FastKML Documentation

Release dev

Christian Ledermann & Ian Lee

Mar 06, 2018

Contents

Fastkml is continually tested with *Travis CI*: 

Is Maintained and documented: Follows best practises: Supports python 2 and 3: fastkml is a library to read, write and manipulate KML files. It aims to keep it simple and fast (using lxml if available). "Fast" refers to the time you spend to write and read KML files, as well as the time you spend to get acquainted with the library or to create KML objects. It provides a subset of KML and is aimed at documents that can be read from multiple clients such as openlayers and google maps rather than to give you all functionality that KML on google earth provides.

For more details about the KML Specification, check out the KML Reference on the Google developers site.

CHAPTER 1

Rationale

Why yet another KML library? None of the existing ones quite fit my requirements, namely:

- fastkml can *read and write* KML files, feeding fastkml's output back into fastkml and serializing it again will result in the same output.
- You can parse any KML snippet, it does not need to be a complete KML document.
- It runs on Python 2 and 3.
- It is fully tested and actively maintained.
- Geometries are handled in the *__geo_interface__* standard.
- Minimal dependencies, pure Python.
- If available, lxml will be used to increase its speed.

1.1 Quickstart

```
$ pip install fastkml

# Start working with the library
$ python
>>> from fastkml import kml
>>> k = kml.KML()
```

1.2 Installation

fastkml works with CPython version 2.6, 2.7, 3.2, 3.3, 3.4 and is continually tested with TravisCI for these version. The tests break intermittently for pypy and pypy3 so they are not tested but *should* work, Jython and IronPython are not tested but *should* work. 

fastkml works on Unix/Linux, OS X, and Windows.

Install it with `pip install fastkml` or `easy_install fastkml`.

If you use fastkml extensively or need to process big KML files, consider installing lxml as it speeds up processing.

You can install all requirements for working with fastkml by using pip from the base of the source tree:

```
pip install -r requirements.txt
```

Note: Shapely requires that libgeos be installed on your system. `apt-get install libgeos-dev` will install these requirements for you on Debian- based systems.

1.3 Usage Guide

You can find more examples in the included `test_main.py` file or in collective.geo.fastkml, here is a quick overview:

(The examples below are available as standalone scripts in the `examples` folder.)

1.3.1 Build a KML from Scratch

Example how to build a simple KML file from the Python interpreter.

```
# Import the library
>>> from fastkml import kml
>>> from shapely.geometry import Point, LineString, Polygon

# Create the root KML object
>>> k = kml.KML()
>>> ns = '{http://www.opengis.net/kml/2.2}'

# Create a KML Document and add it to the KML root object
>>> d = kml.Document(ns, 'docid', 'doc name', 'doc description')
>>> k.append(d)

# Create a KML Folder and add it to the Document
>>> f = kml.Folder(ns, 'fid', 'f name', 'f description')
>>> d.append(f)

# Create a KML Folder and nest it in the first Folder
>>> nf = kml.Folder(ns, 'nested-fid', 'nested f name', 'nested f description')
>>> f.append(nf)

# Create a second KML Folder within the Document
>>> f2 = kml.Folder(ns, 'id2', 'name2', 'description2')
>>> d.append(f2)

# Create a Placemark with a simple polygon geometry and add it to the
# second folder of the Document
>>> p = kml.Placemark(ns, 'id', 'name', 'description')
>>> p.geometry =  Polygon([(0, 0, 0), (1, 1, 0), (1, 0, 1)])
>>> f2.append(p)

# Print out the KML Object as a string
>>> print k.to_string(prettyprint=True)
```

```
'<kml:kml xmlns:ns0="http://www.opengis.net/kml/2.2">
  <kml:Document id="docid">
    <kml:name>doc name</kml:name>
    <kml:description>doc description</kml:description>
    <kml:visibility>1</kml:visibility>
    <kml:open>0</kml:open>
    <kml:Folder id="fid">
      <kml:name>f name</kml:name>
      <kml:description>f description</kml:description>
      <kml:visibility>1</kml:visibility>
      <kml:open>0</kml:open>
      <kml:Folder id="nested-fid">
        <kml:name>nested f name</kml:name>
        <kml:description>nested f description</kml:description>
        <kml:visibility>1</kml:visibility>
        <kml:open>0</kml:open>
      </kml:Folder>
    </kml:Folder>
    <kml:Folder id="id2">
      <kml:name>name2</kml:name>
      <kml:description>description2</kml:description>
      <kml:visibility>1</kml:visibility>
      <kml:open>0</kml:open>
      <kml:Placemark id="id">
        <kml:name>name</kml:name>
        <kml:description>description</kml:description>
        <kml:visibility>1</kml:visibility>
        <kml:open>0</kml:open>
        <kml:Polygon>
          <kml:outerBoundaryIs>
            <kml:LinearRing>
              <kml:coordinates>
                0.000000,0.000000,0.000000
                1.000000,1.000000,0.000000
                1.000000,0.000000,1.000000
                0.000000,0.000000,0.000000
              </kml:coordinates>
            </kml:LinearRing>
         </kml:outerBoundaryIs>
        </kml:Polygon>
      </kml:Placemark>
    </kml:Folder>
  </kml:Document>
</kml:kml>'
```

1.3.2 Read a KML File/String

You can create a KML object by reading a KML file as a string

```
# Start by importing the kml module
>>> from fastkml import kml

#Read file into string and convert to UTF-8 (Python3 style)
>>> with open(kml_file, 'rt', encoding="utf-8") as myfile:
...     doc=myfile.read()
```

```
# OR

# Setup the string which contains the KML file we want to read
>>> doc = """<?xml version="1.0" encoding="UTF-8"?>
... <kml xmlns="http://www.opengis.net/kml/2.2">
... <Document>
...   <name>Document.kml</name>
...   <open>1</open>
...   <Style id="exampleStyleDocument">
...     <LabelStyle>
...       <color>ff0000cc</color>
...     </LabelStyle>
...   </Style>
...   <Placemark>
...     <name>Document Feature 1</name>
...     <styleUrl>#exampleStyleDocument</styleUrl>
...     <Point>
...       <coordinates>-122.371,37.816,0</coordinates>
...     </Point>
...   </Placemark>
...   <Placemark>
...     <name>Document Feature 2</name>
...     <styleUrl>#exampleStyleDocument</styleUrl>
...     <Point>
...       <coordinates>-122.370,37.817,0</coordinates>
...     </Point>
...   </Placemark>
... </Document>
... </kml>"""

# Create the KML object to store the parsed result
>>> k = kml.KML()

# Read in the KML string
>>> k.from_string(doc)

# Next we perform some simple sanity checks

# Check that the number of features is correct
# This corresponds to the single ``Document``
>>> features = list(k.features())
>>> len(features)
1

# Check that we can access the features as a generator
# (The two Placemarks of the Document)
>>> features[0].features()
<generator object features at 0x2d7d870>
>>> f2 = list(features[0].features())
>>> len(f2)
2

# Check specifics of the first Placemark in the Document
>>> f2[0]
<fastkml.kml.Placemark object at 0x2d791d0>
>>> f2[0].description
>>> f2[0].name
'Document Feature 1'
```

```
# Check specifics of the second Placemark in the Document
>>> f2[1].name
'Document Feature 2'
>>> f2[1].name = "ANOTHER NAME"

# Verify that we can print back out the KML object as a string
>>> print k.to_string(prettyprint=True)
<kml:kml xmlns:ns0="http://www.opengis.net/kml/2.2">
  <kml:Document>
    <kml:name>Document.kml</kml:name>
    <kml:visibility>1</kml:visibility>
    <kml:open>1</kml:open>
    <kml:Style id="exampleStyleDocument">
      <kml:LabelStyle>
        <kml:color>ff0000cc</kml:color>
        <kml:scale>1.0</kml:scale>
      </kml:LabelStyle>
    </kml:Style>
    <kml:Placemark>
      <kml:name>Document Feature 1</kml:name>
      <kml:visibility>1</kml:visibility>
      <kml:open>0</kml:open>
      <kml:Point>
        <kml:coordinates>-122.371000,37.816000,0.000000</kml:coordinates>
      </kml:Point>
    </kml:Placemark>
    <kml:Placemark>
      <kml:name>ANOTHER NAME</kml:name>
      <kml:visibility>1</kml:visibility>
      <kml:open>0</kml:open>
      <kml:Point>
        <kml:coordinates>-122.370000,37.817000,0.000000</kml:coordinates>
      </kml:Point>
    </kml:Placemark>
  </kml:Document>
</kml:kml>
```

1.4 Reference Guide

1.4.1 Atom

KML 2.2 supports new elements for including data about the author and related website in your KML file. This information is displayed in geo search results, both in Earth browsers such as Google Earth, and in other applications such as Google Maps. The ascription elements used in KML are as follows:

atom:author element - parent element for atom:name atom:name element - the name of the author atom:link element - contains the href attribute href attribute - URL of the web page containing the KML/KMZ file

These elements are defined in the Atom Syndication Format. The complete specification is found at http://atompub.org.

This library only implements a subset of Atom that is useful with KML

class fastkml.atom.**Author**(*ns=None*, *name=None*, *uri=None*, *email=None*)

Names one author of the feed/entry. A feed/entry may have multiple authors.

class fastkml.atom.**Contributor**(*ns=None*, *name=None*, *uri=None*, *email=None*)
Names one contributor to the feed/entry. A feed/entry may have multiple contributor elements.

class fastkml.atom.**Link**(*ns=None*, *href=None*, *rel=None*, *type=None*, *hreflang=None*, *title=None*, *length=None*)
Identifies a related Web page. The type of relation is defined by the rel attribute. A feed is limited to one alternate per type and hreflang. <link> is patterned after html's link element. It has one required attribute, href, and five optional attributes: rel, type, hreflang, title, and length.

to_string(*prettyprint=True*)
Return the ATOM Object as serialized xml

fastkml.atom.**check_email**()
match(string[, pos[, endpos]]) –> match object or None. Matches zero or more characters at the beginning of the string

1.4.2 Base

Abstract base classes

1.4.3 Config

Frequently used constants and configuration options

1.4.4 Geometry

Import the geometries from shapely if it is installed or otherwise from Pygeoif

class fastkml.geometry.**Geometry**(*ns=None*, *id=None*, *geometry=None*, *extrude=False*, *tessellate=False*, *altitude_mode=None*)

1.4.5 GX

With the launch of Google Earth 5.0, Google has provided extensions to KML to support a number of new features. These extensions use the gx prefix and the following namespace URI:

```
xmlns:gx="http://www.google.com/kml/ext/2.2"
```

This namespace URI must be added to the <kml> element in any KML file using gx-prefixed elements:

```
<kml
    xmlns="http://www.opengis.net/kml/2.2"
    xmlns:gx="http://www.google.com/kml/ext/2.2"
>
```

Extensions to KML may not be supported in all geo-browsers. If your browser doesn't support particular extensions, the data in those extensions should be silently ignored, and the rest of the KML file should load without errors.

Elements that currently use the gx prefix are:

- gx:altitudeMode
- gx:altitudeOffset
- gx:angles

- gx:AnimatedUpdate
- gx:balloonVisibility
- gx:coord
- gx:delayedStart
- gx:drawOrder
- gx:duration
- gx:FlyTo
- gx:flyToMode
- gx:h
- gx:horizFov
- gx:interpolate
- gx:labelVisibility
- gx:LatLonQuad
- gx:MultiTrack
- gx:vieweroptions
- gx:outerColor
- gx:outerWidth
- gx:physicalWidth
- gx:Playlist
- gx:playMode
- gx:SoundCue
- gx:TimeSpan
- gx:TimeStamp
- gx:Tour
- gx:TourControl
- gx:TourPrimitive
- gx:Track
- gx:ViewerOptions
- gx:w
- gx:Wait
- gx:x
- gx:y

The complete XML schema for elements in this extension namespace is located at http://developers.google.com/kml/schema/kml22gx.xsd.

1.4.6 KML

KML is an open standard officially named the OpenGIS KML Encoding Standard (OGC KML). It is maintained by the Open Geospatial Consortium, Inc. (OGC). The complete specification for OGC KML can be found at http://www.opengeospatial.org/standards/kml/.

The complete XML schema for KML is located at http://schemas.opengis.net/kml/.

class fastkml.kml.**Data**(*ns=None*, *name=None*, *value=None*, *display_name=None*)
Represents an untyped name/value pair with optional display name.

class fastkml.kml.**Document**(*ns=None*, *id=None*, *name=None*, *description=None*, *styles=None*, *styleUrl=None*)
A Document is a container for features and styles. This element is required if your KML file uses shared styles or schemata for typed extended data

class fastkml.kml.**ExtendedData**(*ns=None*, *elements=None*)
Represents a list of untyped name/value pairs. See docs:

-> 'Adding Untyped Name/Value Pairs' https://developers.google.com/kml/documentation/extendeddata

class fastkml.kml.**Folder**(*ns=None*, *id=None*, *name=None*, *description=None*, *styles=None*, *styleUrl=None*)
A Folder is used to arrange other Features hierarchically (Folders, Placemarks, #NetworkLinks, or #Overlays).

class fastkml.kml.**GroundOverlay**(*ns=None*, *id=None*, *name=None*, *description=None*, *styles=None*, *styleUrl=None*)
This element draws an image overlay draped onto the terrain. The <href> child of <Icon> specifies the image to be used as the overlay. This file can be either on a local file system or on a web server. If this element is omitted or contains no <href>, a rectangle is drawn using the color and LatLonBox bounds defined by the ground overlay.

class fastkml.kml.**KML**(*ns=None*)
represents a KML File

append(*kmlobj*)
append a feature

features()
iterate over features

from_string(*xml_string*)
create a KML object from a xml string

to_string(*prettyprint=False*)
Return the KML Object as serialized xml

class fastkml.kml.**Placemark**(*ns=None*, *id=None*, *name=None*, *description=None*, *styles=None*, *styleUrl=None*, *extended_data=None*)
A Placemark is a Feature with associated Geometry. In Google Earth, a Placemark appears as a list item in the Places panel. A Placemark with a Point has an icon associated with it that marks a point on the Earth in the 3D viewer.

class fastkml.kml.**Schema**(*ns=None*, *id=None*, *name=None*, *fields=None*)
Specifies a custom KML schema that is used to add custom data to KML Features. The "id" attribute is required and must be unique within the KML file. <Schema> is always a child of <Document>.

append(*type*, *name*, *displayName=None*)
append a field. The declaration of the custom field, must specify both the type and the name of this field. If either the type or the name is omitted, the field is ignored.

The type can be one of the following: string int uint short ushort float double bool

<displayName> The name, if any, to be used when the field name is displayed to the Google Earth user. Use the [CDATA] element to escape standard HTML markup.

class fastkml.kml.**SchemaData**(*ns=None*, *schema_url=None*, *data=None*)
<SchemaData schemaUrl="anyURI"> This element is used in conjunction with <Schema> to add typed custom data to a KML Feature. The Schema element (identified by the schemaUrl attribute) declares the custom data type. The actual data objects ("instances" of the custom data) are defined using the SchemaData element. The <schemaURL> can be a full URL, a reference to a Schema ID defined in an external KML file, or a reference to a Schema ID defined in the same KML file.

class fastkml.kml.**TimeSpan**(*ns=None*, *id=None*, *begin=None*, *begin_res=None*, *end=None*, *end_res=None*)
Represents an extent in time bounded by begin and end dateTimes.

class fastkml.kml.**TimeStamp**(*ns=None*, *id=None*, *timestamp=None*, *resolution=None*)
Represents a single moment in time.

1.4.7 Styles

Once you've created features within Google Earth and examined the KML code Google Earth generates, you'll notice how styles are an important part of how your data is displayed.

class fastkml.styles.**BalloonStyle**(*ns=None*, *id=None*, *bgColor=None*, *textColor=None*, *text=None*, *displayMode=None*)
Specifies how the description balloon for placemarks is drawn. The <bgColor>, if specified, is used as the background color of the balloon.

class fastkml.styles.**IconStyle**(*ns=None*, *id=None*, *color=None*, *colorMode=None*, *scale=1.0*, *heading=None*, *icon_href=None*)
Specifies how icons for point Placemarks are drawn

class fastkml.styles.**LabelStyle**(*ns=None*, *id=None*, *color=None*, *colorMode=None*, *scale=1.0*)
Specifies how the <name> of a Feature is drawn in the 3D viewer

class fastkml.styles.**LineStyle**(*ns=None*, *id=None*, *color=None*, *colorMode=None*, *width=1*)
Specifies the drawing style (color, color mode, and line width) for all line geometry. Line geometry includes the outlines of outlined polygons and the extruded "tether" of Placemark icons (if extrusion is enabled).

class fastkml.styles.**PolyStyle**(*ns=None*, *id=None*, *color=None*, *colorMode=None*, *fill=1*, *outline=1*)
Specifies the drawing style for all polygons, including polygon extrusions (which look like the walls of buildings) and line extrusions (which look like solid fences).

class fastkml.styles.**Style**(*ns=None*, *id=None*, *styles=None*)
A Style defines an addressable style group that can be referenced by StyleMaps and Features. Styles affect how Geometry is presented in the 3D viewer and how Features appear in the Places panel of the List view. Shared styles are collected in a <Document> and must have an id defined for them so that they can be referenced by the individual Features that use them.

class fastkml.styles.**StyleMap**(*ns=None*, *id=None*, *normal=None*, *highlight=None*)
A <StyleMap> maps between two different Styles. Typically a <StyleMap> element is used to provide separate normal and highlighted styles for a placemark, so that the highlighted version appears when the user mouses over the icon in Google Earth.

class fastkml.styles.**StyleUrl**(*ns=None*, *id=None*, *url=None*)
URL of a <Style> or <StyleMap> defined in a Document. If the style is in the same file, use a # reference. If the style is defined in an external file, use a full URL along with # referencing.

1.5 Contributing

1.5.1 Getting Involved

So you'd like to contribute? That's awesome! We would love to have your help, especially in the following ways:

- Making Pull Requests for code, tests, or docs
- Commenting on open issues and pull requests
- Suggesting new features

1.5.2 Pull Requests

Start by submitting a pull request on GitHub against the *master* branch of the repository. Your pull request should provide a good description of the change you are making, and/or the bug that you are fixing. This will then trigger a build in Travis-CI where your contribution will be tested to verify it does not break existing functionality.

1.5.3 Running Tests Locally

You can make use of tox >= 1.8 to test the entire matrix of options:

- with / without lxml
- pygeoif vs shapely
- py26,py27,py32,py33,py34

as well as pep8 style checking in a single call (this approximates what happens when the package is run through Travis-CI)

```
# Install tox
pip install tox>=1.8

# Run tox
tox

# Or optionally
# (to skip tests for Python versions you do not have installed)
tox --skip-missing-interpreters
```

This will run through all of the tests and produce an output similar to:

```
_____ summary _____
↪_____
SKIPPED:  py26: InterpreterNotFound: python2.6
  py27: commands succeeded
SKIPPED:  py32: InterpreterNotFound: python3.2
SKIPPED:  py33: InterpreterNotFound: python3.3
  py34: commands succeeded
SKIPPED:  py26-shapely: InterpreterNotFound: python2.6
SKIPPED:  py26-lxml: InterpreterNotFound: python2.6
  py27-shapely: commands succeeded
  py27-lxml: commands succeeded
SKIPPED:  py32-shapely: InterpreterNotFound: python3.2
SKIPPED:  py32-lxml: InterpreterNotFound: python3.2
SKIPPED:  py33-shapely: InterpreterNotFound: python3.3
```

```
SKIPPED:  py33-lxml: InterpreterNotFound: python3.3
  py34-shapely: commands succeeded
  py34-lxml: commands succeeded
SKIPPED:  py26-shapely-lxml: InterpreterNotFound: python2.6
  py27-shapely-lxml: commands succeeded
SKIPPED:  py32-shapely-lxml: InterpreterNotFound: python3.2
SKIPPED:  py33-shapely-lxml: InterpreterNotFound: python3.3
  py34-shapely-lxml: commands succeeded
  pep8: commands succeeded
  congratulations :)
```

You are primarily looking for the `congratulations :)` line at the bottom, signifying that the code is working as expected on all configurations available.

Python Module Index

f

Index