

Note well: this method is a no-op if the GDAL environment already has credentials, unless session is not None.

Return type
None

classmethod default_options()

Default configuration options

Parameters
None –

Return type
dict

drivers()

Return a mapping of registered drivers.

classmethod from_defaults(**args*, ***kwargs*)

Create an environment with default config options

Parameters

- **args** (*optional*) – Positional arguments for Env()
- **kwargs** (*optional*) – Keyword arguments for Env()

Return type
Env

Notes

The items in kwargs will be overlaid on the default values.

class fiona.env.**GDALVersion**(*major=0*, *minor=0*)

Bases: object

Convenience class for obtaining GDAL major and minor version components and comparing between versions. This is highly simplistic and assumes a very normal numbering scheme for versions and ignores everything except the major and minor components.

at_least(*other*)

major

minor

classmethod parse(*input*)

Parses input tuple or string to GDALVersion. If input is a GDALVersion instance, it is returned.

Parameters
input (*tuple of (major, minor), string, or instance of* GDALVersion) –

Return type
GDALVersion instance

classmethod runtime()

Return GDALVersion of current GDAL runtime

class fiona.env.**NullContextManager**

Bases: object

class fiona.env.**ThreadEnv**

Bases: _local

fiona.env.**defenv**(***options*)

Create a default environment if necessary.

fiona.env.**delenv**()

Delete options in the existing environment.

fiona.env.**ensure_env**(*f*)

A decorator that ensures an env exists before a function calls any GDAL C functions.

Parameters
f (*function*) – A function.

Return type
A function wrapper.

Notes

If there is already an existing environment, the wrapper does nothing and immediately calls f with the given arguments.

fiona.env.**ensure_env_with_credentials**(*f*)

Ensures a config environment exists and has credentials.

Parameters
f (*function*) – A function.

Return type
A function wrapper.

Notes

The function wrapper checks the first argument of f and credentializes the environment if the first argument is a URI with scheme "s3".

If there is already an existing environment, the wrapper does nothing and immediately calls f with the given arguments.

fiona.env.**env_ctx_if_needed**()

Return an Env if one does not exist

Return type
Env or a do-nothing context manager

fiona.env.**getenv**()

Get a mapping of current options.

fiona.env.**hascreds**()

fiona.env.**hasenv**()

fiona.env.**require_gdal_version**(*version*, *param=None*, *values=None*, *is_max_version=False*, *reason=''*)

A decorator that ensures the called function or parameters are supported by the runtime version of GDAL. Raises GDALVersionError if conditions are not met.

Examples:

@require_gdal_version('2.2') def some_func():

calling *some_func* with a runtime version of GDAL that is < 2.2 raises a GDALVersionErorr.

@require_gdal_version('2.2', param='foo') def some_func(foo='bar'):

calling *some_func* with parameter *foo* of any value on GDAL < 2.2 raises a GDALVersionError.

@require_gdal_version('2.2', param='foo', values=('bar',)) def some_func(foo=None):

calling *some_func* with parameter *foo* and value *bar* on GDAL < 2.2 raises a GDALVersionError.

Parameters

- **version** (*tuple, string, or* GDALVersion) –
- **param** (*string (optional, default: None)*) – If *values* are absent, then all use of this parameter with a value other than default value requires at least GDAL *version*.

- **values** (*tuple, list, or set (optional, default: None)*) – contains values that require at least GDAL *version*. *param* is required for *values*.
- **is_max_version** (*bool (optional, default: False)*) – if *True* indicates that the version provided is the maximum version allowed, instead of requiring at least that version.
- **reason** (*string (optional: default: "")*) – custom error message presented to user in addition to message about GDAL version. Use this to provide an explanation of what changed if necessary context to the user.

Return type
wrapped function

fiona.env.**setenv**(***options*)

Set options in the existing environment.

4.1.8 fiona.errors module

exception fiona.errors.**AttributeFilterError**

Bases: *FionaValueError*

Error processing SQL WHERE clause with the dataset.

exception fiona.errors.**CRSError**

Bases: *FionaValueError*

When a crs mapping has neither init or proj items.

exception fiona.errors.**DataIOError**

Bases: OSError

IO errors involving driver registration or availability.

exception fiona.errors.**DatasetDeleteError**

Bases: OSError

Failure to delete a dataset

exception fiona.errors.**DriverError**

Bases: *FionaValueError*

Encapsulates unsupported driver and driver mode errors.

exception fiona.errors.**DriverIOError**

Bases: OSError

A format specific driver error.

exception fiona.errors.**DriverSupportError**

Bases: *DriverIOError*

Driver does not support schema

exception fiona.errors.**EnvError**

Bases: *FionaError*

Environment Errors

exception fiona.errors.**FeatureWarning**

Bases: UserWarning

A warning about serialization of a feature

exception fiona.errors.**FieldNameEncodeError**

Bases: UnicodeEncodeError

Failure to encode a field name.

exception fiona.errors.**FionaDeprecationWarning**

Bases: DeprecationWarning

A warning about deprecation of Fiona features

exception fiona.errors.**FionaError**

Bases: Exception

Base Fiona error

exception fiona.errors.**FionaValueError**

Bases: *FionaError*, ValueError

Fiona-specific value errors

exception fiona.errors.**GDALVersionError**

Bases: *FionaError*

Raised if the runtime version of GDAL does not meet the required version of GDAL.

exception fiona.errors.**GeometryTypeValidationError**

Bases: *FionaValueError*

Tried to write a geometry type not specified in the schema

exception fiona.errors.**SchemaError**

Bases: *FionaValueError*

When a schema mapping has no properties or no geometry.

exception fiona.errors.**TransactionError**

Bases: RuntimeError

Failure relating to GDAL transactions

exception fiona.errors.**TransformError**

Bases: *FionaError*

Raised if a coordinate transformation fails.

exception fiona.errors.**UnsupportedGeometryTypeError**

Bases: KeyError

When a OGR geometry type isn't supported by Fiona.

exception fiona.errors.**UnsupportedOperation**

Bases: *FionaError*

Raised when reading from a file opened in 'w' mode

4.1.9 fiona.inspector module

fiona.inspector.**main**(*srcfile*)

Open a dataset in an iteractive session.

4.1.10 fiona.io module

Classes capable of reading and writing collections

class fiona.io.**MemoryFile**(*file_or_bytes=None*, *filename=None*, *ext=''*)

Bases: *MemoryFileBase*

A BytesIO-like object, backed by an in-memory file.

This allows formatted files to be read and written without I/O.

A MemoryFile created with initial bytes becomes immutable. A MemoryFile created without initial bytes may be written to using either file-like or dataset interfaces.

Parameters

- **file_or_bytes** (*an open Python file, bytes, or None*) – If not None, the MemoryFile becomes immutable and read-only. If None, it is write-only.
- **filename** (*str*) – An optional filename. The default is a UUID-based name.
- **ext** (*str*) – An optional file extension. Some format drivers require a specific value.

listdir(*path=None*)

List files in a directory.

Parameters
path (*URI (str or pathlib.Path)*) – A dataset resource identifier.

Returns
A list of filename strings.

Return type
list

listlayers(*path=None*)

List layer names in their index order

Parameters
path (*URI (str or pathlib.Path)*) – A dataset resource identifier.

Returns
A list of layer name strings.

Return type
list

open(*mode=None*, *driver=None*, *schema=None*, *crs=None*, *encoding=None*, *layer=None*, *vfs=None*, *enabled_drivers=None*, *crs_wkt=None*, *allow_unsupported_drivers=False*, ***kwargs*)

Open the file and return a Fiona collection object.

If data has already been written, the file is opened in 'r' mode. Otherwise, the file is opened in 'w' mode.

Parameters

- **parameter** (*Note well that there is no path*) –

- **a** (*contains a single dataset and there is no need to specify*) –
- **path.** –
- **the** (*Other parameters are optional and have the same semantics as*) –
- **fiona.open().** (*parameters of*) –

class fiona.io.**ZipMemoryFile**(*file_or_bytes=None*, *filename=None*, *ext='.zip'*)

Bases: *MemoryFile*

A read-only BytesIO-like object backed by an in-memory zip file.

This allows a zip file containing formatted files to be read without I/O.

Parameters

- **file_or_bytes** (*an open Python file, bytes, or None*) – If not None, the MemoryFile becomes immutable and read-only. If None, it is write-only.
- **filename** (*str*) – An optional filename. The default is a UUID-based name.
- **ext** (*str*) – An optional file extension. Some format drivers require a specific value. The default is ".zip".

open(*path=None*, *driver=None*, *encoding=None*, *layer=None*, *enabled_drivers=None*, *allow_unsupported_drivers=False*, ***kwargs*)

Open a dataset within the zipped stream.

Parameters
path (*str*) – Path to a dataset in the zip file, relative to the root of the archive.

Return type
A Fiona collection object

4.1.11 fiona.logutils module

Logging helper classes.

class fiona.logutils.**FieldSkipLogFilter**(*name=''*)

Bases: `Filter`

Filter field skip log messges.

At most, one message per field skipped per loop will be passed.

filter(*record*)

Pass record if not seen.

class fiona.logutils.**LogFiltering**(*logger*, *filter*)

Bases: `object`

4.1.12 fiona.ogrext module

class fiona.ogrext.**FeatureBuilder**

Bases: object

Build Fiona features from OGR feature pointers.

No OGR objects are allocated by this function and the feature argument is not destroyed.

class fiona.ogrext.**ItemsIterator**

Bases: *Iterator*

class fiona.ogrext.**Iterator**

Bases: object

Provides iterated access to feature data.

class fiona.ogrext.**KeysIterator**

Bases: *Iterator*

class fiona.ogrext.**MemoryFileBase**

Bases: object

Base for a BytesIO-like class backed by an in-memory file.

close()

Close and tear down VSI file and directory.

exists()

Test if the in-memory file exists.

Returns
True if the in-memory file exists.

Return type
bool

getbuffer()

Return a view on bytes of the file, or None.

read()

Read size bytes from MemoryFile.

seek()

tell()

write()

Write data bytes to MemoryFile

class fiona.ogrext.**OGRFeatureBuilder**

Bases: object

Builds an OGR Feature from a Fiona feature mapping.

Allocates one OGR Feature which should be destroyed by the caller. Borrows a layer definition from the collection.

class fiona.ogrext.**Session**

Bases: object

get()

Provides access to feature data by FID.

Supports Collection.__contains__().

get_crs()

Get the layer's CRS

Return type
CRS

get_crs_wkt()

get_driver()

get_extent()

get_feature()

Provides access to feature data by FID.

Supports Collection.__contains__().

get_fileencoding()

DEPRECATED

get_length()

get_schema()

Get a dictionary representation of a collection's schema.

The schema dict contains "geometry" and "properties" items.

Return type
dict

> **Warning:** Fiona 1.9 does not support multiple fields with the name name. When encountered, a warning message is logged and the field is skipped.

get_tag_item()

Returns tag item value

Parameters

- **key** (*str*) – The key for the metadata item to fetch.
- **ns** (*str, optional*) – Used to select a namespace other than the default.

Return type
str

has_feature()

Provides access to feature data by FID.

Supports Collection.__contains__().

isactive()

start()

stop()

tags()

Returns a dict containing copies of the dataset or layers's tags. Tags are pairs of key and value strings. Tags belong to namespaces. The standard namespaces are: default (None) and 'IMAGE_STRUCTURE'. Applications can create their own additional namespaces.

Parameters
ns (*str, optional*) – Can be used to select a namespace other than the default.

Return type
dict

class fiona.ogrext.TZ(*minutes*)

Bases: tzinfo

utcoffset(*dt*)

datetime -> timedelta showing offset from UTC, negative values indicating West of UTC

class fiona.ogrext.WritingSession

Bases: *Session*

start()

sync()

Syncs OGR to disk.

update_tag_item()

Updates the tag item value

Parameters

- **key** (*str*) – The key for the metadata item to set.
- **tag** (*str*) – The value of the metadata item to set.
- **ns** (*str*) – Used to select a namespace other than the default.

Return type
int

update_tags()

Writes a dict containing the dataset or layers's tags. Tags are pairs of key and value strings. Tags belong to namespaces. The standard namespaces are: default (None) and 'IMAGE_STRUCTURE'. Applications can create their own additional namespaces.

Parameters

- **tags** (*dict*) – The dict of metadata items to set.
- **ns** (*str, optional*) – Used to select a namespace other than the default.

Return type
int

writerecs()

Writes records to collection storage.

Parameters

- **records** (*Iterable*) – A stream of feature records.
- **collection** (Collection) – The collection in which feature records are stored.

Return type
None

fiona.ogrext.**buffer_to_virtual_file**()

Maps a bytes buffer to a virtual file.

ext is empty or begins with a period and contains at most one period.

fiona.ogrext.**featureRT**()

fiona.ogrext.**remove_virtual_file**()

4.1.13 fiona.path module

Dataset paths, identifiers, and filenames

class fiona.path.**ParsedPath**(*path*, *archive*, *scheme*)

Bases: *Path*

Result of parsing a dataset URI/Path

path

Parsed path. Includes the hostname and query string in the case of a URI.

Type
str

archive

Parsed archive path.

Type
str

scheme

URI scheme such as "https" or "zip+s3".

Type
str

archive

classmethod from_uri(*uri*)

property is_local

Test if the path is a local URI

property is_remote

Test if the path is a remote, network URI

property name

The parsed path's original URI

path

scheme

class fiona.path.**Path**

Bases: object

Base class for dataset paths

class fiona.path.**UnparsedPath**(*path*)

Bases: *Path*

Encapsulates legacy GDAL filenames

path

The legacy GDAL filename.

Type
str

property name

The unparsed path's original path

path

fiona.path.**parse_path**(*path*)

Parse a dataset's identifier or path into its parts

Parameters
path (*str or path-like object*) – The path to be parsed.

Return type
ParsedPath or *UnparsedPath*

Notes

When legacy GDAL filenames are encountered, they will be returned in a UnparsedPath.

fiona.path.**vsi_path**(*path*)

Convert a parsed path to a GDAL VSI path

Parameters
path (Path) – A ParsedPath or UnparsedPath object.

Return type
str

4.1.14 fiona.rfc3339 module

class fiona.rfc3339.**FionaDateTimeType**

Bases: str

Dates and times.

class fiona.rfc3339.**FionaDateType**

Bases: str

Dates without time.

class fiona.rfc3339.**FionaTimeType**

Bases: str

Times without dates.

class fiona.rfc3339.**group_accessor**(*m*)

Bases: object

group(*i*)

fiona.rfc3339.**parse_date**(*text*)

Given a date, returns a datetime tuple

Parameters
text (*string to be parsed*) –

Returns
datetime tuple: (year, month, day, hour, minute, second, microsecond, utcoffset in minutes or None)

Return type
(int, int , int, int, int, int, int, int)

fiona.rfc3339.**parse_datetime**(*text*)

Given a datetime, returns a datetime tuple

Parameters
text (*string to be parsed*) –

Returns
datetime tuple: (year, month, day, hour, minute, second, microsecond, utcoffset in minutes or None)

Return type
(int, int , int, int, int, int, int, int)

fiona.rfc3339.**parse_time**(*text*)

Given a time, returns a datetime tuple

Parameters
text (*string to be parsed*) –

Returns
datetime tuple: (year, month, day, hour, minute, second, microsecond, utcoffset in minutes or None)

Return type
(int, int , int, int, int, int, int, int)

4.1.15 fiona.schema module

Fiona schema module.

fiona.schema.**normalize_field_type**()

Normalize free form field types to an element of FIELD_TYPES

Parameters
ftype (*str*) – A type:width format like 'int:9' or 'str:255'

Returns
An element from FIELD_TYPES

Return type
str

4.1.16 fiona.session module

Abstraction for sessions in various clouds.

class fiona.session.**AWSSession**(*session=None*, *aws_unsigned=False*, *aws_access_key_id=None*, *aws_secret_access_key=None*, *aws_session_token=None*, *region_name=None*, *profile_name=None*, *endpoint_url=None*, *requester_pays=False*)

Bases: *Session*

Configures access to secured resources stored in AWS S3.

property credentials

The session credentials as a dict

get_credential_options()

Get credentials as GDAL configuration options

Return type
dict

classmethod hascreds(*config*)

Determine if the given configuration has proper credentials

Parameters

- **cls** (*class*) – A Session class.
- **config** (*dict*) – GDAL configuration as a dict.

Return type
bool

class fiona.session.**AzureSession**(*azure_storage_connection_string=None*, *azure_storage_account=None*, *azure_storage_access_key=None*, *azure_unsigned=False*)

Bases: *Session*

Configures access to secured resources stored in Microsoft Azure Blob Storage.

property credentials

The session credentials as a dict

get_credential_options()

Get credentials as GDAL configuration options

Return type
dict

classmethod hascreds(*config*)

Determine if the given configuration has proper credentials

Parameters

- **cls** (*class*) – A Session class.
- **config** (*dict*) – GDAL configuration as a dict.

Return type
bool

class fiona.session.**DummySession**(**args*, ***kwargs*)

Bases: *Session*

A dummy session.

credentials

The session credentials.

Type
dict

get_credential_options()

Get credentials as GDAL configuration options

Return type
dict

classmethod **hascreds**(*config*)

Determine if the given configuration has proper credentials

Parameters

- **cls** (*class*) – A Session class.
- **config** (*dict*) – GDAL configuration as a dict.

Return type
bool

class fiona.session.**GSSession**(*google_application_credentials=None*)

Bases: *Session*

Configures access to secured resources stored in Google Cloud Storage

property **credentials**

The session credentials as a dict

get_credential_options()

Get credentials as GDAL configuration options

Return type
dict

classmethod **hascreds**(*config*)

Determine if the given configuration has proper credentials

Parameters

- **cls** (*class*) – A Session class.
- **config** (*dict*) – GDAL configuration as a dict.

Return type
bool

class fiona.session.**OSSSession**(*oss_access_key_id=None*, *oss_secret_access_key=None*, *oss_endpoint=None*)

Bases: *Session*

Configures access to secured resources stored in Alibaba Cloud OSS.

property **credentials**

The session credentials as a dict

get_credential_options()

Get credentials as GDAL configuration options

Return type
dict

classmethod hascreds(*config*)

Determine if the given configuration has proper credentials

Parameters

- **cls** (*class*) – A Session class.
- **config** (*dict*) – GDAL configuration as a dict.

Return type
bool

class fiona.session.**Session**

Bases: object

Base for classes that configure access to secured resources.

credentials

Keys and values for session credentials.

Type
dict

Notes

This class is not intended to be instantiated.

static aws_or_dummy(**args*, ***kwargs*)

Create an AWSSession if boto3 is available, else DummySession :param path: A dataset path or identifier. :type path: str :param args: Positional arguments for the foreign session constructor. :type args: sequence :param kwargs: Keyword arguments for the foreign session constructor. :type kwargs: dict

Return type
Session

static cls_from_path(*path*)

Find the session class suited to the data at *path*.

Parameters
path (*str*) – A dataset path or identifier.

Return type
class

static from_environ(**args*, ***kwargs*)

Create a session object suited to the environment. :param path: A dataset path or identifier. :type path: str :param args: Positional arguments for the foreign session constructor. :type args: sequence :param kwargs: Keyword arguments for the foreign session constructor. :type kwargs: dict

Return type
Session

static from_foreign_session(*session*, *cls=None*)

Create a session object matching the foreign *session*.

Parameters

- **session** (*obj*) – A foreign session object.
- **cls** (*Session class, optional*) – The class to return.

Return type
Session

static from_path(*path*, **args*, ***kwargs*)

Create a session object suited to the data at *path*.

Parameters

- **path** (*str*) – A dataset path or identifier.
- **args** (*sequence*) – Positional arguments for the foreign session constructor.
- **kwargs** (*dict*) – Keyword arguments for the foreign session constructor.

Return type
Session

get_credential_options()

Get credentials as GDAL configuration options

Return type
dict

classmethod hascreds(*config*)

Determine if the given configuration has proper credentials

Parameters

- **cls** (*class*) – A Session class.
- **config** (*dict*) – GDAL configuration as a dict.

Return type
bool

class fiona.session.**SwiftSession**(*session=None*, *swift_storage_url=None*, *swift_auth_token=None*, *swift_auth_v1_url=None*, *swift_user=None*, *swift_key=None*)

Bases: *Session*

Configures access to secured resources stored in OpenStack Swift Object Storage.

property credentials

The session credentials as a dict

get_credential_options()

Get credentials as GDAL configuration options

Return type
dict

classmethod hascreds(*config*)

Determine if the given configuration has proper credentials

Parameters

- **cls** (*class*) – A Session class.
- **config** (*dict*) – GDAL configuration as a dict.

Return type
bool

4.1.17 fiona.transform module

Coordinate and geometry warping and reprojection

fiona.transform.**transform**(*src_crs*, *dst_crs*, *xs*, *ys*)

Transform coordinates from one reference system to another.

Parameters

- **src_crs** (*str or dict*) – A string like 'EPSG:4326' or a dict of proj4 parameters like {'proj': 'lcc', 'lat_0': 18.0, 'lat_1': 18.0, 'lon_0': -77.0} representing the coordinate reference system on the "source" or "from" side of the transformation.
- **dst_crs** (*str or dict*) – A string or dict representing the coordinate reference system on the "destination" or "to" side of the transformation.
- **xs** (*sequence of float*) – A list or tuple of x coordinate values. Must have the same length as the `ys` parameter.
- **ys** (*sequence of float*) – A list or tuple of y coordinate values. Must have the same length as the `xs` parameter.

Returns
xp, yp – A pair of transformed coordinate sequences. The elements of `xp` and `yp` correspond exactly to the elements of the `xs` and `ys` input parameters.

Return type
list of float

Examples

```
>>> transform('EPSG:4326', 'EPSG:26953', [-105.0], [40.0])
([957097.0952383667], [378940.8419189212])
```

fiona.transform.**transform_geom**(*src_crs*, *dst_crs*, *geom*, *antimeridian_cutting=False*, *antimeridian_offset=10.0*, *precision=-1*)

Transform a geometry obj from one reference system to another.

Parameters

- **src_crs** (*str or dict*) – A string like 'EPSG:4326' or a dict of proj4 parameters like {'proj': 'lcc', 'lat_0': 18.0, 'lat_1': 18.0, 'lon_0': -77.0} representing the coordinate reference system on the "source" or "from" side of the transformation.
- **dst_crs** (*str or dict*) – A string or dict representing the coordinate reference system on the "destination" or "to" side of the transformation.
- **geom** (*obj*) – A GeoJSON-like geometry object with 'type' and 'coordinates' members or an iterable of GeoJSON-like geometry objects.
- **antimeridian_cutting** (*bool, optional*) – `True` to cut output geometries in two at the antimeridian, the default is ``False`.

- **antimeridian_offset** (*float, optional*) – A distance in decimal degrees from the antimeridian, outside of which geometries will not be cut.
- **precision** (*int, optional*) – Round geometry coordinates to this number of decimal places. This parameter is deprecated and will be removed in 2.0.

Returns

A new GeoJSON-like geometry (or a list of GeoJSON-like geometries if an iterable was given as input) with transformed coordinates. Note that if the output is at the antimeridian, it may be cut and of a different geometry `type` than the input, e.g., a polygon input may result in multi-polygon output.

Return type

obj

Examples

```
>>> transform_geom(
...     'EPSG:4326', 'EPSG:26953',
...     {'type': 'Point', 'coordinates': [-105.0, 40.0]})
{'type': 'Point', 'coordinates': (957097.0952383667, 378940.8419189212)}
```

4.1.18 fiona.vfs module

Implementation of Apache VFS schemes and URLs.

`fiona.vfs.`**is_remote**(*scheme*)

`fiona.vfs.`**parse_paths**(*uri*, *vfs=None*)

Parse a URI or Apache VFS URL into its parts

Returns: tuple

(path, scheme, archive)

`fiona.vfs.`**valid_vsi**(*vsi*)

Ensures all parts of our vsi path are valid schemes.

`fiona.vfs.`**vsi_path**(*path*, *vsi=None*, *archive=None*)

4.1.19 fiona module

Fiona is OGR's neat, nimble API.

Fiona provides a minimal, uncomplicated Python interface to the open source GIS community's most trusted geodata access library and integrates readily with other Python GIS packages such as pyproj, Rtree and Shapely.

A Fiona feature is a Python mapping inspired by the GeoJSON format. It has `id`, `geometry`, and `properties` attributes. The value of `id` is a string identifier unique within the feature's parent collection. The `geometry` is another mapping with `type` and `coordinates` keys. The `properties` of a feature is another mapping corresponding to its attribute table.

Features are read and written using the `Collection` class. These `Collection` objects are a lot like Python `file` objects. A `Collection` opened in reading mode serves as an iterator over features. One opened in a writing mode provides a `write` method.

class fiona.**Feature**(*geometry=None*, *id=None*, *properties=None*, ***data*)

Bases: `Object`

A GeoJSON-like feature

Notes

Delegates geometry and properties to an instance of _Feature, which will become an extension class in Fiona 2.0.

classmethod from_dict(*ob=None*, ***kwargs*)

property geometry

The feature's geometry object

Return type
Geometry

property id

The feature's id

Return type
object

property properties

The feature's properties

Return type
object

property type

The Feature's type

Return type
str

class fiona.**Geometry**(*coordinates=None*, *type=None*, *geometries=None*, ***data*)

Bases: `Object`

A GeoJSON-like geometry

Notes

Delegates coordinates and type properties to an instance of _Geometry, which will become an extension class in Fiona 2.0.

property coordinates

The geometry's coordinates

Return type
Sequence

classmethod from_dict(*ob=None*, ***kwargs*)

property geometries

A collection's geometries.

Return type
list

property type

The geometry's type

Return type
str

class fiona.**Properties**(***kwds*)

Bases: Object

A GeoJSON-like feature's properties

classmethod **from_dict**(*mapping=None*, ***kwargs*)

fiona.**bounds**(*ob*)

Returns a (minx, miny, maxx, maxy) bounding box.

The ob may be a feature record or geometry.

fiona.**listdir**(*fp*)

Lists the datasets in a directory or archive file.

Archive files must be prefixed like "zip://" or "tar://".

Parameters
fp (*str or pathlib.Path*) – Directory or archive path.

Returns
A list of datasets.

Return type
list of str

Raises
TypeError – If the input is not a str or Path.

fiona.**listlayers**(*fp*, *vfs=None*, ***kwargs*)

Lists the layers (collections) in a dataset.

Archive files must be prefixed like "zip://" or "tar://".

Parameters

- **fp** (*str, pathlib.Path, or file-like object*) – A dataset identifier or file object containing a dataset.
- **vfs** (*str*) – This is a deprecated parameter. A URI scheme such as "zip://" should be used instead.
- **kwargs** (*dict*) – Dataset opening options and other keyword args.

Returns
A list of layer name strings.

Return type
list of str

Raises
TypeError – If the input is not a str, Path, or file object.

fiona.**open**(*fp*, *mode='r'*, *driver=None*, *schema=None*, *crs=None*, *encoding=None*, *layer=None*, *vfs=None*, *enabled_drivers=None*, *crs_wkt=None*, *allow_unsupported_drivers=False*, ***kwargs*)

Open a collection for read, append, or write

In write mode, a driver name such as "ESRI Shapefile" or "GPX" (see OGR docs or `ogr2ogr --help` on the command line) and a schema mapping such as:

{'geometry': 'Point',

'properties': [('class', 'int'), ('label', 'str'),
('value', 'float')]}

must be provided. If a particular ordering of properties ("fields" in GIS parlance) in the written file is desired, a list of (key, value) pairs as above or an ordered dict is required. If no ordering is needed, a standard dict will suffice.

A coordinate reference system for collections in write mode can be defined by the `crs` parameter. It takes Proj4 style mappings like

{'proj': 'longlat', 'ellps': 'WGS84', 'datum': 'WGS84',
'no_defs': True}

short hand strings like

EPSG:4326

or WKT representations of coordinate reference systems.

The drivers used by Fiona will try to detect the encoding of data files. If they fail, you may provide the proper `encoding`, such as 'Windows-1252' for the Natural Earth datasets.

When the provided path is to a file containing multiple named layers of data, a layer can be singled out by `layer`.

The drivers enabled for opening datasets may be restricted to those listed in the `enabled_drivers` parameter. This and the `driver` parameter afford much control over opening of files.

Trying only the GeoJSON driver when opening to read, the # following raises `DataIOError`: fiona.open('example.shp', driver='GeoJSON')

Trying first the GeoJSON driver, then the Shapefile driver, # the following succeeds: fiona.open(

'example.shp', enabled_drivers=['GeoJSON', 'ESRI Shapefile'])

Parameters

- **fp** (*URI (str or pathlib.Path), or file-like object*) – A dataset resource identifier or file object.
- **mode** (*str*) – One of 'r', to read (the default); 'a', to append; or 'w', to write.
- **driver** (*str*) – In 'w' mode a format driver name is required. In 'r' or 'a' mode this parameter has no effect.
- **schema** (*dict*) – Required in 'w' mode, has no effect in 'r' or 'a' mode.
- **crs** (*str or dict*) – Required in 'w' mode, has no effect in 'r' or 'a' mode.
- **encoding** (*str*) – Name of the encoding used to encode or decode the dataset.
- **layer** (*int or str*) – The integer index or name of a layer in a multi-layer dataset.
- **vfs** (*str*) – This is a deprecated parameter. A URI scheme such as "zip://" should be used instead.
- **enabled_drivers** (*list*) – An optional list of driver names to used when opening a collection.
- **crs_wkt** (*str*) – An optional WKT representation of a coordinate reference system.
- **ignore_fields** (*list*) – List of field names to ignore on load.

- **ignore_geometry** (*bool*) – Ignore the geometry on load.
- **include_fields** (*list*) – List of a subset of field names to include on load.
- **wkt_version** (*fiona.enums.WktVersion or str, optional*) – Version to use to for the CRS WKT. Defaults to GDAL's default (WKT1_GDAL for GDAL 3).
- **allow_unsupported_drivers** (*bool*) – If set to true do not limit GDAL drivers to set set of known working.
- **kwargs** (*mapping*) – Other driver-specific parameters that will be interpreted by the OGR library as layer creation or opening options.

Return type
Collection

fiona.**prop_type**(*text*)

Returns a schema property's proper Python type.

Parameters
text (*str*) – A type name, with or without width.

Returns
A Python class.

Return type
obj

Examples

```
>>> prop_type('int')
<class 'int'>
>>> prop_type('str:25')
<class 'str'>
```

fiona.**prop_width**(*val*)

Returns the width of a str type property.

Undefined for non-str properties.

Parameters
val (*str*) – A type:width string from a collection schema.

Return type
int or None

Examples

```
>>> prop_width('str:25')
25
>>> prop_width('str')
80
```

fiona.**remove**(*path_or_collection*, *driver=None*, *layer=None*)

Delete an OGR data source or one of its layers.

If no layer is specified, the entire dataset and all of its layers and associated sidecar files will be deleted.

Parameters

- **path_or_collection** (*str, pathlib.Path, or* Collection) – The target Collection or its path.
- **driver** (*str, optional*) – The name of a driver to be used for deletion, optional. Can usually be detected.
- **layer** (*str or int, optional*) – The name or index of a specific layer.

Return type
None

Raises
DatasetDeleteError – If the data source cannot be deleted.

CHAPTER

FIVE

COMMAND LINE INTERFACE

Fiona's new command line interface is a program named "fio".

```
Usage: fio [OPTIONS] COMMAND [ARGS]...

  Fiona command line interface.

Options:
  -v, --verbose     Increase verbosity.
  -q, --quiet       Decrease verbosity.
  --version         Show the version and exit.
  --gdal-version    Show the version and exit.
  --python-version  Show the version and exit.
  --help            Show this message and exit.

Commands:
  bounds   Print the extent of GeoJSON objects
  calc     Calculate GeoJSON property by Python expression
  cat      Concatenate and print the features of datasets
  collect  Collect a sequence of features.
  distrib  Distribute features from a collection.
  dump     Dump a dataset to GeoJSON.
  env      Print information about the fio environment.
  filter   Filter GeoJSON features by python expression.
  info     Print information about a dataset.
  insp     Open a dataset and start an interpreter.
  load     Load GeoJSON to a dataset in another format.
  ls       List layers in a datasource.
  rm       Remove a datasource or an individual layer.
```

It is developed using the `click` package and is new in 1.1.6.

5.1 bounds

New in 1.4.5.

Fio-bounds reads LF or RS-delimited GeoJSON texts, either features or collections, from stdin and prints their bounds with or without other data to stdout.

With no options, it works like this:

```
$ fio cat docs/data/test_uk.shp | head -n 1 \
> | fio bounds
[0.735, 51.357216, 0.947778, 51.444717]
```

Using `--with-id` gives you

```
$ fio cat docs/data/test_uk.shp | head -n 1 \
> | fio bounds --with-id
{"id": "0", "bbox": [0.735, 51.357216, 0.947778, 51.444717]}
```

5.2 calc

New in 1.7b1

The calc command creates a new property on GeoJSON features using the specified expression.

The expression is evaluated in a restricted namespace containing 4 functions (*sum*, *pow*, *min*, *max*), the *math* module, the shapely *shape* function, type conversions (*bool*, *int*, *str*, *len*, *float*), and an object *f* representing the feature to be evaluated. This *f* object allows access in javascript-style dot notation for convenience.

The expression will be evaluated for each feature and its return value will be added to the properties as the specified property_name. Existing properties will not be overwritten by default (an *Exception* is raised).

```
$ fio cat data.shp | fio calc sumAB  "f.properties.A + f.properties.B"
```

5.3 cat

The cat command concatenates the features of one or more datasets and prints them as a JSON text sequence of features. In other words: GeoJSON feature objects, possibly pretty printed, optionally separated by ASCII RS (x1e) chars using *–rs*.

The output of `fio cat` can be piped to `fio load` to create new concatenated datasets.

```
$ fio cat docs/data/test_uk.shp docs/data/test_uk.shp \
> | fio load /tmp/double.shp --driver Shapefile
$ fio info /tmp/double.shp --count
96
$ fio info docs/data/test_uk.shp --count
48
```

New in 1.4.0.

The cat command provides optional methods to filter data, which are different to the `fio filter` tool. A bounding box `--bbox w,s,e,n` tests for a spatial intersection with the geometries. An attribute filter `--where TEXT` can use

an SQL WHERE clause. If more than one datasets is passed to `fio cat`, the attributes used in the WHERE clause must be valid for each dataset.

5.4 collect

The collect command takes a JSON text sequence of GeoJSON feature objects, such as the output of `fio cat` and writes a GeoJSON feature collection.

```
$ fio cat docs/data/test_uk.shp docs/data/test_uk.shp \
> | fio collect > /tmp/collected.json
$ fio info /tmp/collected.json --count
96
```

New in 1.4.0.

5.5 distrib

The inverse of fio-collect, fio-distrib takes a GeoJSON feature collection and writes a JSON text sequence of GeoJSON feature objects.

```
$ fio info --count tests/data/coutwildrnp.shp
67
$ fio cat tests/data/coutwildrnp.shp | fio collect | fio distrib | wc -l
67
```

New in 1.4.0.

5.6 dump

The dump command reads a vector dataset and writes a GeoJSON feature collection to stdout. Its output can be piped to `fio load` (see below).

```
$ fio dump docs/data/test_uk.shp --indent 2 --precision 2 | head
{
  "features": [
    {
      "geometry": {
        "coordinates": [
          [
            [
              0.9,
              51.36
            ],
```

You can optionally dump out JSON text sequences using `--x-json-seq`. Since version 1.4.0, `fio cat` is the better tool for generating sequences.

```
$ fio dump docs/data/test_uk.shp --precision 2 --x-json-seq | head -n 2
{"geometry": {"coordinates": [[[0.9, 51.36], [0.89, 51.36], [0.79, 51.37], [0.78, 51.37],
↪ [0.77, 51.38], [0.76, 51.38], [0.75, 51.39], [0.74, 51.4], [0.73, 51.41], [0.74, 51.
↪43], [0.75, 51.44], [0.76, 51.44], [0.79, 51.44], [0.89, 51.42], [0.9, 51.42], [0.91,␣
↪51.42], [0.93, 51.4], [0.94, 51.39], [0.94, 51.38], [0.95, 51.38], [0.95, 51.37], [0.
↪95, 51.37], [0.94, 51.37], [0.9, 51.36], [0.9, 51.36]]], "type": "Polygon"}, "id": "0",
↪ "properties": {"AREA": 244820.0, "CAT": 232.0, "CNTRY_NAME": "United Kingdom", "FIPS_
↪CNTRY": "UK", "POP_CNTRY": 60270708.0}, "type": "Feature"}
{"geometry": {"coordinates": [[[-4.66, 51.16], [-4.67, 51.16], [-4.67, 51.16], [-4.67,␣
↪51.17], [-4.67, 51.19], [-4.67, 51.19], [-4.67, 51.2], [-4.66, 51.2], [-4.66, 51.19],␣
↪[-4.65, 51.16], [-4.65, 51.16], [-4.65, 51.16], [-4.66, 51.16]]], "type": "Polygon"},
↪"id": "1", "properties": {"AREA": 244820.0, "CAT": 232.0, "CNTRY_NAME": "United Kingdom
↪", "FIPS_CNTRY": "UK", "POP_CNTRY": 60270708.0}, "type": "Feature"}
```

5.7 info

The info command prints information about a dataset as a JSON object.

```
$ fio info docs/data/test_uk.shp --indent 2
{
  "count": 48,
  "crs": "+datum=WGS84 +no_defs +proj=longlat",
  "driver": "ESRI Shapefile",
  "bounds": [
    -8.621389,
    49.911659,
    1.749444,
    60.844444
  ],
  "schema": {
    "geometry": "Polygon",
    "properties": {
      "CAT": "float:16",
      "FIPS_CNTRY": "str:80",
      "CNTRY_NAME": "str:80",
      "AREA": "float:15.2",
      "POP_CNTRY": "float:15.2"
    }
  }
}
```

You can process this JSON using, e.g., underscore-cli.

```
$ fio info docs/data/test_uk.shp | underscore extract count
48
```

You can also optionally get single info items as plain text (not JSON) strings

```
$ fio info docs/data/test_uk.shp --count
48
```

(continues on next page)

(continued from previous page)

```
$ fio info docs/data/test_uk.shp --bounds
-8.621389 49.911659 1.749444 60.844444
```

5.8 load

The load command reads GeoJSON features from stdin and writes them to a vector dataset using another format.

```
$ fio dump docs/data/test_uk.shp \
> | fio load /tmp/test.shp --driver Shapefile
```

This command also supports GeoJSON text sequences. RS-separated sequences will be detected. If you want to load LF-separated sequences, you must specfiy `--x-json-seq`.

```
$ fio cat docs/data/test_uk.shp | fio load /tmp/foo.shp --driver Shapefile
$ fio info /tmp/foo.shp --indent 2
{
  "count": 48,
  "crs": "+datum=WGS84 +no_defs +proj=longlat",
  "driver": "ESRI Shapefile",
  "bounds": [
    -8.621389,
    49.911659,
    1.749444,
    60.844444
  ],
  "schema": {
    "geometry": "Polygon",
    "properties": {
      "AREA": "float:24.15",
      "CNTRY_NAME": "str:80",
      "POP_CNTRY": "float:24.15",
      "FIPS_CNTRY": "str:80",
      "CAT": "float:24.15"
    }
  }
}
```

The underscore-cli process command is another way of turning a GeoJSON feature collection into a feature sequence.

```
$ fio dump docs/data/test_uk.shp \
> | underscore process \
> 'each(data.features,function(o){console.log(JSON.stringify(o))})' \
> | fio load /tmp/test-seq.shp --x-json-seq --driver Shapefile
```

5.9 filter

The filter command reads GeoJSON features from stdin and writes the feature to stdout *if* the provided expression evalutates to *True* for that feature.

The python expression is evaluated in a restricted namespace containing 3 functions (*sum*, *min*, *max*), the *math* module, the shapely *shape* function, and an object *f* representing the feature to be evaluated. This *f* object allows access in javascript-style dot notation for convenience.

If the expression evaluates to a "truthy" value, the feature is printed verbatim. Otherwise, the feature is excluded from the output.

```
$ fio cat data.shp \
> | fio filter "f.properties.area > 1000.0" \
> | fio collect > large_polygons.geojson
```

Would create a geojson file with only those features from *data.shp* where the area was over a given threshold.

Note this tool is different than `fio cat --where TEXT ...`, which provides SQL WHERE clause filtering of feature attributes.

5.10 rm

The `fio rm` command deletes an entire datasource or a single layer in a multi-layer datasource. If the datasource is composed of multiple files (e.g. an ESRI Shapefile) all of the files will be removed.

```
$ fio rm countries.shp
$ fio rm --layer forests land_cover.gpkg
```

New in 1.8.0.

5.11 Coordinate Reference System Transformations

The `fio cat` command can optionally transform feature geometries to a new coordinate reference system specified with `--dst_crs`. The `fio collect` command can optionally transform from a coordinate reference system specified with `--src_crs` to the default WGS84 GeoJSON CRS. Like collect, `fio load` can accept non-WGS84 features, but as it can write files in formats other than GeoJSON, you can optionally specify a `--dst_crs`. For example, the WGS84 features read from docs/data/test_uk.shp,

```
$ fio cat docs/data/test_uk.shp --dst_crs EPSG:3857 \
> | fio collect --src_crs EPSG:3857 > /tmp/foo.json
```

make a detour through EPSG:3857 (Web Mercator) and are transformed back to WGS84 by fio cat. The following,

```
$ fio cat docs/data/test_uk.shp --dst_crs EPSG:3857 \
> | fio load --src_crs EPSG:3857 --dst_crs EPSG:4326 --driver Shapefile \
> /tmp/foo.shp
```

does the same thing, but for ESRI Shapefile output.

New in 1.4.2.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Kent1978] William Kent, Data and Reality, North Holland, 1978.

[ESRI1998] ESRI Shapefile Technical Description. July 1998. https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[GeoJSON] https://geojson.org

[JSON] https://www.ietf.org/rfc/rfc4627

[SFA] https://en.wikipedia.org/wiki/Simple_feature_access

PYTHON MODULE INDEX

f

INDEX

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Z