simplekml Documentation

Release 1.3.6

Kyle Lancaster

Sep 16, 2021

Contents

The python package simplekml was created to generate kml (or kmz). It was designed to alleviate the burden of having to study KML in order to achieve anything worthwhile with it. If you have a simple understanding of the structure of KML, then simplekml is easy to run with and create usable KML.

If you want get started right away you can download the latest version from PyPi and then head over to *Getting Started* for a quick example.

Simplekml is licensed under the GNU Lesser General Public License.

CHAPTER 1

Resources

- Download the latest version from PyPi.
- KML Reference as published by Google for a good understanding of what KML is capable of.
- Samples File (right-click > Save link as...) for example code with corresponding KML. This file is simply a network link. When examples and tutorials are updated the updates will reflect in the sample file (needs to be redownloaded if it was downloaded prior to 1.2.8).
- Polycircles is a Python package that uses simplekml to create Polygonal circle approximation KMLs (because KML does not support circle geometry).

CHAPTER 2

Table of Contents

2.1 Download

To get the latest version of simplekml head over to:

http://pypi.python.org/pypi/simplekml.

2.2 Getting Started

2.2.1 Quick Example

Here is a quick example to get you started:

```
import simplekml
kml = simplekml.Kml()
kml.newpoint(name="Kirstenbosch", coords=[(18.432314,-33.988862)])  # lon, lat,␣
↪optional height
kml.save("botanicalgarden.kml")
```

2.2.2 Concepts

With simplekml everything starts with creating an instance of the *simplekml.Kml* class:

```
kml = simplekml.Kml()
```

The compilation of the KML file will be done through this class. The base feature attached to this class is a document, all arguments passed to the class on creation are the same as that of a *simplekml.Document*. To change any properties after creation you can do so through the *simplekml.Kml.document()* property:

```
kml.document.name = "Test"
```

To create a fully fledged KML document a document tree is created via calls to various functions like:

```
kml.newpoint()
```

These calls build up a relationships between classes which will be converted into KML when a call to `kml.save()`, `kml.savekmz()` or `kml.kml()` is made. These relationships are created in the background and do not need direct manipulation.

There are three ways to go about using simplekml: property changes, property assignment and a mixture of property changes and assignment.

With property assignment, when you change the properties of the objects you are working with the class of the property (and thus th associated KML tag) gets activated. What is meant by this is when starting to build up your KML, the KML tags are kept to a minimum when generated by `kml.save()`, `kml.savekmz()` or `kml.kml()` and more and more tags are added as you assign values to properties. For example, below we create an kml object and attach a point to it and then print the result to screen:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name="A Point")
print(kml.kml())
```

This is what is generated:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.
↪2">
    <Document id="feat_1">
        <Placemark id="feat_2">
            <name>A Point</name>
            <Point id="geom_0">
                <coordinates>0.0, 0.0, 0.0</coordinates>
            </Point>
        </Placemark>
    </Document>
</kml>
```

As you can see, only the least amount of KML is generated for it to be viewable in Google Earth. In this case we have a document containing a placemark containing a point and the point has the default coordinate of 0.0, 0.0, 0.0 and the name we gave it: *A Point*

Now we can add a description to our point after it was created:

```
pnt.description = "This is a description"
```

And the result:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.
↪2">
    <Document id="feat_1">
        <Placemark id="feat_2">
            <name>A Point</name>
            <description>This is a description</description>
            <Point id="geom_0">
```

(continues on next page)

(continued from previous page)

```
                <coordinates>0.0, 0.0, 0.0</coordinates>
            </Point>
        </Placemark>
    </Document>
</kml>
```

You can see that the description tag was created that contains our description. This is what is meant by activating the KML tags. Here is a more complicated example where we are going to set the *simplekml.Point.snippet()* property:

```
pnt.snippet.content = "This is the content of the snippet"
pnt.snippet.maxlines = 1
```

The result:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.
↪2">
    <Document id="feat_1">
        <Placemark id="feat_2">
            <name>A Point</name>
            <description>This is a description</description>
            <Snippet maxLines="1">This is the content of the snippet</Snippet>
            <Point id="geom_0">
                <coordinates>0.0, 0.0, 0.0</coordinates>
            </Point>
        </Placemark>
    </Document>
</kml>
```

Above the snippet had two properties that we changed: *content* and *maxlines*. Again, notice the tags being generated. This is what is meant by the property changes method. This method is fine for one or two properties that have to be changed, but when we have the following, typing becomes repetitive and tedious:

```
pnt.lookat.gxaltitudemode = simplekml.GxAltitudeMode.relativetoseafloor
pnt.lookat.latitude = 0.0
pnt.lookat.longitude = 0.0
pnt.lookat.range = 3000
pnt.lookat.heading = 56
pnt.lookat.tilt = 78
```

The result:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.
↪2">
    <Document id="feat_1">
        <Placemark id="feat_2">
            <name>A Point</name>
            <description>This is a description</description>
            <LookAt>
                <longitude>18.356852</longitude>
                <latitude>-34.028242</latitude>
                <heading>56</heading>
                <tilt>78</tilt>
                <range>3000</range>
```

(continues on next page)

(continued from previous page)

```
                <gx:altitudeMode>relativeToSeaFloor </gx:altitudeMode>
            </LookAt>
            <Snippet maxLines="1">This is the content of the snippet</Snippet>
            <Point id="geom_0">
                <coordinates>0.0, 0.0, 0.0</coordinates>
            </Point>
        </Placemark>
    </Document>
</kml>
```

In this case it would be easier to use the property assignment method. Here we are going to create an instance of the *simplekml.LookAt* class and then assign it to the lookat property of the point:

```
pnt.lookat = simplekml.LookAt(gxaltitudemode=simplekml.GxAltitudeMode.
↪relativetoseafloor,
                              latitude=0.0, longitude=0.0,
                              range=3000, heading=56, tilt=78)
```

As you can plainly see, that is far less typing. So, for the property assignment method, all you need to do is create an instance of the class you want to assign to the property with its properties filled out and then assign it.

And finally, you mix the two methods as you see fit. Each circumstance will require a different approach.

Note: If there is a property that needs to be removed from the tree, simply set it to *None*:

```
kml = simplekml.Kml(name="A name")  # Give the KML document a name
kml.document.name = None  # remove the name we gave above
```

This is useful to prevent an image from being displayed for a point. By default a point has the image of a yellow push pin (in Google Earth), but if you want to remove it you have to do this:

```
pnt.style.iconstyle.icon.href = None
```

This removes the href from the icon, thus nothing will be displayed in google earth, except the point's text.

Also, if you access some properties without assigning to them, the KML tag is also created. For instance, if you print the snippet of a point without a having changed any of its properties, a blank snippet tag will be generated:

```
import simplekml
kml = simplekml.Kml()
print(kml.document.snippet)
```

The following will be generated when saving the KML:

```
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.
↪2">
    <Document id="feat_1">
        <Snippet/>
    </Document>
</kml>
```

2.2.3 Creating a KML document

To create a KML document you just have to import simplekml and then create an instance of *simplekml.Kml*. Doing this will create a KML 'file' in memory and assign a *simplekml.Document* as the main feature:

```
import simplekml
kml = simplekml.Kml()
```

2.2.4 Saving a KML document

Simply call *kml.save("pathtomyfile.kml")* passing a path to the file you want to create. Alternatively you can call *kml.savekmz("pathtomyfile.kmz")* to save the KML as a KMZ, or even *kml.kml()* to get the KML as a string. See *simplekml.Kml.save()*, *simplekml.Kml.savekmz()* and *simplekml.Kml.kml()* for more details.

2.2.5 Creating a Point

A Point is a geographic location defined by longitude, latitude, and altitude.

Note: All coordinates in simplekml are in the order longitude, latitude and then an optional height.

Creating a *simplekml.Point* is quite simple and has been done in the section above. Once you have your *simplekml.Kml* object you have to ask it to create a new *simplekml.Point* for you by calling *simplekml.Kml.newpoint()*. If you call *simplekml.Kml.newpoint()* without any parameters a *simplekml.Point* is created at 0.0, 0.0 with no name. You can later change the name and location (among other things) by changing the attributes of the *simplekml.Point* instance that was returned to you by calling *simplekml.Kml.newpoint()*. Passing parameters to *simplekml.Kml.newpoint()* may be more convenient. All the attributes have to be set like so: *attributename=value*. See *simplekml.Point* for a list of possible parameters and attributes.

Here is an example:

```
pnt = kml.newpoint(name="Kirstenbosch", description="A botanical Garden",
                   coords=[(18.432314,-33.988862)])  # lon, lat optional height
```

The values of the above parameters can be changed later by directly assigning to them:

```
pnt.name = "Tree"
pnt.description = "A big plant."
```

2.2.6 Creating a LineString

A Linestring is a connected set of line segments.

A *simplekml.LineString* is created in a similar manner to a *simplekml.Point*, except you can have more than one coordinate. Just call *simplekml.Kml.newlinestring()*. See *simplekml.LineString* for a list of possible parameters and attributes.

Here is an example:

```
lin = kml.newlinestring(name="Pathway", description="A pathway in Kirstenbosch",
                        coords=[(18.43312,-33.98924), (18.43224,-33.98914),
                                (18.43144,-33.98911), (18.43095,-33.98904)])
```

2.2.7 Creating a Polygon

A Polygon is defined by an outer boundary and/or an inner boundary.

A *simplekml.Polygon* is created in a similar manner to a *simplekml.LineString*, except there is no coordinate parameter. Just call *simplekml.Kml.newpolygon()*. The coordinate parameter has been replaced with two others, *simplekml.Polygon.outerboundaryis()* and *simplekml.Polygon.innerboundaryis()*. The outer and inner boundaries describe the outside of the *simplekml.Polygon* and an inner opening. You pass a list of tuples to these parameters, as if it were a coordinate list. See *simplekml.Polygon* for a list of possible parameters and attributes.

Here is an example:

```
pol = kml.newpolygon(name="Atrium Garden",
                     outerboundaryis=[(18.43348,-33.98985), (18.43387,-33.99004),
                                      (18.43410,-33.98972), (18.43371,-33.98952),
                                      (18.43348,-33.98985)],
                     innerboundaryis=[(18.43360,-33.98982), (18.43386,-33.98995),
                                      (18.43401,-33.98974), (18.43376,-33.98962),
                                      (18.43360,-33.98982)])
```

2.3 Reference

2.3.1 AbstractViews

Camera

class simplekml.**Camera**(*roll=None*, ***kwargs*)

A virtual camera that views the scene.

The arguments are the same as the properties.

Basic Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint()
pnt.camera.latitude = 0.02
pnt.camera.longitude = 0.012
pnt.camera.altitude = 10000
pnt.camera.tilt = 45
pnt.camera.heading = 0
pnt.camera.roll = 0
pnt.camera.altitudemode = simplekml.AltitudeMode.relativetoground
kml.save("Camera.kml")
```

Assignment Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint()
camera = simplekml.Camera(latitude=0.0, longitude=0.0, altitude=0.0, roll=0,␣
↪tilt=45,
                          altitudemode=simplekml.AltitudeMode.relativetoground)
pnt.camera = camera
kml.save("Camera Alternative.kml")
```

altitude
Height above the earth in meters (m), accepts int.

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts *simplekml.AltitudeMode* constants.

gxaltitudemode
Specifies how the altitude for the Camera is interpreted.

With the addition of being relative to the sea floor. Accepts *simplekml.GxAltitudeMode* constants.

gxhorizfov
Rotation about the x axis, accepts float.

gxtimespan
Period of time, accepts *simplekml.GxTimeSpan*

gxtimestamp
Represents a single moment in time, accepts *simplekml.GxTimeStamp*

gxvieweroptions
Enables special viewing modes , accepts *simplekml.GxViewerOptions*

heading
Rotation about the z axis, accepts float.

latitude
Decimal degree value in WGS84 datum, accepts float.

longitude
Decimal degree value in WGS84 datum, accepts float.

roll
Rotation about the y axis, accepts float.

tilt
Rotation about the x axis, accepts float.

LookAt

class simplekml.**LookAt**(*range=None*, ***kwargs*)
Positions the camera in relation to the object that is being viewed.

The arguments are the same as the properties (most inherited from `simplekml.AbstractView`)

Usage:

```
import simplekml
kml = simplekml.Kml()
ls = kml.newlinestring(name='A LineString')
ls.coords = [(18.333868,-34.038274,10.0), (18.370618,-34.034421,10.0)]
ls.extrude = 1
ls.altitudemode = simplekml.AltitudeMode.relativetoground
ls.lookat.gxaltitudemode = simplekml.GxAltitudeMode.relativetoseafloor
ls.lookat.latitude = -34.028242
ls.lookat.longitude = 18.356852
ls.lookat.range = 3000
ls.lookat.heading = 56
ls.lookat.tilt = 78
kml.save("LookAt.kml")
```

altitude
Height above the earth in meters (m), accepts int.

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts *simplekml.AltitudeMode* constants.

gxaltitudemode
Specifies how the altitude for the Camera is interpreted.

With the addition of being relative to the sea floor. Accepts *simplekml.GxAltitudeMode* constants.

gxhorizfov
Rotation about the x axis, accepts float.

gxtimespan
Period of time, accepts *simplekml.GxTimeSpan*

gxtimestamp
Represents a single moment in time, accepts *simplekml.GxTimeStamp*

gxvieweroptions
Enables special viewing modes , accepts *simplekml.GxViewerOptions*

heading
Rotation about the z axis, accepts float.

latitude
Decimal degree value in WGS84 datum, accepts float.

longitude
Decimal degree value in WGS84 datum, accepts float.

range
Distance in meters from the point, accepts int.

tilt
Rotation about the x axis, accepts float.

GxOption

class simplekml.**GxOption**(*name=None*, *enabled=False*)
Child element of *simplekml.GxViewerOptions*.

The arguments are the same as the properties.

enabled
Whether the effect must be turned on or off, boolean.

historicalimagery = 'historicalimagery'

name
Name of the effect being applied.

The following strings can be used *simplekml.GxOption.streetview*, *simplekml.GxOption.historicalimagery* or *simplekml.GxOption.sunlight*

streetview = 'streetview'

sunlight = 'sunlight'

GxViewerOptions

class simplekml.**GxViewerOptions**(*gxoptions=None*)
Enables special viewer modes.

The arguments are the same as the properties.

newgxoption(*name*, *enabled=True*)
Creates a *simplekml.GxOption* with name *name* and sets it to *enabled*.

2.3.2 Containers

Document

class simplekml.**Document**(***kwargs*)
A container for features and styles.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
doc = kml.newdocument(name='A Document')
pnt = doc.newpoint()
kml.save("Document.kml")
```

address
Standard address, accepts string.

allcontainers
Returns a list of all the containers that have been attached to this container, and all sub containers.

New in version 1.1.0

allfeatures
Returns a list of all the features that have been attached to this container, and all sub features.

New in version 1.1.0

allgeometries
Returns a list of all the geometries that have been attached to this container, and all sub geometries.

New in version 1.1.0

allstylemaps
Returns a list of all the stylemaps that have been attached to this container, and all sub stylemaps.

New in version 1.1.0

allstyles
Returns a list of all the styles that have been attached to this container, and all sub styles.

New in version 1.1.0

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

containers
Returns a list of all the containers that have been attached to this container.

New in version 1.1.0

description
Description shown in the information balloon, accepts string.

extendeddata
Extra data for the feature.

features
Returns a list of all the features that have been attached to this container.

New in version 1.1.0

geometries
Returns a list of all the geometries that have been attached to this container.

New in version 1.1.0

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
Id number of feature, read-only.

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

newdocument(***kwargs*)
Creates a new *simplekml.Folder* and attaches it to this KML document.

Arguments are the same as *simplekml.Folder*

Returns:

- an instance of *simplekml.Folder* class.

newfolder(***kwargs*)
Creates a new *simplekml.Folder* and attaches it to this KML document.

Arguments are the same as *simplekml.Folder*

Returns:

- an instance of *simplekml.Folder* class.

newgroundoverlay(***kwargs*)
Creates a new *simplekml.GroundOverlay* and attaches it to this KML document.

Arguments are the same as *simplekml.GroundOverlay*

Returns:

- an instance of *simplekml.GroundOverlay* class.

newgxmultitrack(***kwargs*)
Creates a new *simplekml.GxMultiTrack* and attaches it to this KML document.

Arguments are the same as *simplekml.GxMultiTrack*

Returns:

- an instance of *simplekml.GxMultiTrack* class.

newgxtour(***kwargs*)
Creates a new *simplekml.GxTour* and attaches it to this KML document.

Arguments are the same as *simplekml.GxTour*

Returns:

- an instance of *simplekml.NetworkLink* class.

newgxtrack(***kwargs*)
Creates a new *simplekml.GxTrack* and attaches it to this KML document.

Arguments are the same as *simplekml.GxTrack*

Returns:

- an instance of *simplekml.GxTrack* class.

newlinestring(***kwargs*)
Creates a new *simplekml.LineString* and attaches it to this KML document.

Arguments are the same as *simplekml.LineString*

Returns:

- an instance of *simplekml.LineString* class.

newmodel(***kwargs*)
Creates a new *simplekml.Model* and attaches it to this KML document.

Arguments are the same as *simplekml.Model*

Returns:

- an instance of *simplekml.Model* class.

newmultigeometry(***kwargs*)
Creates a new *simplekml.MultiGeometry* and attaches it to this KML document.

Arguments are the same as *simplekml.MultiGeometry*

Returns:

- an instance of *simplekml.MultiGeometry* class.

newnetworklink(***kwargs*)
Creates a new *simplekml.NetworkLink* and attaches it to this KML document.

Arguments are the same as *simplekml.NetworkLink*

Returns:

- an instance of *simplekml.NetworkLink* class.

newphotooverlay(***kwargs*)
Creates a new *simplekml.PhotoOverlay* and attaches it to this KML document.

Arguments are the same as *simplekml.PhotoOverlay*

Returns:

- an instance of *simplekml.PhotoOverlay* class.

newpoint(***kwargs*)
Creates a new *simplekml.Point* and attaches it to this KML document.

Arguments are the same as *simplekml.Point*

Returns:

- an instance of *simplekml.Point* class.

newpolygon(***kwargs*)
Creates a new *simplekml.Polygon* and attaches it to this KML document.

Arguments are the same as *simplekml.Polygon*

Returns:

- an instance of *simplekml.Polygon* class.

newschema(***kwargs*)
Creates a new *simplekml.Schema* and attaches it to this KML document.

Arguments are the same as *simplekml.Schema*

Returns:

- an instance of *simplekml.Schema* class.

newscreenoverlay(***kwargs*)
Creates a new *simplekml.ScreenOverlay* and attaches it to this KML document.

Arguments are the same as *simplekml.ScreenOverlay*

Returns:

- an instance of *simplekml.ScreenOverlay* class.

open
Whether open or closed in Places panel, accepts int 0 or 1.

phonenumber
Phone number used by Google Maps mobile, accepts string.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

stylemaps
Returns a list of all the stylemaps that have been attached to this container.

New in version 1.1.0

styles
Returns a list of all the styles that have been attached to this container.

New in version 1.1.0

styleurl
Reference to the current styleurl or the feature, accepts string.

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

Note: There seems to be a bug in Google Earth where the inclusion of the namespace xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0" seems to break some other elements of the KML such as touring (a tour will not play). If xaladdressdetails is used the above namespace will be added to the KML and will possibly break other elements. Use with caution.

Folder

class simplekml.**Folder**(***kwargs*)
A container for features that act like a folder.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
fol = kml.newfolder(name='A Folder')
pnt = fol.newpoint()
kml.save("Folder.kml")
```

address
Standard address, accepts string.

allcontainers
Returns a list of all the containers that have been attached to this container, and all sub containers.

New in version 1.1.0

allfeatures
Returns a list of all the features that have been attached to this container, and all sub features.

New in version 1.1.0

allgeometries
Returns a list of all the geometries that have been attached to this container, and all sub geometries.

New in version 1.1.0

allstylemaps
Returns a list of all the stylemaps that have been attached to this container, and all sub stylemaps.

New in version 1.1.0

allstyles
Returns a list of all the styles that have been attached to this container, and all sub styles.

New in version 1.1.0

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

containers
Returns a list of all the containers that have been attached to this container.

New in version 1.1.0

description
Description shown in the information balloon, accepts string.

extendeddata
Extra data for the feature.

features
Returns a list of all the features that have been attached to this container.

New in version 1.1.0

geometries
Returns a list of all the geometries that have been attached to this container.

New in version 1.1.0

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
Id number of feature, read-only.

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

newdocument(***kwargs*)
Creates a new *simplekml.Folder* and attaches it to this KML document.

Arguments are the same as *simplekml.Folder*

Returns:

- an instance of *simplekml.Folder* class.

newfolder(***kwargs*)
Creates a new *simplekml.Folder* and attaches it to this KML document.

Arguments are the same as *simplekml.Folder*

Returns:

- an instance of *simplekml.Folder* class.

newgroundoverlay(***kwargs*)
Creates a new *simplekml.GroundOverlay* and attaches it to this KML document.

Arguments are the same as *simplekml.GroundOverlay*

Returns:

- an instance of *simplekml.GroundOverlay* class.

newgxmultitrack(***kwargs*)
Creates a new *simplekml.GxMultiTrack* and attaches it to this KML document.

Arguments are the same as *simplekml.GxMultiTrack*

Returns:

- an instance of *simplekml.GxMultiTrack* class.

newgxtour(***kwargs*)
Creates a new *simplekml.GxTour* and attaches it to this KML document.

Arguments are the same as *simplekml.GxTour*

Returns:

- an instance of *simplekml.NetworkLink* class.

newgxtrack(***kwargs*)
Creates a new *simplekml.GxTrack* and attaches it to this KML document.

Arguments are the same as *simplekml.GxTrack*

Returns:

- an instance of *simplekml.GxTrack* class.

newlinestring(***kwargs*)
Creates a new *simplekml.LineString* and attaches it to this KML document.

Arguments are the same as *simplekml.LineString*

Returns:

- an instance of *simplekml.LineString* class.

newmodel(***kwargs*)
Creates a new *simplekml.Model* and attaches it to this KML document.

Arguments are the same as *simplekml.Model*

Returns:

- an instance of *simplekml.Model* class.

newmultigeometry(***kwargs*)
Creates a new *simplekml.MultiGeometry* and attaches it to this KML document.

Arguments are the same as *simplekml.MultiGeometry*

Returns:

- an instance of *simplekml.MultiGeometry* class.

newnetworklink(***kwargs*)
Creates a new *simplekml.NetworkLink* and attaches it to this KML document.

Arguments are the same as *simplekml.NetworkLink*

Returns:

- an instance of *simplekml.NetworkLink* class.

newphotooverlay(***kwargs*)
Creates a new *simplekml.PhotoOverlay* and attaches it to this KML document.

Arguments are the same as *simplekml.PhotoOverlay*

Returns:

- an instance of *simplekml.PhotoOverlay* class.

newpoint(***kwargs*)
Creates a new *simplekml.Point* and attaches it to this KML document.

Arguments are the same as *simplekml.Point*

Returns:

- an instance of *simplekml.Point* class.

newpolygon(***kwargs*)
Creates a new *simplekml.Polygon* and attaches it to this KML document.

Arguments are the same as *simplekml.Polygon*

Returns:

- an instance of *simplekml.Polygon* class.

newscreenoverlay(***kwargs*)
Creates a new *simplekml.ScreenOverlay* and attaches it to this KML document.

Arguments are the same as *simplekml.ScreenOverlay*

Returns:

- an instance of *simplekml.ScreenOverlay* class.

open
Whether open or closed in Places panel, accepts int 0 or 1.

phonenumber
Phone number used by Google Maps mobile, accepts string.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

stylemaps
Returns a list of all the stylemaps that have been attached to this container.

New in version 1.1.0

styles
Returns a list of all the styles that have been attached to this container.

New in version 1.1.0

styleurl
Reference to the current styleurl or the feature, accepts string.

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

Note: There seems to be a bug in Google Earth where the inclusion of the namespace xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0" seems to break some other elements of the KML such as touring (a tour will not play). If xaladdressdetails is used the above namespace will be added to the KML and will possibly break other elements. Use with caution.

NetworkLink

class simplekml.**NetworkLink**(*refreshvisibility=None*, *flytoview=None*, *link=None*, ***kwargs*)
References a KML file or KMZ archive on a local or remote network.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
netlink = kml.newnetworklink(name="Network Link")
netlink.link.href = "http://simplekml.googlecode.com/hg/samples/samples.kml"
netlink.link.viewrefreshmode = simplekml.ViewRefreshMode.onrequest
kml.save("NetworkLink.kml")
```

address
Standard address, accepts string.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

description
Description shown in the information balloon, accepts string.

extendeddata
Extra data for the feature.

flytoview
A value of 1 causes Google Earth to fly to the view of the AbstractView.

Accepts int (0 or 1).

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
Id number of feature, read-only.

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

link
A *simplekml.Link* class instance, accepts *simplekml.Link*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

open
Whether open or closed in Places panel, accepts int 0 or 1.

phonenumber
Phone number used by Google Maps mobile, accepts string.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

refreshvisibility
How the visibility is affected by a refresh

A value of 0 leaves the visibility of features within the control of the Google Earth user. Set the value to 1 to reset the visibility of features each time the NetworkLink is refreshed, accepts int (0 or 1).

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

styleurl
Reference to the current styleurl or the feature, accepts string.

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

Note: There seems to be a bug in Google Earth where the inclusion of the namespace xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0" seems to break some other elements of the KML such as touring (a tour will not play). If xaladdressdetails is used the above namespace will be added to the KML and will possibly break other elements. Use with caution.

2.3.3 Constants

AltitudeMode

class simplekml.**AltitudeMode**
AltitudeMode constants.

absolute = 'absolute'

clamptoground = 'clampToGround'

relativetoground = 'relativeToGround'

GxAltitudeMode

class simplekml.**GxAltitudeMode**
gx:AltitudeMode constants.

clampToSeaFloor = 'clampToSeaFloor '

relativetoseafloor = 'relativeToSeaFloor '

Color

class simplekml.**Color**
Color constants (HTML and CSS) and converters.

Constants: Same as HTML and CSS standard colors. All constants are lowercase.

Class methods:

- *simplekml.Color.rgb()* - convert RGB to KML HEX
- *simplekml.Color.hex()* - convert HEX to KML HEX
- *simplekml.Color.hexa()* - convert HEX (with alpha) to KML HEX
- *simplekml.Color.changealpha()* - change KML HEX alpha value with a HEX
- *simplekml.Color.changealphaint()* - change KML HEX alpha value with an int

aliceblue = 'fffff8f0'

antiquewhite = 'ffd7ebfa'

aqua = 'ffffff00'

aquamarine = 'ffd4ff7f'

azure = 'fffffff0'

beige = 'ffdcf5f5'

bisque = 'ffc4e4ff'

black = 'ff000000'

blanchedalmond = 'ffcdebff'

blue = 'ffff0000'

blueviolet = 'ffe22b8a'

brown = 'ff2a2aa5'

burlywood = 'ff87b8de'

cadetblue = 'ffa09e5f'

classmethod changealpha(*alpha*, *gehex*)
Changes the alpha value of the given Google Earth hex value to the given aplha hex value.

Args:

- alpha: aplha hex string
- gehex: Google Earth hex string

classmethod changealphaint(*alpha*, *gehex*)
Changes the alpha value of the given Google Earth hex value to the given alpha integer value.

Args:

- alpha: aplha integer (0 - 255)
- gehex: Google Earth hex string

```
chartreuse = 'ff00ff7f'
chocolate = 'ff1e69d2'
coral = 'ff507fff'
cornflowerblue = 'ffed9564'
cornsilk = 'ffdcf8ff'
crimson = 'ff3c14dc'
cyan = 'ffffff00'
darkblue = 'ff8b0000'
darkcyan = 'ff8b8b00'
darkgoldenrod = 'ff0b86b8'
darkgray = 'ffa9a9a9'
darkgreen = 'ff006400'
darkgrey = 'ffa9a9a9'
darkkhaki = 'ff6bb7bd'
darkmagenta = 'ff8b008b'
darkolivegreen = 'ff2f6b55'
darkorange = 'ff008cff'
darkorchid = 'ffcc3299'
darkred = 'ff00008b'
darksalmon = 'ff7a96e9'
darkseagreen = 'ff8fbc8f'
darkslateblue = 'ff8b3d48'
darkslategray = 'ff4f4f2f'
darkslategrey = 'ff4f4f2f'
darkturquoise = 'ffd1ce00'
darkviolet = 'ffd30094'
deeppink = 'ff9314ff'
deepskyblue = 'ffffbf00'
dimgray = 'ff696969'
dimgrey = 'ff696969'
dodgerblue = 'ffff901e'
firebrick = 'ff2222b2'
floralwhite = 'fff0faff'
```

forestgreen = 'ff228b22'

fuchsia = 'ffff00ff'

gainsboro = 'ffdcdcdc'

ghostwhite = 'fffff8f8'

gold = 'ff00d7ff'

goldenrod = 'ff20a5da'

gray = 'ff808080'

green = 'ff008000'

greenyellow = 'ff2fffad'

grey = 'ff808080'

classmethod hex(*hstr*)
Convert hex (without alpha) to KML hex value.

Args:

- hstr: hex string without alpha value

classmethod hexa(*hstr*)
Convert hex (with alpha) to KML hex value.

Args:

- hstr: hex string with alpha value

honeydew = 'fff0fff0'

hotpink = 'ffb469ff'

indianred = 'ff5c5ccd'

indigo = 'ff82004b'

ivory = 'fff0ffff'

khaki = 'ff8ce6f0'

lavender = 'fffae6e6'

lavenderblush = 'fff5f0ff'

lawngreen = 'ff00fc7c'

lemonchiffon = 'ffcdfaff'

lightblue = 'ffe6d8ad'

lightcoral = 'ff8080f0'

lightcyan = 'ffffffe0'

lightgoldenrodyellow = 'ffd2fafa'

lightgray = 'ffd3d3d3'

lightgreen = 'ff90ee90'

lightgrey = 'ffd3d3d3'

lightpink = 'ffc1b6ff'

lightsalmon = 'ff7aa0ff'

```
    lightseagreen = 'ffaab220'
    lightskyblue = 'fffface87'
    lightslategray = 'ff998877'
    lightslategrey = 'ff998877'
    lightsteelblue = 'ffdec4b0'
    lightyellow = 'ffe0ffff'
    lime = 'ff00ff00'
    limegreen = 'ff32cd32'
    linen = 'ffe6f0fa'
    magenta = 'ffff00ff'
    maroon = 'ff000080'
    mediumaquamarine = 'ffaacd66'
    mediumblue = 'ffcd0000'
    mediumorchid = 'ffd355ba'
    mediumpurple = 'ffd87093'
    mediumseagreen = 'ff71b33c'
    mediumslateblue = 'ffee687b'
    mediumspringgreen = 'ff9afa00'
    mediumturquoise = 'ffccd148'
    mediumvioletred = 'ff8515c7'
    midnightblue = 'ff701919'
    mintcream = 'fffafff5'
    mistyrose = 'ffe1e4ff'
    moccasin = 'ffb5e4ff'
    navajowhite = 'ffaddeff'
    navy = 'ff800000'
    oldlace = 'ffe6f5fd'
    olive = 'ff008080'
    olivedrab = 'ff238e6b'
    orange = 'ff00a5ff'
    orangered = 'ff0045ff'
    orchid = 'ffd670da'
    palegoldenrod = 'ffaae8ee'
    palegreen = 'ff98fb98'
    paleturquoise = 'ffeeeeaf'
    palevioletred = 'ff9370d8'
```

papayawhip = 'ffd5efff'

peachpuff = 'ffb9daff'

peru = 'ff3f85cd'

pink = 'ffcbc0ff'

plum = 'ffdda0dd'

powderblue = 'ffe6e0b0'

purple = 'ff800080'

red = 'ff0000ff'

classmethod rgb(*r*, *g*, *b*, *a=255*)

Convert rgba to KML hex value.

Args:

- r: int between 0 - 255 representing red
- g: int between 0 - 255 representing green
- b: int between 0 - 255 representing blue
- a: int between 0 - 255 representing alpha (default 255)

rosybrown = 'ff8f8fbc'

royalblue = 'ffe16941'

saddlebrown = 'ff13458b'

salmon = 'ff7280fa'

sandybrown = 'ff60a4f4'

seagreen = 'ff578b2e'

seashell = 'ffeef5ff'

sienna = 'ff2d52a0'

silver = 'ffc0c0c0'

skyblue = 'ffebce87'

slateblue = 'ffcd5a6a'

slategray = 'ff908070'

slategrey = 'ff908070'

snow = 'fffafaff'

springgreen = 'ff7fff00'

steelblue = 'ffb48246'

tan = 'ff8cb4d2'

teal = 'ff808000'

thistle = 'ffd8bfd8'

tomato = 'ff4763ff'

turquoise = 'ffd0e040'

violet = 'ffee82ee'

wheat = 'ffb3def5'

white = 'ffffffff'

whitesmoke = 'fff5f5f5'

yellow = 'ff00ffff'

yellowgreen = 'ff32cd9a'

ColorMode

class simplekml.ColorMode
ColorMode constants.

normal = 'normal'

random = 'random'

DisplayMode

class simplekml.DisplayMode
DisplayMode constants.

default = 'default'

hide = 'hide'

GridOrigin

class simplekml.GridOrigin
GridOrigin constants.

lowerleft = 'lowerLeft'

upperleft = 'upperLeft'

ListItemType

class simplekml.ListItemType
ListItemType constants.

check = 'check'

checkhidechildren = 'checkHideChildren'

checkoffonly = 'checkOffOnly'

radiofolder = 'radioFolder'

RefreshMode

class simplekml.RefreshMode
RefreshMode constants.

onchange = 'onChange'

onexpire = 'onExpire'

oninterval = 'onInterval'

Shape

class simplekml.**Shape**
Shape constants.

circle = 'circle'

rectangle = 'rectangle'

sphere = 'sphere'

State

class simplekml.**State**
State constants.

closed = 'closed'

error = 'error'

fetching0 = 'fetching0'

fetching1 = 'fetching1'

fetching2 = 'fetching2'

open = 'open'

Types

class simplekml.**Types**
Types constants.

bool = 'bool'

double = 'double'

float = 'float'

int = 'int'

short = 'short'

string = 'string'

uint = 'uint'

ushort = 'ushort'

Units

class simplekml.**Units**
Units constants.

fraction = 'fraction'

insetpixels = 'insetPixels'

pixels = 'pixels'

ViewRefreshMode

class simplekml.**ViewRefreshMode**
ViewRefreshMode constants.

never = 'never'

onregion = 'onRegion'

onrequest = 'onRequest'

onstop = 'onStop'

2.3.4 Geometries

Point

class simplekml.**Point**(*extrude=None*, *altitudemode=None*, *gxaltitudemode=None*, ***kwargs*)
A geographic location defined by lon, lat, and altitude.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.coords = [(1.0, 2.0)]
kml.save("Point.kml")
```

Styling a Single Point:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.coords = [(1.0, 2.0)]
pnt.style.labelstyle.color = simplekml.Color.red  # Make the text red
pnt.style.labelstyle.scale = 2  # Make the text twice as big
pnt.style.iconstyle.icon.href = 'http://maps.google.com/mapfiles/kml/shapes/
↪placemark_circle.png'
kml.save("Point Styling.kml")
```

Sharing a Style with many Points (Shared Style):

```
import simplekml
kml = simplekml.Kml()
style = simplekml.Style()
style.labelstyle.color = simplekml.Color.red  # Make the text red
style.labelstyle.scale = 2  # Make the text twice as big
style.iconstyle.icon.href = 'http://maps.google.com/mapfiles/kml/shapes/placemark_
↪circle.png'
for lon in range(2):  # Generate longitude values
    for lat in range(2): # Generate latitude values
        pnt = kml.newpoint(name='Point: {0}{0}'.format(lon,lat))
        pnt.coords = [(lon, lat)]
```

(continues on next page)

(continued from previous page)

```
        pnt.style = style
kml.save("Point Shared Style.kml")
```

address
Standard address, accepts string.

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts *simplekml.AltitudeMode* constants.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

coords
The coordinates of the feature, accepts list of tuples.

A tuple represents a coordinate in the order longitude then latitude. The tuple has the option of specifying a height. If no height is given, it defaults to zero. A point feature has just one point, therefore a list with one tuple is given.

Examples:

- No height: *[(1.0, 1.0), (2.0, 1.0)]*
- Height: *[(1.0, 1.0, 50.0), (2.0, 1.0, 10.0)]*
- Point: *[(1.0, 1.0)]* # longitude, latitude
- Point + height: *[(1.0, 1.0, 100)]* # longitude, latitude, height of 100m

description
Description shown in the information balloon, accepts string.

extendeddata
Short description of the feature, accepts *simplekml.Snippet*

extrude
Connect the Point to the ground, accepts int (0 or 1).

gxaltitudemode
Specifies how the altitude for the Camera is interpreted.

With the addition of being relative to the sea floor. Accepts *simplekml.GxAltitudeMode* constants.

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
The id string (read only).

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

phonenumber
Phone number used by Google Maps mobile, accepts string.

placemark
The placemark that contains this feature, read-only.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

LinearRing

class simplekml.**LinearRing**(*coords=()*, *extrude=None*, *tessellate=None*, *altitudemode=None*, *gxaltitudemode=None*, *gxaltitudeoffset=None*, ***kwargs*)

A closed line string, typically the outer boundary of a *simplekml.Polygon*

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pol = kml.newpolygon()
print(pol.outerboundaryis) # Shows that the outer boundary of a polygon is a␣
↪linear ring
```

(continues on next page)

(continued from previous page)

```
pol.outerboundaryis.coords = [(0.0,0.0), (1.0,1.0), (2.0,2.0)]
kml.save("LinearRing.kml")
```

address
Standard address, accepts string.

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts *simplekml.AltitudeMode* constants.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

coords
The coordinates of the feature, accepts list of tuples.

A tuple represents a coordinate in the order longitude then latitude. The tuple has the option of specifying a height. If no height is given, it defaults to zero. A point feature has just one point, therefore a list with one tuple is given.

Examples:

- No height: *[(1.0, 1.0), (2.0, 1.0)]*
- Height: *[(1.0, 1.0, 50.0), (2.0, 1.0, 10.0)]*
- Point: *[(1.0, 1.0)]* # longitude, latitude
- Point + height: *[(1.0, 1.0, 100)]* # longitude, latitude, height of 100m

description
Description shown in the information balloon, accepts string.

extendeddata
Short description of the feature, accepts *simplekml.Snippet*

extrude
Connect the LinearRing to the ground, accepts int (0 or 1).

gxaltitudemode
Specifies how the altitude for the Camera is interpreted.

With the addition of being relative to the sea floor. Accepts *simplekml.GxAltitudeMode* constants.

gxaltitudeoffset
How much to offsets the LinearRing vertically, accepts int.

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
The id string (read only).

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

phonenumber
Phone number used by Google Maps mobile, accepts string.

placemark
The placemark that contains this feature, read-only.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

tessellate
Allows the LinearRing to follow the terrain, accepts int (0 or 1).

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

LineString

class simplekml.**LineString**(*extrude=None*, *tessellate=None*, *altitudemode=None*, *gxaltitudemode=None*, *gxaltitudeoffset=None*, *gxdraworder=None*, ***kwargs*)
A connected set of line segments.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
ls = kml.newlinestring(name='A LineString')
ls.coords = [(18.333868,-34.038274,10.0), (18.370618,-34.034421,10.0)]
ls.extrude = 1
ls.altitudemode = simplekml.AltitudeMode.relativetoground
kml.save("LineString.kml")
```

Styling:

```
import simplekml
kml = simplekml.Kml()
ls = kml.newlinestring(name='A LineString')
ls.coords = [(18.333868,-34.038274,10.0), (18.370618,-34.034421,10.0)]
ls.extrude = 1
ls.altitudemode = simplekml.AltitudeMode.relativetoground
ls.style.linestyle.width = 5
ls.style.linestyle.color = simplekml.Color.blue
kml.save("LineString Styling.kml")
```

address
Standard address, accepts string.

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts *simplekml.AltitudeMode* constants.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

coords
The coordinates of the feature, accepts list of tuples.

A tuple represents a coordinate in the order longitude then latitude. The tuple has the option of specifying a height. If no height is given, it defaults to zero. A point feature has just one point, therefore a list with one tuple is given.

Examples:

- No height: *[(1.0, 1.0), (2.0, 1.0)]*
- Height: *[(1.0, 1.0, 50.0), (2.0, 1.0, 10.0)]*
- Point: *[(1.0, 1.0)]* # longitude, latitude
- Point + height: *[(1.0, 1.0, 100)]* # longitude, latitude, height of 100m

description
Description shown in the information balloon, accepts string.

extendeddata
Short description of the feature, accepts *simplekml.Snippet*

extrude
Connect the LinearRing to the ground, accepts int (0 or 1).

gxaltitudemode
Specifies how the altitude for the Camera is interpreted.

With the addition of being relative to the sea floor. Accepts *simplekml.GxAltitudeMode* constants.

gxaltitudeoffset
How much to offsets the LinearRing vertically, accepts int.

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

gxdraworder
The order to draw the linestring, accepts int.

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
The id string (read only).

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

phonenumber
Phone number used by Google Maps mobile, accepts string.

placemark
The placemark that contains this feature, read-only.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

tessellate
Allowe the LinearRing to follow the terrain, accepts int (0 or 1).

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

Polygon

class simplekml.**Polygon**(*extrude=None*, *tessellate=None*, *altitudemode=None*, *gxaltitudemode=None*, *outerboundaryis=()*, *innerboundaryis=()*, ***kwargs*)
A Polygon is defined by an outer boundary and/or an inner boundary.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pol = kml.newpolygon(name='A Polygon')
pol.outerboundaryis = [(18.333868,-34.038274), (18.370618,-34.034421),
                       (18.350616,-34.051677),(18.333868,-34.038274)]
pol.innerboundaryis = [(18.347171,-34.040177), (18.355741,-34.039730),
                       (18.350467,-34.048388),(18.347171,-34.040177)]
kml.save("Polygon.kml")
```

Styling:

```
import simplekml
kml = simplekml.Kml()
pol = kml.newpolygon(name='A Polygon')
pol.outerboundaryis = [(18.333868,-34.038274), (18.370618,-34.034421),
                       (18.350616,-34.051677),(18.333868,-34.038274)]
pol.innerboundaryis = [(18.347171,-34.040177), (18.355741,-34.039730),
                       (18.350467,-34.048388),(18.347171,-34.040177)]
pol.style.linestyle.color = simplekml.Color.green
pol.style.linestyle.width = 5
pol.style.polystyle.color = simplekml.Color.changealphaint(100, simplekml.Color.
↪green)
kml.save("Polygon Styling.kml")
```

address
Standard address, accepts string.

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts *simplekml.AltitudeMode* constants.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

description
Description shown in the information balloon, accepts string.

extendeddata
Short description of the feature, accepts *simplekml.Snippet*

extrude
Connect the LinearRing to the ground, accepts int (0 or 1).

gxaltitudemode
Specifies how the altitude for the Camera is interpreted.

With the addition of being relative to the sea floor. Accepts *simplekml.GxAltitudeMode* constants.

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
The id string (read only).

innerboundaryis
The inner boundaries.

Accepts list of list of tuples of floats for multiple boundaries, or a list of tuples of floats for a single boundary.

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

outerboundaryis
The outer boundary, accepts a list of tuples of floats.

phonenumber
Phone number used by Google Maps mobile, accepts string.

placemark
The placemark that contains this feature, read-only.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

tessellate
Allows the Polygon to follow the terrain, accepts int (0 or 1).

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

MultiGeometry

class simplekml.**MultiGeometry**(*geometries=()*, ***kwargs*)
MultiGeometry is a collection of simple features (Points, LineStrings, etc).

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
multipnt = kml.newmultigeometry(name="MultiPoint")
for lon in range(2):  # Generate longitude values
    for lat in range(2): # Generate latitude values
        multipnt.newpoint(coords=[(lon, lat)])
kml.save("MultiGeometry.kml")
```

Styling:

```
import simplekml
kml = simplekml.Kml()
multipnt = kml.newmultigeometry(name="MultiPoint")
multipnt.style.labelstyle.scale = 0  # Remove the labels from all the points
multipnt.style.iconstyle.color = simplekml.Color.red
for lon in range(2):  # Generate longitude values
    for lat in range(2): # Generate latitude values
        multipnt.newpoint(coords=[(lon, lat)])
kml.save("MultiGeometry Styling.kml")
```

address
Standard address, accepts string.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

description
Description shown in the information balloon, accepts string.

extendeddata
Short description of the feature, accepts *simplekml.Snippet*

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
The id string (read only).

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

newgroundoverlay(***kwargs*)
Creates a new *simplekml.GroundOverlay* and attaches it to this MultiGeometry.

The arguments are the same as *simplekml.GroundOverlay*

Returns:

- an instance of *simplekml.GroundOverlay*

newlinestring(***kwargs*)
Creates a new *simplekml.LineString* and attaches it to this MultiGeometry.

The arguments are the same as *simplekml.LineString*

Returns:

- an instance of *simplekml.LineString*

newmodel(***kwargs*)
Creates a new *simplekml.Model* and attaches it to this MultiGeometry.

The arguments are the same as *simplekml.Model*

Returns:

- an instance of *simplekml.Model*

newphotooverlay(***kwargs*)
Creates a new *simplekml.PhotoOverlay* and attaches it to this MultiGeometry.

The arguments are the same as *simplekml.PhotoOverlay*

Returns:

- an instance of *simplekml.PhotoOverlay*

newpoint(***kwargs*)
Creates a new *simplekml.Point* and attaches it to this MultiGeometry.

The arguments are the same as *simplekml.Point*

Returns:

- an instance of *simplekml.Point*

newpolygon(***kwargs*)
Creates a new *simplekml.Polygon* and attaches it to this MultiGeometry.

The arguments are the same as *simplekml.Polygon*

Returns:

- an instance of *simplekml.Polygon*

newscreenoverlay(***kwargs*)
Creates a new *simplekml.ScreenOverlay* and attaches it to this MultiGeometry.

The arguments are the same as *simplekml.ScreenOverlay*

Returns:

- an instance of *simplekml.ScreenOverlay*

phonenumber
Phone number used by Google Maps mobile, accepts string.

placemark
The placemark that contains this feature, read-only.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

Model

class simplekml.**Model**(*altitudemode=None*, *gxaltitudemode=None*, *location=None*, *orientation=None*, *scale=None*, *link=None*, *resourcemap=None*, ***kwargs*)
A 3D object described in a COLLADA file.

Arguments are the same as the properties.

address
Standard address, accepts string.

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts *simplekml.AltitudeMode* constants.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

description
Description shown in the information balloon, accepts string.

extendeddata
Short description of the feature, accepts *simplekml.Snippet*

gxaltitudemode
Specifies how the altitude for the Camera is interpreted.

With the addition of being relative to the sea floor. Accepts *simplekml.GxAltitudeMode* constants.

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
The id string (read only).

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

link
"A *simplekml.Link* class instance, accepts *simplekml.Link*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

location
Position of the origin of the model, accepts *simplekml.Location*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

orientation
The rotation on the model, accepts *simplekml.Orientation*

phonenumber
Phone number used by Google Maps mobile, accepts string.

placemark
The placemark that contains this feature, read-only.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

resourcemap
Used for mapping textures, accepts *simplekml.ResourceMap*

scale
"The scale of the model, accepts *simplekml.Scale*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

GxTrack

class simplekml.**GxTrack**(*extrude=None*, *altitudemode=None*, *gxaltitudemode=None*, *model=None*, ***kwargs*)

A track describes how an object moves through the world over a given time period.

Arguments are the same as the properties.

Usage:

```
# This is a recreation of the example found in the KML Reference:
# http://code.google.com/apis/kml/documentation/kmlreference.html#gxtrack

import os
from simplekml import Kml, Snippet, Types

# Data for the track
when = ["2010-05-28T02:02:09Z",
    "2010-05-28T02:02:35Z",
    "2010-05-28T02:02:44Z",
    "2010-05-28T02:02:53Z",
    "2010-05-28T02:02:54Z",
    "2010-05-28T02:02:55Z",
    "2010-05-28T02:02:56Z"]

coord = [(-122.207881,37.371915,156.000000),
    (-122.205712,37.373288,152.000000),
    (-122.204678,37.373939,147.000000),
    (-122.203572,37.374630,142.199997),
    (-122.203451,37.374706,141.800003),
    (-122.203329,37.374780,141.199997),
    (-122.203207,37.374857,140.199997)]

cadence = [86, 103, 108, 113, 113, 113, 113]
heartrate = [181, 177, 175, 173, 173, 173, 173]
power = [327.0, 177.0, 179.0, 162.0, 166.0, 177.0, 183.0]

# Create the KML document
kml = Kml(name="Tracks", open=1)
doc = kml.newdocument(name='GPS device', snippet=Snippet('Created Wed Jun 2␣
↪15:33:39 2010'))
doc.lookat.gxtimespan.begin = '2010-05-28T02:02:09Z'
doc.lookat.gxtimespan.end = '2010-05-28T02:02:56Z'
doc.lookat.longitude = -122.205544
doc.lookat.latitude = 37.373386
doc.lookat.range = 1300.000000

# Create a folder
fol = doc.newfolder(name='Tracks')

# Create a schema for extended data: heart rate, cadence and power
schema = kml.newschema()
schema.newgxsimplearrayfield(name='heartrate', type=Types.int, displayname='Heart␣
↪Rate')
schema.newgxsimplearrayfield(name='cadence', type=Types.int, displayname='Cadence
↪')
schema.newgxsimplearrayfield(name='power', type=Types.float, displayname='Power')

# Create a new track in the folder
trk = fol.newgxtrack(name='2010-05-28T01:16:35.000Z')

# Apply the above schema to this track
trk.extendeddata.schemadata.schemaurl = schema.id

# Add all the information to the track
trk.newwhen(when) # Each item in the give nlist will become a new <when> tag
trk.newgxcoord(coord) # Ditto
```

(continues on next page)

(continued from previous page)

```
trk.extendeddata.schemadata.newgxsimplearraydata('heartrate', heartrate) # Ditto
trk.extendeddata.schemadata.newgxsimplearraydata('cadence', cadence) # Ditto
trk.extendeddata.schemadata.newgxsimplearraydata('power', power) # Ditto

# Styling
trk.stylemap.normalstyle.iconstyle.icon.href = 'http://earth.google.com/images/
↪kml-icons/track-directional/track-0.png'
trk.stylemap.normalstyle.linestyle.color = '99ffac59'
trk.stylemap.normalstyle.linestyle.width = 6
trk.stylemap.highlightstyle.iconstyle.icon.href = 'http://earth.google.com/images/
↪kml-icons/track-directional/track-0.png'
trk.stylemap.highlightstyle.iconstyle.scale = 1.2
trk.stylemap.highlightstyle.linestyle.color = '99ffac59'
trk.stylemap.highlightstyle.linestyle.width = 8

# Save the kml to file
kml.save("GxTrack.kml")
```

address
Standard address, accepts string.

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts *simplekml.AltitudeMode* constants.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

description
Description shown in the information balloon, accepts string.

extendeddata
Extra data for the feature.

extrude
Connect the GxTrack to the ground, accepts int (0 or 1).

gxaltitudemode
Specifies how the altitude for the Camera is interpreted.

With the addition of being relative to the sea floor. Accepts *simplekml.GxAltitudeMode* constants.

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
The id string (read only).

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

model
A model to use on the track, accepts *simplekml.Model*

New in version 1.2.1

name
Name of placemark, accepts string.

newdata(*gxcoord*, *when*, *angle=None*)
Creates a new gxcoord, when time and angle (if provided).

This is a convenience method for calling newwhen, newgxcoord and newangle. when and gxcoord are required, angle is optional.

newgxangle(*angle*)
Creates a new gx:angle, accepts float or list of floats.

If one float is given a single angle entry is created, but if a list of floats is given, a angle entry is created for each float in the list.

newgxcoord(*coord*)
Creates a gx:coord, accepts list of one tuples.

A gxcoord entry is created for every tuple in the list.

newwhen(*when*)
Creates a new when time, accepts string or list of string.

If one string is given a single when entry is created, but if a list of strings is given, a when entry is created for each string in the list.

phonenumber
Phone number used by Google Maps mobile, accepts string.

placemark
The placemark that contains this feature, read-only.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

GxMultiTrack

class simplekml.**GxMultiTrack**(*tracks=()*, *gxinterpolate=None*, ***kwargs*)
A container for grouping gx:tracks.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
multitrack = kml.newgxmultitrack()
track1 = multitrack.newgxtrack(name="track1")
track2 = multitrack.newgxtrack(name="track2")
kml.save("GxMultiTrack.kml")
```

address
Standard address, accepts string.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

description
Description shown in the information balloon, accepts string.

extendeddata
Short description of the feature, accepts *simplekml.Snippet*

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
The id string (read only).

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

newgxtrack(***kwargs*)
Creates a new *simplekml.GxTrack* and attaches it to this mutlitrack.

Returns an instance of *simplekml.GxTrack* class.

Args:

- Same as *simplekml.GxTrack*, except arguments that are not applicable in a multitrack grouping will be ignored, such as name, visibility, open, etc.

phonenumber
Phone number used by Google Maps mobile, accepts string.

placemark
The placemark that contains this feature, read-only.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

2.3.5 Kml

Kml

class simplekml.**Kml**(***kwargs*)
The main class that represents a KML file.

This class represents a KML file, and the compilation of the KML file will be done through this class. The base feature is a document, all arguments passed to the class on creation are the same as that of a *simplekml.Document*. To change any properties after creation you can do so through the *simplekml.Kml.document* property (eg. *kml.document.name = "Test"*). For a description of what the arguments mean see the KML reference documentation published by Google: http://code.google.com/apis/kml/documentation/kmlreference.html

Simple Example:

```
from simplekml import Kml
kml = Kml(name='KmlUsage')
kml.newpoint(name="Kirstenbosch", coords=[(18.432314,-33.988862)])  # A simple␣
↪Point
kml.save("KmlClass.kml")  # Saving
kml.savekmz("KmlClass.kmz", format=False)  # Saving as KMZ
print(kml.kml())  # Printing out the kml to screen
```

addfile(*path*)

Adds an file to a KMZ and returns the path contained inside of the KMZ (files/...)

This is useful for including images in a KMZ that are referenced from description balloons, as these files are not automatically included in a KMZ.

Usage:

```
import simplekml
kml = simplekml.Kml()
path = kml.addfile("a/path/to/somefile.file")
pnt = pnt.newpoint()
pnt.description = '<img src="' + path +'" alt="picture" width="400" height=
↪"300" align="left" />'
```

New in version 1.2.0

allcontainers

Returns a list of all the containers that have been attached to the top level document, and all sub containers.

New in version 1.1.0

allfeatures

Returns a list of all the features that have been attached to the top level document, and all sub features.

New in version 1.1.0

allgeometries

Returns a list of all the geometries that have been attached to the top level document, and all sub geometries.

New in version 1.1.0

allstylemaps

Returns a list of all the stylemaps that have been attached to the top level document, and all sub stylemaps.

New in version 1.1.0

allstyles

Returns a list of all the styles that have been attached to the top level document, and all sub styles.

New in version 1.1.0

containers

Returns a list of all the containers that have been attached to to the top level document.

New in version 1.1.0

document
The top level item in the kml document.

0 or 1 top level document is required for a kml document, the default is an instance of *simplekml.Document*. This property can be set to an instance of *simplekml.Document* or *simplekml.Folder* or to remove it completely set it to None

Example:

```
import simplekml
kml = simplekml.Kml()
kml.document = simplekml.Folder(name = "Top Level Folder")
kml.save('Document Replacement.kml')
```

features
Returns a list of all the features that have been attached to the top level document.

geometries
Returns a list of all the geometries that have been attached to the top level document.

New in version 1.1.0

hint
Assign a hint attribute to the KML tag.

Possible values to use are:

- target=moon
- target=sky
- target=mars

Usage:

```
from simplekml import Kml
kml = Kml()
kml.hint = 'target=moon'
print(kml.kml())
```

Result:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml hint="target=moon" xmlns="http://www.opengis.net/kml/2.2" xmlns:gx=
↪"http://www.google.com/kml/ext/2.2">
    <Document id="feat_1"/>
</kml>
```

New in version 1.1.0

kml (*format=True*)
Returns the kml as a string or "prettyprinted" if *format* = *True*.

Note: Setting *format* = *False* will produce smaller files, as well as decrease the memory required while processing the kml.

PrettyPrinted Example (default):

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.coords = [(1.0, 2.0)]
print(kml.kml())
```

PrettyPrinted Result:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/
↪kml/ext/2.2">
    <Document id="feat_1">
        <Placemark id="feat_2">
            <name>A Point</name>
            <Point id="geom_0">
                <coordinates>1.0,2.0,0.0</coordinates>
            </Point>
        </Placemark>
    </Document>
</kml>
```

Single Line Example:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.coords = [(1.0, 2.0)]
print(kml.kml(False))
```

Single Line Result:

```
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/
↪kml/ext/2.2"><Document id="feat_1"><Placemark id="feat_2"><name>A Point</
↪name><Point id="geom_0"><coordinates>1.0,2.0,0.0</coordinates></Point></
↪Placemark></Document></kml>
```

networklinkcontrol
Accesses/Creates the *simplekml.NetworkLinkControl*.

See *simplekml.NetworkLinkControl* for usage example.

New in version 1.1.1

newdocument(***kwargs*)
Creates a new *simplekml.Document*.

The document is attached to this KML document. The arguments are the same as for *simplekml.Document*. See *simplekml.Document* for usage.

newfolder(***kwargs*)
Creates a new *simplekml.Folder*.

The folder is attached to this KML document. The arguments are the same as those for *simplekml.Folder* See *simplekml.Folder* for usage.

newgroundoverlay(***kwargs*)
Creates a new *simplekml.GroundOverlay*.

The groundoverlay is attached to this KML document. The arguments are the same as those for *simplekml.GroundOverlay*. See *simplekml.GroundOverlay* for usage.

newgxmultitrack(***kwargs*)
Creates a new *simplekml.GxMultiTrack*.

The gxmultitrack is attached to this KML document. The arguments are the same as those for *simplekml.GxMultiTrack*. See *simplekml.GxMultiTrack* for usage.

newgxtour(***kwargs*)
Creates a new *simplekml.GxTour*.

The tour is attached to this KML document. The arguments are the same as those for *simplekml.GxTour* See *simplekml.GxTour* for usage.

newgxtrack(***kwargs*)
Creates a new *simplekml.GxTrack*.

The gxtrack is attached to this KML document. The arguments are the same as those for *simplekml.GxTrack* See *simplekml.GxTrack* for usage.

newlinestring(***kwargs*)
Creates a new *simplekml.LineString*.

The linestring is attached to this KML document. The arguments are the same as for *simplekml.LineString* See *simplekml.LineString* for usage.

newmodel(***kwargs*)
Creates a new *simplekml.Model*.

The model is attached to this KML document. The arguments are the same as those for *simplekml.Model*

newmultigeometry(***kwargs*)
Creates a new *simplekml.MultiGeometry*.

The multigeometry is attached to this KML document. The arguments are the same as for *simplekml.MultiGeometry*. See *simplekml.MultiGeometry* for usage.

newnetworklink(***kwargs*)
Creates a new *simplekml.NetworkLink*.

The networklink is attached to this KML document. The arguments are the same as those for *simplekml.NetworkLink*. See *simplekml.NetworkLink* for usage.

newphotooverlay(***kwargs*)
Creates a new *simplekml.PhotoOverlay*.

The photooverlay is attached to this KML document. The arguments are the same as those for *simplekml.PhotoOverlay*. See *simplekml.PhotoOverlay* for usage.

newpoint(***kwargs*)
Creates a new *simplekml.Point*.

The point is attached to this KML document. The arguments are the same as those for *simplekml.Point* See *simplekml.Point* for usage.

newpolygon(***kwargs*)
Creates a new *simplekml.Polygon*.

The polygon is attached to this KML document. The arguments are the same as those for *simplekml.Polygon* See *simplekml.Polygon* for usage.

newschema(***kwargs*)
Creates a new *simplekml.Schema*.

The schem is attached to this KML document. The arguments are the same as those for *simplekml.Schema*

newscreenoverlay (***kwargs*)
Creates a new *simplekml.ScreenOverlay*.

The screenoverlay is attached to this KML document. The arguments are the same as those for *simplekml.ScreenOverlay*. See *simplekml.ScreenOverlay* for usage.

parsetext (*parse=True*)
Sets the behavior of how text tags are parsed.

If True the values of the text tags (<name>, <description> and <text>) are escaped, so that the values are rendered properly. If False, the values are left as is. In both cases the CDATA element is left unchanged.

Changed in version 1.1.0

static resetidcounter ()
Resets the id counter so that ids count from 0.

New in version 1.3.1

save (*path*, *format=True*)
Save the kml to the given file supplied by *path*.

The KML is saved to a file in one long string if *format=False* else it gets saved "prettyprinted" (as formatted xml). This works the same as *simplekml.Kml.kml()*

Note: Setting *format = False* will produce smaller files, as well as decrease the memory required while processing the kml.

Usage:

```
import simplekml
kml = simplekml.Kml()
kml.save("Saving.kml")
#kml.save("Saving.kml", False)  # or this
```

savekmz (*path*, *format=True*)
Save the kml as a kmz to the given file supplied by *path*.

The KML is saved to a file in a long string if *format=False* else it gets saved "prettyprinted". This works the same as *simplekml.Kml.kml()*

Usage:

```
import simplekml
kml = simplekml.Kml()
kml.savekmz("Saving.kml")
#kml.savekmz("Saving.kml", False)  # or this
```

stylemaps
Returns a list of all the stylemaps that have been attached to the top level document.

New in version 1.1.0

styles
Returns a list of all the styles that have been attached to the top level document.

New in version 1.1.0

NetworkLinkControl

class simplekml.**NetworkLinkControl**(*minrefreshperiod=None*, *maxsessionlength=None*, *cookie=None*, *message=None*, *linkname=None*, *linkdescription=None*, *linksnippet=None*, *expires=None*, *update=None*, *camera=None*, *lookat=None*, ***kwargs*)

Controls the behavior of files fetched by a *simplekml.NetworkLink*.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
kml.document = None  # Removes the default document
kml.networklinkcontrol.minrefreshperiod = 5  # By accessing the␣
↪networklinkcontrol property one it created
kml.save('NetworkLinkControl.kml')
```

New in version 1.1.1

camera
Camera that views the scene, accepts *simplekml.Camera*

New in version 1.1.1

cookie
Use this to append a string to the URL query on the next refresh of the network link, accepts string.

New in version 1.1.1

expires
Date/time at which the link should be refreshed, accepts string.

New in version 1.1.1

linkdescription
Description of the network link, accepts string.

New in version 1.1.1

linkname
Name of the network link, accepts string.

New in version 1.1.1

linksnippet
Short description of the feature, accepts *simplekml.LinkSnippet*

New in version 1.1.1

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

New in version 1.1.1

maxsessionlength
Maximum amount of time for which the client *simplekml.NetworkLink* can remain connected in seconds, accepts int.

New in version 1.1.1

message
A message that appears when the network link is first loaded into Google Earth, accepts string.

New in version 1.1.1

minrefreshperiod
Minimum allowed time between fetches of the file in seconds, accepts int.

New in version 1.1.1

update
Instance of *simplekml.Update*

New in version 1.1.1

LinkSnippet

class simplekml.**LinkSnippet**(***kwargs*)
A short description of the feature.

Arguments are the same as the properties.

New in version 1.1.1

content
The description to be used in the snippet, accepts string.

maxlines
Number of lines to display, accepts int.

2.3.6 Overlays

GroundOverlay

class simplekml.**GroundOverlay**(*altitude=None*, *altitudemode=None*, *gxaltitudemode=None*, *latlonbox=None*, *gxlatlonquad=None*, ***kwargs*)
Draws an image overlay draped onto the terrain.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
ground = kml.newgroundoverlay(name='GroundOverlay')
ground.icon.href = 'http://simplekml.googlecode.com/hg/samples/resources/smile.png
↪'
ground.gxlatlonquad.coords = [(18.410524,-33.903972),(18.411429,-33.904171),
                              (18.411757,-33.902944),(18.410850,-33.902767)]
# or
#ground.latlonbox.north = -33.902828
#ground.latlonbox.south = -33.904104
#ground.latlonbox.east =  18.410684
#ground.latlonbox.west =  18.411633
#ground.latlonbox.rotation = -14
kml.save("GroundOverlay.kml")
```

address
Standard address, accepts string.

altitude
Distance above earth surface, accepts float.

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts *simplekml.AltitudeMode* constants.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

color
The color of the overlay, accepts hex string.

description
Description shown in the information balloon, accepts string.

draworder
The order to draw the overlay, accepts int.

extendeddata
Extra data for the feature.

gxaltitudemode
Specifies how the altitude for the Camera is interpreted.

With the addition of being relative to the sea floor. Accepts *simplekml.GxAltitudeMode* constants.

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

gxlatlonquad
Specifies the coordinates of the four corner points of a quadrilateral defining the overlay area. Accepts *simplekml.GxLatLonQuad*

icon
The icon to use for the overlay, accepts *simplekml.Icon*

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
Id number of feature, read-only.

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

latlonbox
Specifies where the top, bottom, right, and left sides are.

Accepts *simplekml.LatLonBox*.

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

open
Whether open or closed in Places panel, accepts int 0 or 1.

phonenumber
Phone number used by Google Maps mobile, accepts string.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

styleurl
Reference to the current styleurl or the feature, accepts string.

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

Note: There seems to be a bug in Google Earth where the inclusion of the namespace xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0" seems to break some other elements of the KML such as touring (a tour will not play). If xaladdressdetails is used the above namespace will be added to the KML and will possibly break other elements. Use with caution.

ScreenOverlay

class simplekml.**ScreenOverlay**(*overlayxy=None*, *screenxy=None*, *rotationxy=None*, *size=None*, *rotation=None*, ***kwargs*)

Draws an image overlay fixed to the screen.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
screen = kml.newscreenoverlay(name='ScreenOverlay')
screen.icon.href = 'http://simplekml.googlecode.com/hg/samples/resources/
↪simplekml-logo.png'
screen.overlayxy = simplekml.OverlayXY(x=0,y=1,xunits=simplekml.Units.fraction,
                                       yunits=simplekml.Units.fraction)
screen.screenxy = simplekml.ScreenXY(x=15,y=15,xunits=simplekml.Units.pixels,
                                     yunits=simplekml.Units.insetpixels)
screen.size.x = -1
screen.size.y = -1
screen.size.xunits = simplekml.Units.fraction
screen.size.yunits = simplekml.Units.fraction
kml.save("ScreenOverlay.kml")
```

address
Standard address, accepts string.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

color
The color of the overlay, accepts hex string.

description
Description shown in the information balloon, accepts string.

draworder
The order to draw the overlay, accepts int.

extendeddata
Extra data for the feature.

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

icon
The icon to use for the overlay, accepts *simplekml.Icon*

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
Id number of feature, read-only.

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

open
Whether open or closed in Places panel, accepts int 0 or 1.

overlayxy
Point on the overlay image that is mapped to a screen coordinate.

Specifies a point on (or outside of) the overlay image that is mapped to the screen coordinate *simplekml.ScreenXY*, accepts *simplekml.OverlayXY*

phonenumber
Phone number used by Google Maps mobile, accepts string.

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

rotation
Rotation of the overlay, accepts float.

rotationxy
Point relative to the screen about which the overlay is rotated.

Accepts *simplekml.RotationXY*

screenxy
Point relative to screen origin that the image is mapped to.

Specifies a point relative to the screen origin that the overlay image is mapped to, accepts *simplekml.ScreenXY*

size
The size of the image for the screen overlay, accepts *simplekml.Size*

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

styleurl
Reference to the current styleurl or the feature, accepts string.

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

Note: There seems to be a bug in Google Earth where the inclusion of the namespace xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0" seems to break some other elements of the KML such as touring (a tour will not play). If xaladdressdetails is used the above namespace will be added to the KML and will possibly break other elements. Use with caution.

PhotoOverlay

class simplekml.**PhotoOverlay**(*rotation=None*, *viewvolume=None*, *imagepyramid=None*, *point=None*, *shape=None*, ***kwargs*)

Geographically locate a photograph in Google Earth.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
photo = kml.newphotooverlay(name='PhotoOverlay Test')
photo.camera = simplekml.Camera(longitude=18.410858, latitude=-33.904446,␣
↪altitude=50,
                                altitudemode=simplekml.AltitudeMode.clamptoground)
photo.point.coords = [(18.410858,-33.90444)]
photo.style.iconstyle.icon.href = 'http://maps.google.com/mapfiles/kml/shapes/
↪camera.png'
photo.icon.href = 'http://simplekml.googlecode.com/hg/samples/resources/stadium.
↪jpg'
photo.viewvolume = simplekml.ViewVolume(-25,25,-15,15,1)
kml.save("PhotoOverlay.kml")
```

address
Standard address, accepts string.

atomauthor
Author of the feature, accepts string.

atomlink
URL containing this KML, accepts string.

balloonstyle
BalloonStyle of the feature, accepts *simplekml.BalloonStyle*

camera
Camera that views the scene, accepts *simplekml.Camera*

color
The color of the overlay, accepts hex string.

description
Description shown in the information balloon, accepts string.

draworder
The order to draw the overlay, accepts int.

extendeddata
Extra data for the feature.

gxballoonvisibility
Toggles visibility of a description balloon, accepts int 0 or 1

New in version 1.1.1

icon
The icon to use for the overlay, accepts *simplekml.Icon*

iconstyle
IconStyle of the feature, accepts *simplekml.IconStyle*

id
Id number of feature, read-only.

imagepyramid
Hierarchical set of images, accepts *simplekml.ImagePyramid*

labelstyle
LabelStyle of the feature, accepts *simplekml.LabelStyle*

linestyle
LineStyle of the feature, accepts *simplekml.LineStyle*

liststyle
ListStyle of the feature, accepts *simplekml.ListStyle*

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

name
Name of placemark, accepts string.

open
Whether open or closed in Places panel, accepts int 0 or 1.

phonenumber
Phone number used by Google Maps mobile, accepts string.

point
Draws an icon to mark the position of the overlay,accepts *simplekml.Point*

polystyle
PolyStyle of the feature, accepts *simplekml.PolyStyle*

region
Bounding box of feature, accepts *simplekml.Region*

rotation
Rotation of the overlay, accepts float.

shape
Shape the photo is drawn, accepts string from *simplekml.Shape* constants.

snippet
Short description of the feature, accepts *simplekml.Snippet*

style
The current style of the feature, accepts *simplekml.Style*

stylemap
The current StyleMap of the feature, accepts *simplekml.StyleMap*

styleurl
Reference to the current styleurl or the feature, accepts string.

timespan
Period of time, accepts *simplekml.TimeSpan*

timestamp
Single moment in time, accepts *simplekml.TimeStamp*

viewvolume
How much of the current scene is visible, accepts *simplekml.ViewVolume*

visibility
Whether the feature is shown, accepts int 0 or 1.

xaladdressdetails
Address in xAL format, accepts string.

Note: There seems to be a bug in Google Earth where the inclusion of the namespace xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0" seems to break some other elements of the KML such as touring (a tour will not play). If xaladdressdetails is used the above namespace will be added to the KML and will possibly break other elements. Use with caution.

2.3.7 Schema

Data

class simplekml.**Data**(*name=None*, *value=None*, *displayname=None*)
Data of extended data used to add custom data to KML Features.

The arguments are the same as the properties.

displayname
The name that is displayed to the user, accepts string.

name
Data name, accepts string.

value
Data value, accepts string.

ExtendedData

class simplekml.**ExtendedData**
Data of a schema that is used to add custom data to KML Features.

The arguments are the same as the properties.

newdata(*name*, *value*, *displayname=None*)
Creates a new *simplekml.Data* and attaches it to this schemadata.

Returns an instance of *simplekml.Data* class.

Args:

- name: string name of simplefield (required)
- value: int, float or string for value of field (required)
- displayname: string for pretty name that will be displayed (default None)

schemadata
Extra data for the feature, accepts *simplekml.SchemaData*.

GxSimpleArrayData

class simplekml.**GxSimpleArrayData**(*name*, *values=None*)
Data of a *simplekml.GxSimpleArrayField*.

The arguments are the same as the properties.

name
Name of field, accepts string.

newvalue(*value*)
Adds a value to the gxsimpledarraydata.

GxSimpleArrayField

class simplekml.**GxSimpleArrayField**(*name=None*, *type='string'*, *displayname=None*)
Custom array field for gx:track, forms part of a schema.

Args:

- *same as properties*
- *all other args same as simplekml.SimpleField*

displayname
Pretty name of field that is shown in google earth, accepts string.

name
Name of field, accepts string.

type
Type of field, accepts string from *simplekml.Types* constants.

Schema

class simplekml.**Schema**(*name=None*)
Custom KML schema that is used to add custom data to KML Features.

The arguments are the same as the properties.

id
Unique id of the schema.

name
Name of schema, accepts string.

newgxsimplearrayfield(*name*, *type*, *displayname=None*)
Creates a new *simplekml.GxSimpleArrayField* and attaches it to this schema.

Returns an instance of *simplekml.GxSimpleArrayField* class.

Args:

- name: string name of simplefield (required)
- type: string type of field (required)
- displayname: string for pretty name that will be displayed (default None)

newsimplefield(*name*, *type*, *displayname=None*)
Creates a new *simplekml.SimpleField* and attaches it to this schema.

Returns an instance of *simplekml.SimpleField* class.

Args:

- name: string name of simplefield (required)
- type: string type of field (required)
- displayname: string for pretty name that will be displayed (default None)

SchemaData

class simplekml.**SchemaData**(*schemaurl=None*)
Data of a schema that is used to add custom data to KML Features.

The arguments are the same as the properties.

newgxsimplearraydata(*name*, *value*)
Creates a new *simplekml.GxSimpleArrayData* and attaches it to this schemadata.

Returns an instance of *simplekml.GxSimpleArrayData* class.

Args:

- name: string name of simplefield (required)
- value: int, float or string for value of field (required)

newsimpledata(*name*, *value*)
Creates a new *simplekml.SimpleData* and attaches it to this schemadata.

Returns an instance of *simplekml.SimpleData* class.

Args:

- name: string name of simplefield (required)
- value: int, float or string for value of field (required)

schemaurl
Schema url, accepts string.

SimpleData

class simplekml.**SimpleData**(*name*, *value*)
Data of a schema.

The arguments are the same as the properties.

name
Name of field, accepts string.

value
Value of field, accepts int, float or string.

SimpleField

class simplekml.**SimpleField**(*name=None*, *type='string'*, *displayname=None*)
Custom field, forms part of a schema.

The arguments are the same as the properties.

displayname
Pretty name of field that is shown in google earth, accepts string.

name
Name of field, accepts string.

type
Type of field, accepts string from *simplekml.Types* constants.

2.3.8 Styles

Style

class simplekml.**Style**(*iconstyle=None*, *labelstyle=None*, *linestyle=None*, *polystyle=None*, *balloonstyle=None*, *liststyle=None*)
Styles affect how Geometry is presented.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.coords = [(1.0, 2.0)]
pnt.style.labelstyle.color = simplekml.Color.red  # Make the text red
pnt.style.labelstyle.scale = 2  # Make the text twice as big
pnt.style.iconstyle.icon.href = 'http://maps.google.com/mapfiles/kml/shapes/
↪placemark_circle.png'
kml.save("Style.kml")
```

balloonstyle
The balloonstyle, accepts *simplekml.BalloonStyle*.

iconstyle
The iconstyle, accepts *simplekml.IconStyle*.

id
The id of the style, read-only.

labelstyle
The labelstyle, accepts *simplekml.LabelStyle*.

linestyle
The linestyle, accepts *simplekml.LineStyle*.

liststyle
The liststyle, accepts *simplekml.ListStyle*.

polystyle
The polystyle, accepts *simplekml.PolyStyle*.

StyleMap

class simplekml.**StyleMap**(*normalstyle=None*, *highlightstyle=None*)

Styles affect how Geometry is presented.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(coords=[(18.432314,-33.988862)])
pnt.stylemap.normalstyle.labelstyle.color = simplekml.Color.blue
pnt.stylemap.highlightstyle.labelstyle.color = simplekml.Color.red
kml.save("StyleMap.kml")
```

highlightstyle

The highlighted *simplekml.Style*, accepts *simplekml.Style*.

id

The id of the style, read-only.

normalstyle

The normal *simplekml.Style*, accepts *simplekml.Style*.

BalloonStyle

class simplekml.**BalloonStyle**(*bgcolor=None*, *textcolor=None*, *text=None*, *displaymode='default'*)

Specifies the content and layout of the description balloon.

The arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name="BallonStyle", coords=[(18.429191, -33.987286)])
pnt.style.balloonstyle.text = 'These are trees and this text is blue with a green␣
↪background.'
pnt.style.balloonstyle.bgcolor = simplekml.Color.lightgreen
pnt.style.balloonstyle.textcolor = simplekml.Color.rgb(0, 0, 255)
kml.save("BalloomStyle.kml")
```

bgcolor

Background color of the balloon, accepts hex string.

displaymode

How the balloon is tyo be displayed, accepts string from *simplekml.DisplayMode* constants.

id

The unique id of the substyle.

text

The actual text that will appear in the balloon, accepts string.

textcolor

Text color in the balloon, accepts hex string.

IconStyle

class simplekml.**IconStyle**(*scale=1*, *heading=0*, *icon=None*, *hotspot=None*, ***kwargs*)

Specifies how icons for point Placemarks are drawn.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.coords = [(1.0, 2.0)]
pnt.style.iconstyle.scale = 3  # Icon thrice as big
pnt.style.iconstyle.icon.href = 'http://maps.google.com/mapfiles/kml/shapes/info-
↪i.png'
kml.save("IconStyle.kml")
```

color
Hex string representing a color, accepts string.

colormode
How the color is to be used, string from *simplekml.ColorMode* constants.

heading
Rotation of the icon, accepts float.

hotspot
Anchor position inside of the icon, accepts *simplekml.HotSpot*.

icon
The actual *simplekml.Icon* to be displayed, accepts *simplekml.Icon*.

id
The unique id of the substyle.

scale
Size of the icon, accepts float.

LabelStyle

class simplekml.**LabelStyle**(*scale=1*, ***kwargs*)

Specifies how the name of a Feature is drawn.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.coords = [(1.0, 2.0)]
pnt.style.labelstyle.color = simplekml.Color.red
pnt.style.labelstyle.scale = 2  # Text twice as big
pnt.style.labelstyle.color = simplekml.Color.blue
kml.save("LabelStyle.kml")
```

color
Hex string representing a color, accepts string.

colormode
How the color is to be used, string from *simplekml.ColorMode* constants.

id
The unique id of the substyle.

scale
Size of the icon, accepts float.

LineStyle

class simplekml.**LineStyle**(*width=None*, *gxoutercolor=None*, *gxouterwidth=None*, *gxphysicalwidth=None*, *gxlabelvisibility=None*, ***kwargs*)
Specifies the drawing style for all line geometry.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
lin = kml.newlinestring(name="Pathway", description="A pathway in Kirstenbosch",
                coords=[(18.43312,-33.98924), (18.43224,-33.98914),
                        (18.43144,-33.98911), (18.43095,-33.98904)])
lin.style.linestyle.color = simplekml.Color.red  # Red
lin.style.linestyle.width = 10  # 10 pixels
kml.save("LineStyle.kml")
```

color
Hex string representing a color, accepts string.

colormode
How the color is to be used, string from *simplekml.ColorMode* constants.

gxlabelvisibility
Whether or not to display a text label.

gxoutercolor
Outer color of the line, accepts string.

gxouterwidth
Outer width of the line, accepts float.

gxphysicalwidth
Physical width of the line, accepts float.

id
The unique id of the substyle.

width
Width of the line, accepts float.

ListStyle

class simplekml.**ListStyle**(*listitemtype='check'*, *bgcolor=None*, *itemicon=None*)
Specifies the display of the elements style in the navigation bar.

The arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
fol = kml.newfolder(name='Folder')
fol.style.liststyle.listitemtype = ListItemType.radiofolder
fol.style.liststyle.itemicon.href = 'http://maps.google.com/mapfiles/kml/shapes/
↪info.png'
kml.save("ListStyle.kml")
```

bgcolor
The background color of the item, accepts a hex string.

id
The unique id of the substyle.

itemicon
An instance of an *simplekml.ItemIcon* class, accepts *simplekml.ItemIcon*.

listitemtype
How an item is diaplyed, accepts string from *simplekml.ListItemType* constants.

PolyStyle

class simplekml.**PolyStyle**(*fill=1*, *outline=1*, ***kwargs*)
Specifies the drawing style for all polygons.

Arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pol = kml.newpolygon(name="Atrium Garden",
             outerboundaryis=[(18.43348,-33.98985),(18.43387,-33.99004),(18.43410,
↪-33.98972),
                              (18.43371,-33.98952),(18.43348,-33.98985)],
             innerboundaryis=[(18.43360,-33.98982),(18.43386,-33.98995),(18.43401,
↪-33.98974),
                              (18.43376,-33.98962),(18.43360,-33.98982)])
pol.style.polystyle.color = simplekml.Color.red
pol.style.polystyle.outline = 0
kml.save("PolyStyle.kml")
```

color
Hex string representing a color, accepts string.

colormode
How the color is to be used, string from *simplekml.ColorMode* constants.

fill
Must the polygon be filled, accepts int of 0 or 1.

id
The unique id of the substyle.

outline
Must the polygon be outlined, accepts int of 0 or 1.

2.3.9 TimePrimitives

TimeSpan

class simplekml.**TimeSpan**(*begin=None*, *end=None*)

Represents an extent in time bounded by begin and end dates.

The arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.timespan.begin = "2011-02-20"
pnt.timespan.end = "2012-07-31"
kml.save("TimeStamp.kml")
```

begin

The starting time, accepts string.

end

The ending time, accepts string.

GxTimeSpan

class simplekml.**GxTimeSpan**(***kwargs*)

A copy of the *simplekml.TimeSpan* element, in the extension namespace.

Args:

- *same as properties*
- *all other args same as simplekml.TimeSpan*

Usage:

```
import simplekml
kml = simplekml.Kml()
ls = kml.newlinestring(name='A LineString')
ls.coords = [(18.333868,-34.038274,10.0), (18.370618,-34.034421,10.0)]
ls.extrude = 1
ls.altitudemode = simplekml.AltitudeMode.relativetoground
ls.lookat.gxaltitudemode = simplekml.GxAltitudeMode.relativetoseafloor
ls.lookat.latitude = -34.028242
ls.lookat.longitude = 18.356852
ls.lookat.range = 3000
ls.lookat.heading = 56
ls.lookat.tilt = 78
ls.lookat.gxtimespan.begin = "2011-02-20"
ls.lookat.gxtimespan.end = "2012-07-31"
kml.save("GxTimeSpan.kml")
```

begin

The starting time, accepts string.

end

The ending time, accepts string.

id
The id string.

TimeStamp

class simplekml.**TimeStamp**(*when=None*)
Represents a single moment in time.

The arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.timestamp.when = 2011
kml.save("TimeStamp.kml")
```

when
A moment in time, accepts string.

GxTimeStamp

class simplekml.**GxTimeStamp**(***kwargs*)
A copy of the *simplekml.TimeStamp* element, in the extension namespace.

Args:

- *same as properties*
- *all other args same as simplekml.TimeStamp*

Usage:

```
import simplekml
kml = simplekml.Kml()
ls = kml.newlinestring(name='A LineString')
ls.coords = [(18.333868,-34.038274,10.0), (18.370618,-34.034421,10.0)]
ls.extrude = 1
ls.altitudemode = simplekml.AltitudeMode.relativetoground
ls.lookat.gxaltitudemode = simplekml.GxAltitudeMode.relativetoseafloor
ls.lookat.latitude = -34.028242
ls.lookat.longitude = 18.356852
ls.lookat.range = 3000
ls.lookat.heading = 56
ls.lookat.tilt = 78
ls.lookat.gxtimestamp.when = 2011
kml.save("GxTimeStamp.kml")
```

id
The id string.

when
A moment in time, accepts string.

2.3.10 Tour

GxAnimatedUpdate

class simplekml.**GxAnimatedUpdate**(*gxduration=None*, *gxdelayedstart=None*, *update=None*)
Controls changes during a tour to KML features.

The arguments are the same as the properties. See *simplekml.GxTour* for usage.

gxdelayedstart
Double of number of seconds to wait before starting.

gxduration
Double indicating how long the camera remains still.

id
The id string.

update
Instance of *simplekml.Update*

GxFlyTo

class simplekml.**GxFlyTo**(*gxduration=None*, *gxflytomode=None*, *camera=None*, *lookat=None*)
Allows unbroken flight from point to point.

The arguments are the same as the properties. See *simplekml.GxTour* for usage.

addfile(*path*)

bounce = 'bounce'

camera
Camera that views the scene, accepts *simplekml.Camera*

gxduration
Double indicating how long the camera remains still.

gxflytomode
How the camera behaves, accepts simplekml.GxFlyToMode constants.

id
The id string.

lookat
Camera relative to the feature, accepts *simplekml.LookAt*

smooth = 'smooth'

GxPlaylist

class simplekml.**GxPlaylist**(*gxtourprimitives=None*)
Defines a part of a tour.

The arguments are the same as the properties. See *simplekml.GxTour* for usage.

addgxtourprimitive(*value*)

newgxanimatedupdate(***kwargs*)
Creates a new *simplekml.GxAnimatedUpdate* and adds it to the playlist.

Accepts the same agruments as *simplekml.GxAnimatedUpdate* and returns an instance of *simplekml.GxAnimatedUpdate*

newgxflyto(***kwargs*)
Creates a new *simplekml.GxFlyTo* and adds it to the playlist.

Accepts the same agruments as *simplekml.GxFlyTo* and returns an instance of *simplekml.GxFlyTo*

newgxsoundcue(***kwargs*)
Creates a new *simplekml.GxSoundCue* and adds it to the playlist.

Accepts the same agruments as *simplekml.GxSoundCue* and returns an instance of *simplekml.GxSoundCue*

newgxtourcontrol(***kwargs*)
Creates a new *simplekml.GxTourControl* and adds it to the playlist.

Accepts the same agruments as *simplekml.GxTourControl* and returns an instance of *simplekml.GxTourControl*

newgxwait(***kwargs*)
Creates a new *simplekml.GxWait* and adds it to the playlist.

Accepts the same agruments as *simplekml.GxWait* and returns an instance of *simplekml.GxWait*

GxSoundCue

class simplekml.**GxSoundCue**(*href=None*, *gxdelayedstart=None*)
Specifies a sound to be played in a tour.

The arguments are the same as the properties. See *simplekml.GxTour* for usage.

gxdelayedstart
Double telling the number of seconds to delay playing the file.

href
A string reference to a sound file to play.

id
The id string.

GxTour

class simplekml.**GxTour**(*name=None*, *description=None*, *gxplaylists=None*)
Defines a tour.

The arguments are the same as the properties.

Usage:

```
# Demonstrates touring with the reproduction of the tour sample in the KML␣
↪Reference
# https://developers.google.com/kml/documentation/kmlreference#gxtour with the␣
↪addition of GxSoundCue

import os
```

(continues on next page)

(continued from previous page)

```
import simplekml

# Create an instance of kml
kml = simplekml.Kml(name="Tours", open=1)

# Create a new point and style it
pnt = kml.newpoint(name="New Zealand's Southern Alps", coords=[(170.144,-43.605)])
pnt.style.iconstyle.scale = 1.0

# Create a tour and attach a playlist to it
tour = kml.newgxtour(name="Play me!")
playlist = tour.newgxplaylist()

# Attach a gx:SoundCue to the playlist and delay playing by 2 second (sound clip␣
↪is about 4 seconds long)
soundcue = playlist.newgxsoundcue()
soundcue.href = "http://simplekml.googlecode.com/hg/samples/resources/drum_roll_1.
↪wav"
soundcue.gxdelayedstart = 2

# Attach a gx:AnimatedUpdate to the playlist
animatedupdate = playlist.newgxanimatedupdate(gxduration=6.5)
animatedupdate.update.change = '<IconStyle targetId="{0}"><scale>10.0</scale></
↪IconStyle>'.format(pnt.style.iconstyle.id)

# Attach a gx:FlyTo to the playlist
flyto = playlist.newgxflyto(gxduration=4.1)
flyto.camera.longitude = 170.157
flyto.camera.latitude = -43.671
flyto.camera.altitude = 9700
flyto.camera.heading = -6.333
flyto.camera.tilt = 33.5
flyto.camera.roll = 0

# Attach a gx:Wait to the playlist to give the gx:AnimatedUpdate time to finish
wait = playlist.newgxwait(gxduration=2.4)

# Save to file
kml.save(os.path.splitext(__file__)[0] + ".kml")
```

description
String description of the tour.

name
String name of the tour

newgxplaylist(*gxtourprimitives=None*)
Adds a *simplekml.GxPlaylist* and returns it.

GxTourControl

class simplekml.**GxTourControl**(*gxplaymode='pause'*)
Allows a tour to be paused.

The arguments are the same as the properties.

gxplaymode
String to pause the tour, accepts `simplekml.GxPlayMode` constants.

id
The id string.

Update

class simplekml.**Update**(*targethref=None*, *change=None*, *create=None*, *delete=None*)
Action to take when animation updates.

The arguments are the same as the properties. See *simplekml.GxTour* for usage.

change
KML string representing a change in animation.

create
KML string representing a creation during animation.

delete
KML string representing a deletion during animation.

targethref
The target url.

GxWait

class simplekml.**GxWait**(*gxduration=None*)
Allows a tour to be paused.

The arguments are the same as the properties. See *simplekml.GxTour* for usage.

gxduration
Double indicating how long the camera remains still.

id
The id string.

2.3.11 Various

Alias

class simplekml.**Alias**(*targethref=None*, *sourcehref=None*)
Contains a mapping from a sourcehref to a targethref.

The arguments are the same as the properties.

sourcehref
The source href, accepts string.

targethref
The target href, accepts string.

Box

class simplekml.**Box**(*north=None*, *south=None*, *east=None*, *west=None*)
Abstract class for box elements.

The arguments are the same as the properties.

Note: Not to be used directly.

---

east
Longitude of the east edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

north
Latitude of the north edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

south
Latitude of the south edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

west
Longitude of the west edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

HotSpot

class simplekml.**HotSpot**(***kwargs*)
Specifies the position inside the [Icon] that is anchored to the [Point].

Arguments are the same as the properties.

x
Number in xunits, accepts int.

xunits
Type of x units, see [Units] for values.

y
Number in yunits, accepts int.

yunits
Type of y units, See *simplekml.Units* for values.

Icon

class simplekml.**Icon**(*gxx=None*, *gxy=None*, *gxw=None*, *gxh=None*, ***kwargs*)
Defines an image associated with an Icon style or overlay.

The arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.coords = [(1.0, 2.0)]
pnt.style.iconstyle.icon.href = 'http://maps.google.com/mapfiles/kml/shapes/
↪placemark_circle.png'
kml.save("Icon.kml")
```

gxh
Height of icon palette, accpets int.

gxw
Width of icon palette, accpets int.

gxx
x position of icon palette, accpets int.

gxy
y position of icon palette, accpets int.

href
Target url, accepts string.

httpquery
Extra information to append to the query string, accepts string.

id
The id string.

refreshinterval
Time between refreshed, accepts float.

refreshmode
Type of refresh, accepts string of *simplekml.RefreshMode* constants.

viewboundscale
Extent to request from server, accepts float.

viewformat
Format of the query string, accepts string.

viewrefreshmode
Camera specific refresh, accepts *simplekml.ViewRefreshMode* constants.

viewrefreshtime
Camera specific refresh time, accepts float.

ImagePyramid

class simplekml.**ImagePyramid**(*titlesize=256*, *maxwidth=0*, *maxheight=0*, *gridorigin='lowerLeft'*)
A hierarchical set of images.

The arguments are the same as the properties.

gridorigin
Specifies where to begin numbering the tiles, accepts string.

maxheight
Height in pixels of the original image, accepts int.

maxwidth
Width in pixels of the original image, accepts int.

titlesize
Size of the tiles, in pixels, accepts int.

ItemIcon

class simplekml.**ItemIcon**(*state=None*, *href=None*)
con used in the List view that reflects the state of a Folder or Link fetch.

The arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
pnt = kml.newpoint(name='A Point')
pnt.coords = [(1.0, 2.0)]
pnt.style.liststyle.itemicon.href = 'http://maps.google.com/mapfiles/kml/shapes/
↪info.png'
kml.save('ItemIcon.kml')
```

href
URL of the image used in List View for Feature, accepts string.

state
Current state of the link, accepts string from *simplekml.State* constants.

GxLatLonQuad

class simplekml.**GxLatLonQuad**(*coords=None*)
Used for nonrectangular quadrilateral ground overlays.

The arguments are the same as the properties.

coords
Four corners of quad coordinates, accepts list of four tuples in the order lon, lat.

The coordinates must be specified in counter-clockwise order with the first coordinate corresponding to the lower-left corner of the overlayed image. eg. [(0, 1), (1,1), (1,0), (0,0)]

LatLonAltBox

class simplekml.**LatLonAltBox**(*minaltitude=0*, *maxaltitude=0*, *altitudemode='clampToGround'*, ***kwargs*)
A bounding box that describes an area of interest defined by geographic coordinates and altitudes.

Args:

- *same as properties*
- *all other args same as* *simplekml.Box*

altitudemode
Specifies how the altitude for the Camera is interpreted.

Accepts simplkml.AltitudeMode constants.

east
Longitude of the east edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

maxaltitude
Maximum altitude in meters, accepts float.

minaltitude
Minimum altitude in meters, accepts float.

north
Latitude of the north edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

south
Latitude of the south edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

west
Longitude of the west edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

LatLonBox

class simplekml.**LatLonBox**(*rotation=None*, ***kwargs*)
Specifies where the top, bottom, right, and left sides of a bounding box for the ground overlay are aligned.

Args:

- *same as properties*
- *all other args same as* `simplekml.Box`

east
Longitude of the east edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

north
Latitude of the north edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

rotation
Rotation of the overlay about its center, in degrees.

Values can be 180, accepts float.

south
Latitude of the south edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

west
Longitude of the west edge of the bounding box, in decimal degrees from 0 to 90, accepts float.

Link

class simplekml.**Link**(*href=' '*, *refreshmode=None*, *refreshinterval=None*, *viewrefreshmode=None*, *viewrefreshtime=None*, *viewboundscale=None*, *viewformat=None*, *httpquery=None*)

Specifies the location of:

- KML files fetched by network links
- Model files

The arguments are the same as the properties.

Usage:

```
import simplekml
kml = simplekml.Kml()
netlink = kml.newnetworklink(name="Network Link")
netlink.link.href = "http://simplekml.googlecode.com/hg/samples/samples.kml"
```

(continues on next page)

(continued from previous page)

```
netlink.link.viewrefreshmode = simplekml.ViewRefreshMode.onrequest
kml.save("Link.kml")
```

href
Target url, accepts string.

httpquery
Extra information to append to the query string, accepts string.

id
The id string.

refreshinterval
Time between refreshed, accepts float.

refreshmode
Type of refresh, accepts string of *simplekml.RefreshMode* constants.

viewboundscale
Extent to request from server, accepts float.

viewformat
Format of the query string, accepts string.

viewrefreshmode
Camera specific refresh, accepts *simplekml.ViewRefreshMode* constants.

viewrefreshtime
Camera specific refresh time, accepts float.

Location

class simplekml.**Location**(*longitude=None*, *latitude=None*, *altitude=0*)
Specifies the exact coordinates of the Model's origin.

The arguments are the same as the properties.

altitude
Height above the earth's surface in meters, accepts float.

latitude
Decimal degree, accepts float.

longitude
Decimal degree, accepts float.

Lod

class simplekml.**Lod**(*minlodpixels=0*, *maxlodpixels=-1*, *minfadeextent=0*, *maxfadeextent=0*)
Level of Detail describes the size of the projected region.

The arguments are the same as the properties.

maxfadeextent
Maximum distance over which the geometry fades, accepts int.

maxlodpixels
Maximum limit of the visibility range, accepts int.

minfadeextent
Minumum distance over which the geometry fades, accepts int.

minlodpixels
Minimum limit of the visibility range, accepts int.

Orientation

class simplekml.**Orientation**(*heading=0*, *tilt=0*, *roll=0*)
Describes rotation of a 3D model's coordinate system.

The arguments are the same as the properties.

heading
Rotation about the z axis, accepts float.

roll
Rotation about the y axis, accepts float.

tilt
Rotation about the x axis, accepts float.

OverlayXY

class simplekml.**OverlayXY**(***kwargs*)
Point in overlay image that is mapped to screen coordinate *simplekml.ScreenXY*

Arguments are the same as the properties.

x
Number in xunits, accepts int.

xunits
Type of x units, see [Units] for values.

y
Number in yunits, accepts int.

yunits
Type of y units, See *simplekml.Units* for values.

Region

class simplekml.**Region**(*latlonaltbox=None*, *lod=None*)
Used for nonrectangular quadrilateral ground overlays.

The arguments are the same as the properties.

latlonaltbox
Bounding box that describes an area, accepts *simplkml.LatLonAltBox*

lod
Level of Detail, accepts *simplkml.Lod*

ResourceMap

class simplekml.**ResourceMap**(*aliases=None*)
Contains and specifies 0 or more [Alias] elements.

The arguments are the same as the properties.

aliases
A list of all the aliases, accepts a list of aliases

newalias(***kwargs*)
Creates a new *simplekml.Alias* and attaches it to the *simplekml.ResourceMap*.

Args:

- Same as *simplekml.Alias*

RotationXY

class simplekml.**RotationXY**(***kwargs*)
Point relative to the screen about which the screen overlay is rotated.

Arguments are the same as the properties.

x
Number in xunits, accepts int.

xunits
Type of x units, see [Units] for values.

y
Number in yunits, accepts int.

yunits
Type of y units, See *simplekml.Units* for values.

Scale

class simplekml.**Scale**(*x=1*, *y=1*, *z=1*)
Scales a model along the x, y, and z axes in the model's coordinate space.

The arguments are the same as the properties.

x
Scale in the x direction, accepts float.

y
Scale in the y direction, accepts float.

z
Scale in the z direction, accepts float.

ScreenXY

class simplekml.**ScreenXY**(***kwargs*)
Point relative to the screen origin that the overlay image is mapped to.

Arguments are the same as the properties.

x
Number in xunits, accepts int.

xunits
Type of x units, see [Units] for values.

y
Number in yunits, accepts int.

yunits
Type of y units, See *simplekml.Units* for values.

Size

class simplekml.**Size**(***kwargs*)
Specifies the size of the image for the screen overlay.

Arguments are the same as the properties.

x
Number in xunits, accepts int.

xunits
Type of x units, see [Units] for values.

y
Number in yunits, accepts int.

yunits
Type of y units, See *simplekml.Units* for values.

Snippet

class simplekml.**Snippet**(*content=''*, *maxlines=None*)
A short description of the feature.

Arguments are the same as the properties.

content
The description to be used in the snippet, accepts string.

maxlines
Number of lines to display, accepts int.

ViewVolume

class simplekml.**ViewVolume**(*leftfov=0*, *rightfov=0*, *bottomfov=0*, *topfov=0*, *near=0*)
Defines how much of the current scene is visible.

The arguments are the same as the properties.

bottomfov
Angle, in degrees, accepts float.

leftfov
Angle, in degrees, accepts float.

near
Measurement of viewing direction from the camera, accepts float.

rightfov
Angle, in degrees, accepts float.

topfov
Angle, in degrees, accepts float.

2.4 Styling

A style tells Google Earth how to render a feature. For more information on styling please see KML Reference.

2.4.1 Concept

Every feature can have a *simplekml.Style* that tells Google Earth how to render it. A *simplekml.Style* can have different 'substyles': *simplekml.IconStyle*, *simplekml.IconStyle*, *simplekml.LineStyle*, *simplekml.PolyStyle*, *simplekml.BalloonStyle* and *simplekml.ListStyle*.

In simplekml a feature, by default, has no style, but as soon as you assign a value to one of the feature's *simplekml.Style*'s properties the style is automatically created. In the generated KML the style becomes a child of the containing element (*simplekml.Document*, *simplekml.Folder*, etc). Here is an example:

```
from simplekml import Kml

kml = Kml()
fol = kml.newfolder("A Folder")
pnt = fol.newpoint(name="Kirstenbosch", coords=[(18.432314,-33.988862)])
pnt.style.labelstyle.color = 'ff0000ff'  # Red
kml.save("singlestyle.kml")
```

With the resulting generated KML:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom"␣
↪xmlns:gx="http://www.google.com/kml/ext/2.2" xmlns:kml="http://www.opengis.net/kml/
↪2.2" xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
    <Document id="feat_1">
        <Folder id="feat_2">
            <Style id="stylesel_0">
                <LabelStyle>
                    <color>ff0000ff</color>
                    <colorMode>normal</colorMode>
                    <scale>1</scale>
                </LabelStyle>
            </Style>
            <name>A Folder</name>
            <Placemark id="feat_3">
                <name>Kirstenbosch</name>
                <styleUrl>#stylesel_0</styleUrl>
                <Point id="geom_0">
                    <coordinates>18.432314,-33.988862,0.0</coordinates>
                </Point>
            </Placemark>
        </Folder>
    </Document>
</kml>
```

Above we created a *simplekml.Point* inside of a *simplekml.Folder* and then changed the color of the point's label by typing *pnt.style.labelstyle.color = 'ff0000ff'*. This resulted in a folder containing a `simplekml.Placemark` with a point as a child element. The placemark also contains a reference to the *simplekml.Style* *<styleUrl>#stylesel_0</styleUrl>*, which is a child of the folder with a labelstyle as a child.

The above is fine if we are dealing with one or to features, but if we are dealing with thousands of points the generated KML becomes very bloated, because every time you access a features style's properties a new style is created. Just imagine we modified the above to do the following:

```
from simplekml import Kml

kml = Kml()
fol = kml.newfolder(name="A Folder")
for lon in range(-180, 180, 10):
    for lat in range(-180, 180, 10):  # 10 Degree grid of points
        pnt = fol.newpoint(name="{0},{1}".format(lon, lat), coords=[(lon,lat)])
        pnt.style.labelstyle.color = 'ff0000ff'  # Red

kml.save("manystyles.kml")
```

And the generated KML:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom"␣
↪xmlns:gx="http://www.google.com/kml/ext/2.2" xmlns:kml="http://www.opengis.net/kml/
↪2.2" xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
    <Document id="feat_1">
        <Folder id="feat_2">
            <Style id="stylesel_0">
                <LabelStyle>
                    <color>ff0000ff</color>
                    <colorMode>normal</colorMode>
                    <scale>1</scale>
                </LabelStyle>
            </Style>
            <Style id="stylesel_1">
                <LabelStyle>
                    <color>ff0000ff</color>
                    <colorMode>normal</colorMode>
                    <scale>1</scale>
                </LabelStyle>
            </Style>
            <Style id="stylesel_2">
                <LabelStyle>
                    <color>ff0000ff</color>
                    <colorMode>normal</colorMode>
                    <scale>1</scale>
                </LabelStyle>
            </Style>

            ... many, many lines of kml go here

        </Folder>
    </Document>
</kml>
```

The above was abbreviated a bit because the KML contains (2*180/10)^2 styles (one for each of the points we created, which is 1296 styles). As you can imagine, the resulting KML file will be quite huge!

To make the KML much smaller we can create a 'shared' style and associate it with each feature:

```
from simplekml import Kml, Style

kml = Kml()

fol = kml.newfolder(name="A Folder")

sharedstyle = Style()
sharedstyle.labelstyle.color = 'ff0000ff'  # Red

for lon in range(-180, 180, 10):
    for lat in range(-180, 180, 10):  # 10 Degree grid of points
        pnt = fol.newpoint(name="{0},{1}".format(lon, lat), coords=[(lon,lat)])
    # pnt.style.labelstyle.color = 'ff0000ff'  # (Bad!) This results in (2*180/10)^2␣
↪styles
        pnt.style = sharedstyle                # (Much better!) This results in a␣
↪single styles

kml.save("sharedstyle.kml")
```

And the KML:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom"␣
↪xmlns:gx="http://www.google.com/kml/ext/2.2" xmlns:kml="http://www.opengis.net/kml/
↪2.2" xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0">
    <Document id="feat_1">
        <Folder id="feat_2">
            <Style id="stylesel_0">
                <LabelStyle>
                    <color>ff0000ff</color>
                    <colorMode>normal</colorMode>
                    <scale>1</scale>
                </LabelStyle>
            </Style>
            <name>A Folder</name>
            <Placemark id="feat_3">
                <name>-180,-180</name>
                <styleUrl>#stylesel_0</styleUrl>
                <Point id="geom_0">
                    <coordinates>-180,-180,0.0</coordinates>
                </Point>
            </Placemark>
            <Placemark id="feat_4">
                <name>-180,-170</name>
                <styleUrl>#stylesel_0</styleUrl>
                <Point id="geom_1">
                    <coordinates>-180,-170,0.0</coordinates>
                </Point>
            </Placemark>

            ... many, many more points (1294 to be exact)

        </Folder>
    </Document>
</kml>
```

Now this is much better! We only have one style at the beginning of the KML followed by all the points. What

happened here is that a 'shared' style was created by creating an instance of the *simplekml.Style* class *sharedstyle = Style()*, then the style's properties were changed and finally the *sharedstyle* was assigned to each point's style property.

In summary, there are two ways to style: changing the properties of an individual feature and creating a 'shared' style and assigning it to all the relevant features.

Note: There is a 'shorthand' method when dealing with changing the properties of an individual feature. The following 'longhand' line of code:

```
pnt.style.labelstyle.color = 'ff0000ff'  # Red
```

is the same as this 'shorthand' version:

```
pnt.labelstyle.color = 'ff0000ff'  # Red
```

This helps to eliminate the need to type *.style* every time you need to change a style's property, as well as, reducing the size of your script. But, the *shorthand* makes the code less readable. It is suggested that you use the long hand method.

2.4.2 Styling a Point

A *simplekml.Point* has two 'substyles' that can be altered to render it: *simplekml.IconStyle* and *simplekml.LabelStyle*. To change a point's style simply assign a value to one of its properties:

```
pnt = kml.newpoint(name="Kirstenbosch", coords=[(18.432314,-33.988862)])
pnt.style.labelstyle.color = 'ff0000ff'  # Red
```

That changed the text "Kirstenbosch" to red. See the KML Reference for the format of the color string (you could also use the *simplekml.Color* class). Now lets edit some more of the style:

```
pnt.style.labelstyle.scale = 2  # Text twice as big
pnt.style.iconstyle.color = 'ffff0000'  # Blue
pnt.style.iconstyle.scale = 3  # Icon thrice as big
pnt.style.iconstyle.icon.href = 'http://maps.google.com/mapfiles/kml/shapes/info-i.png
↪'
```

2.4.3 Styling a LineString

A *simplekml.LineString* has one 'substyle' that can be altered to render it:

```
lin = kml.newlinestring(name="Pathway", description="A pathway in Kirstenbosch",
                        coords=[(18.43312,-33.98924), (18.43224,-33.98914),
                                (18.43144,-33.98911), (18.43095,-33.98904)])
lin.style.linestyle.color = 'ff0000ff'  # Red
lin.style.linestyle.width= 10  # 10 pixels
```

2.4.4 Styling a Polygon

A *simplekml.Polygon* has two 'substyles' that can be altered to render it: *simplekml.LineStyle* and *simplekml.PolyStyle*. Below is code for a *simplekml.Polygon* without a border that is slightly transparent:

```
pol = kml.newpolygon(name="Atrium Garden",
                     outerboundaryis=[(18.43348,-33.98985), (18.43387,-33.99004),
                                      (18.43410,-33.98972), (18.43371,-33.98952),
                                      (18.43348,-33.98985)],
                     innerboundaryis=[(18.43360,-33.98982), (18.43386,-33.98995),
                                      (18.43401,-33.98974), (18.43376,-33.98962),
                                      (18.43360,-33.98982)])
pol.style.polystyle.color = '990000ff'  # Transparent red
pol.style.polystyle.outline = 0
```

2.4.5 Styling MultiGeometry

Applying a style to MultiGeometry applies the style to all the individual geometries in that MultiGeometry collection. Therefore, styling multigeometry is the same as styling normal geometry:

```
from simplekml import Kml
kml = Kml()
multipnt = kml.newmultigeometry(name="Points")
for lon in range(4):
    for lat in range(4):
        multipnt.newpoint(coords=[(lon,lat)])
multipnt.style.labelstyle.color = 'ff0000ff'  # Red
```

2.5 Tutorials

Here is a collection of tutorials covering the various aspects of simplekml. Tutorials are being added all the time, so check back regularly. To get the kml created in the tutorials download this samples file (the file includes all code samples).

2.5.1 Points Tutorial

About

How to create a simple point and change the points properties.

Creating the Code

First import simplekml and create a KML object:

```
import simplekml
```

We then have some base data to work with. It is a list of tuples containing (in the following order) a city name, time corresponding to 12:00 noon Eastern Standard Time, latitude and logitude:

```
cities = [
    ('Aberdeen, Scotland', '5:00 p.m.', 57.15, -2.15),
    ('Adelaide, Australia', '2:30 a.m.', -34.916667, 138.6),
    ('Algiers, Algeria', '6:00 p.m.', 36.833333, 3),
    # ...many, many more cities, and then...
```

(continues on next page)

(continued from previous page)

```
    ('Zurich, Switzerland', '6:00 p.m.', 47.35, 8.516667)
]
```

Create the KML object:

```
kml = simplekml.Kml(open=1) # the folder will be open in the table of contents
```

Next is a simple point example, we create a point feature at 0 degrees latitude and logitude and name it "The World". Here we pass all the information to the named parameters (note - the coordinates can contain an optional height value):

```
single_point = kml.newpoint(name="The World", coords=[(0.0,0.0)])
```

Next is a real world example, we create a point for each city. The points' properties are assigned after the point is created:

```
for city, time, lat, lon in cities:
    pnt = kml.newpoint()
    pnt.name = city
    pnt.description = "Time corresponding to 12:00 noon, Eastern Standard Time: {0}".
↪format(time)
    pnt.coords = [(lon, lat)]
```

And finally we save the kml:

```
kml.save("T00 Points.kml")
```

Complete Code Example

Here is the complete code:

```
import simplekml

# Cities of the World with their coordinates and time corresponding to 12:00 noon,␣
↪Eastern Standard Time
# Source: http://www.infoplease.com/ipa/A0001769.html
cities = [
    ('Aberdeen, Scotland', '5:00 p.m.', 57.15, -2.15),
    ('Adelaide, Australia', '2:30 a.m.', -34.916667, 138.6),
    ('Algiers, Algeria', '6:00 p.m.', 36.833333, 3),
    ('Amsterdam, Netherlands', '6:00 p.m.', 52.366667, 4.883333),
    ('Ankara, Turkey', '7:00 p.m.', 39.916667, 32.916667),
    ('Asuncion, Paraguay', '1:00 p.m.', -25.25, -57.666667),
    ('Athens, Greece', '7:00 p.m.', 37.966667, 23.716667),
    ('Auckland, New Zealand', '5:00 a.m.', -36.866667, 174.75),
    ('Bangkok, Thailand', 'midnight', 13.75, 100.5),
    ('Barcelona, Spain', '6:00 p.m.', 41.383333, 2.15),
    ('Beijing, China', '1:00 a.m.', 39.916667, 116.416667),
    ('Belem, Brazil', '2:00 p.m.', -1.466667, -48.483333),
    ('Belfast, Northern Ireland', '5:00 p.m.', 54.616667, -5.933333),
    ('Belgrade, Serbia', '6:00 p.m.', 44.866667, 20.533333),
    ('Berlin, Germany', '6:00 p.m.', 52.5, 13.416667),
    ('Birmingham, England', '5:00 p.m.', 52.416667, -1.916667),
    ('Bogota, Colombia', '12:00 noon', 4.533333, -74.25),
    ('Bombay, India', '10:30 p.m.', 19, 72.8),
    ('Bordeaux, France', '6:00 p.m.', 44.833333, -0.516667),
```

(continues on next page)

(continued from previous page)

```
    ('Bremen, Germany', '6:00 p.m.', 53.083333, 8.816667),
    ('Brisbane, Australia', '3:00 a.m.', -27.483333, 153.133333),
    ('Bristol, England', '5:00 p.m.', 51.466667, -2.583333),
    ('Brussels, Belgium', '6:00 p.m.', 50.866667, 4.366667),
    ('Bucharest, Romania', '7:00 p.m.', 44.416667, 26.116667),
    ('Budapest, Hungary', '6:00 p.m.', 47.5, 19.083333),
    ('Buenos Aires, Argentina', '2:00 p.m.', -34.583333, -58.366667),
    ('Cairo, Egypt', '7:00 p.m.', 30.033333, 31.35),
    ('Calcutta, India', '10:30 p.m.', 22.566667, 88.4),
    ('Canton, China', '1:00 a.m.', 23.116667, 113.25),
    ('Cape Town, South Africa', '7:00 p.m.', -33.916667, 18.366667),
    ('Caracas, Venezuela', '1:00 p.m.', 10.466667, -67.033333),
    ('Cayenne, French Guiana', '2:00 p.m.', 4.816667, -52.3),
    ('Chihuahua, Mexico', '10:00 a.m.', 28.616667, -106.083333),
    ('Chongqing, China', '1:00 a.m.', 29.766667, 106.566667),
    ('Copenhagen, Denmark', '6:00 p.m.', 55.666667, 12.566667),
    ('Cordoba, Argentina', '2:00 p.m.', -31.466667, -64.166667),
    ('Dakar, Senegal', '5:00 p.m.', 14.666667, -17.466667),
    ('Darwin, Australia', '2:30 a.m.', -12.466667, 130.85),
    ('Djibouti, Djibouti', '8:00 p.m.', 11.5, 43.05),
    ('Dublin, Ireland', '5:00 p.m.', 53.333333, -6.25),
    ('Durban, South Africa', '7:00 p.m.', -29.883333, 30.883333),
    ('Edinburgh, Scotland', '5:00 p.m.', 55.916667, -3.166667),
    ('Frankfurt, Germany', '6:00 p.m.', 50.116667, 8.683333),
    ('Georgetown, Guyana', '1:00 p.m.', 6.75, -58.25),
    ('Glasgow, Scotland', '5:00 p.m.', 55.833333, -4.25),
    ('Guatemala City, Guatemala', '11:00 a.m.', 14.616667, -90.516667),
    ('Guayaquil, Ecuador', '12:00 noon', -2.166667, -79.933333),
    ('Hamburg, Germany', '6:00 p.m.', 53.55, 10.033333),
    ('Hammerfest, Norway', '6:00 p.m.', 70.633333, 23.633333),
    ('Havana, Cuba', '12:00 noon', 23.133333, -82.383333),
    ('Helsinki, Finland', '7:00 p.m.', 60.166667, 25),
    ('Hobart, Tasmania', '3:00 a.m.', -42.866667, 147.316667),
    ('Hong Kong, China', '1:00 a.m.', 22.333333, 114.183333),
    ('Iquique, Chile', '1:00 p.m.', -20.166667, -70.116667),
    ('Irkutsk, Russia', '1:00 a.m.', 52.5, 104.333333),
    ('Jakarta, Indonesia', 'midnight', -6.266667, 106.8),
    ('Johannesburg, South Africa', '7:00 p.m.', -26.2, 28.066667),
    ('Kingston, Jamaica', '12:00 noon', 17.983333, -76.816667),
    ('Kinshasa, Congo', '6:00 p.m.', -4.3, 15.283333),
    ('Kuala Lumpur, Malaysia', '1:00 a.m.', 3.133333, 101.7),
    ('La Paz, Bolivia', '1:00 p.m.', -16.45, -68.366667),
    ('Leeds, England', '5:00 p.m.', 53.75, -1.5),
    ('Lima, Peru', '12:00 noon', -12, -77.033333),
    ('Lisbon, Portugal', '5:00 p.m.', 38.733333, -9.15),
    ('Liverpool, England', '5:00 p.m.', 53.416667, -3),
    ('London, England', '5:00 p.m.', 51.533333, -0.083333),
    ('Lyons, France', '6:00 p.m.', 45.75, 4.833333),
    ('Madrid, Spain', '6:00 p.m.', 40.433333, -3.7),
    ('Manchester, England', '5:00 p.m.', 53.5, -2.25),
    ('Manila, Philippines', '1:00 a.m.', 14.583333, 120.95),
    ('Marseilles, France', '6:00 p.m.', 43.333333, 5.333333),
    ('Mazatlan, Mexico', '10:00 a.m.', 23.2, -106.416667),
    ('Mecca, Saudi Arabia', '8:00 p.m.', 21.483333, 39.75),
    ('Melbourne, Australia', '3:00 a.m.', -37.783333, 144.966667),
    ('Mexico City, Mexico', '11:00 a.m.', 19.433333, -99.116667),
    ('Milan, Italy', '6:00 p.m.', 45.45, 9.166667),
```

(continues on next page)

(continued from previous page)

```
    ('Montevideo, Uruguay', '2:00 p.m.', -34.883333, -56.166667),
    ('Moscow, Russia', '8:00 p.m.', 55.75, 37.6),
    ('Munich, Germany', '6:00 p.m.', 48.133333, 11.583333),
    ('Nagasaki, Japan', '2:00 a.m.', 32.8, 129.95),
    ('Nagoya, Japan', '2:00 a.m.', 35.116667, 136.933333),
    ('Nairobi, Kenya', '8:00 p.m.', -1.416667, 36.916667),
    ('Nanjing (Nanking), China', '1:00 a.m.', 32.05, 118.883333),
    ('Naples, Italy', '6:00 p.m.', 40.833333, 14.25),
    ('New Delhi, India', '10:30 p.m.', 28.583333, 77.2),
    ('Newcastle-on-Tyne, England', '5:00 p.m.', 54.966667, -1.616667),
    ('Odessa, Ukraine', '7:00 p.m.', 46.45, 30.8),
    ('Osaka, Japan', '2:00 a.m.', 34.533333, 135.5),
    ('Oslo, Norway', '6:00 p.m.', 59.95, 10.7),
    ('Panama City, Panama', '12:00 noon', 8.966667, -79.533333),
    ('Paramaribo, Suriname', '2:00 p.m.', 5.75, -55.25),
    ('Paris, France', '6:00 p.m.', 48.8, 2.333333),
    ('Perth, Australia', '1:00 a.m.', -31.95, 115.866667),
    ('Plymouth, England', '5:00 p.m.', 50.416667, -4.083333),
    ('Port Moresby, Papua New Guinea', '3:00 a.m.', -9.416667, 147.133333),
    ('Prague, Czech Republic', '6:00 p.m.', 50.083333, 14.433333),
    ('Rangoon, Myanmar', '11:30 p.m.', 16.833333, 96),
    ('Reykjavik, Iceland', '5:00 p.m.', 64.066667, -21.966667),
    ('Rio de Janeiro, Brazil', '2:00 p.m.', -22.95, -43.2),
    ('Rome, Italy', '6:00 p.m.', 41.9, 12.45),
    ('Salvador, Brazil', '2:00 p.m.', -12.933333, -38.45),
    ('Santiago, Chile', '1:00 p.m.', -33.466667, -70.75),
    ('St. Petersburg, Russia', '8:00 p.m.', 59.933333, 30.3),
    ('Sao Paulo, Brazil', '2:00 p.m.', -23.516667, -46.516667),
    ('Shanghai, China', '1:00 a.m.', 31.166667, 121.466667),
    ('Singapore, Singapore', '1:00 a.m.', 1.233333, 103.916667),
    ('Sofia, Bulgaria', '7:00 p.m.', 42.666667, 23.333333),
    ('Stockholm, Sweden', '6:00 p.m.', 59.283333, 18.05),
    ('Sydney, Australia', '3:00 a.m.', -34, 151),
    ('Tananarive, Madagascar', '8:00 p.m.', -18.833333, 47.55),
    ('Teheran, Iran', '8:30 p.m.', 35.75, 51.75),
    ('Tokyo, Japan', '2:00 a.m.', 35.666667, 139.75),
    ('Tripoli, Libya', '7:00 p.m.', 32.95, 13.2),
    ('Venice, Italy', '6:00 p.m.', 45.433333, 12.333333),
    ('Veracruz, Mexico', '11:00 a.m.', 19.166667, -96.166667),
    ('Vienna, Austria', '6:00 p.m.', 48.233333, 16.333333),
    ('Vladivostok, Russia', '3:00 a.m.', 43.166667, 132),
    ('Warsaw, Poland', '6:00 p.m.', 52.233333, 21),
    ('Wellington, New Zealand', '5:00 a.m.', -41.283333, 174.783333),
    ('Zurich, Switzerland', '6:00 p.m.', 47.35, 8.516667)
]

# Create an instance of Kml
kml = simplekml.Kml(open=1)

# Create a point named "The World" attached to the KML document with its coordinate␣
↪at 0 degrees latitude and longitude.
# All the point's properties are given when it is constructed.
single_point = kml.newpoint(name="The World", coords=[(0.0,0.0)])

# Create a point for each city. The points' properties are assigned after the point␣
↪is created
for city, time, lat, lon in cities:
```

(continues on next page)

(continued from previous page)

```
    pnt = kml.newpoint()
    pnt.name = city
    pnt.description = "Time corresponding to 12:00 noon, Eastern Standard Time: {0}".
↪format(time)
    pnt.coords = [(lon, lat)]

# Save the KML
kml.save("T00 Point.kml")
```

2.5.2 Linestring Tutorial

About

How to create a linestring. We will create different linestrings showing off the altitudemode and extrude properties.

Creating the Code

First import simplekml and create a KML object:

```
import simplekml
```

Create the KML object:

```
kml = simplekml.Kml(open=1) # the folder will be open in the table of contents
```

Next we create a simple linestring feature that lies on the ground:

```
linestring = kml.newlinestring(name="A Line")
linestring.coords = [(-122.364383,37.824664),(-122.364152,37.824322)]
```

Now to build on the previous linestring. Here we make a linestring that hovers 50m above the ground. To achieve this we give each "vertex" of the linestring (in this case the two coordinates representing the start and end of the linestring) a height value of 50m (the 50 in the two tuples). We also have to tell Google Earth that the height we gave each vertex is relative to the ground, we do this by assigning the value "relativetoground" to the atlitude property:

```
linestring = kml.newlinestring(name="A Hovering Line")
linestring.coords = [(-122.364167,37.824787,50), (-122.363917,37.824423,50)]
linestring.altitudemode = simplekml.AltitudeMode.relativetoground
```

Let's make it more interesting and join the linestring to the ground. To this all we have to do is set the property extrude to 1 to tell Google Earth the extend the linestring all the way to the ground:

```
linestring = kml.newlinestring(name="An Extended Line")
linestring.coords = [(-122.363965,37.824844,100), (-122.363747,37.824501,100)]
linestring.altitudemode = simplekml.AltitudeMode.relativetoground
linestring.extrude = 1
```

Let's go completely wild and make an extruded line climb out of the ground up to a height of 100m. To do this we simply change the first coordinates height value to zero:

```
linestring = kml.newlinestring(name="A Sloped Line")
linestring.coords = [(-122.363604,37.825009,0), (-122.363331,37.824604,100)]
```

(continues on next page)

(continued from previous page)

```
linestring.altitudemode = simplekml.AltitudeMode.relativetoground
linestring.extrude = 1
```

And finally we save the kml:

```
kml.save("T00 LineString.kml")
```

Complete Code Example

Here is the complete code:

```
import os
import simplekml

# Create an instance of Kml
kml = simplekml.Kml(open=1)

# Create a linestring with two points (ie. a line)
linestring = kml.newlinestring(name="A Line")
linestring.coords = [(-122.364383,37.824664),(-122.364152,37.824322)]

# Create a linestring that will hover 50m above the ground
linestring = kml.newlinestring(name="A Hovering Line")
linestring.coords = [(-122.364167,37.824787,50), (-122.363917,37.824423,50)]
linestring.altitudemode = simplekml.AltitudeMode.relativetoground

# Create a linestring that will hover 100m above the ground that is extended to the␣
↪ground
linestring = kml.newlinestring(name="An Extended Line")
linestring.coords = [(-122.363965,37.824844,100), (-122.363747,37.824501,100)]
linestring.altitudemode = simplekml.AltitudeMode.relativetoground
linestring.extrude = 1

# Create a linestring that will be extended to the ground but sloped from the ground␣
↪up to 100m
linestring = kml.newlinestring(name="A Sloped Line")
linestring.coords = [(-122.363604,37.825009,0), (-122.363331,37.824604,100)]
linestring.altitudemode = simplekml.AltitudeMode.relativetoground
linestring.extrude = 1

# Save the KML
kml.save(os.path.splitext(__file__)[0] + ".kml")
```

2.5.3 MultiGeometry Tutorial

About

How to use the MultiGeometry features. We will create one KML with three MultiGeometry features. We will group *simplekml.Polygon*, *simplekml.LineString* and *simplekml.Point* into the three MultiGeometry features. An additional folder will be created with normal points inside (not grouped with a MultiGeometry feature).

Background

This tutorial is creating a reference for the South African coordinate system (Hartebeesthoek 94). The system divides South Africa into vertical bands of 3 degrees each with the odd longitude as the center of each band. Each of the bands is named Lo. 19, etc. based on the longitude that is represented. What this tutorial is going to do is create a polygon representing each of these bands. Between the bands will be a vertical line showing the separation. Each of the bands will have a label with its name. In addition, points will be created with the intersection of every odd number of latitude (just to show a MultiGeometry with points).

Creating the Code

First import simplekml and create a KML object:

```
from simplekml import Kml, Color
kml = Kml(open=1)
```

Next create a variable for each of the MultiGeometry elemnts and folder:

```
multipnt = kml.newmultigeometry(name="MultiPoint") # SA (Hartebeeshoek94) Grid␣
↪Intersections
multilin = kml.newmultigeometry(name="MultiLine") # SA (Hartebeeshoek94) Lo. Lines
multipolodd = kml.newmultigeometry(name="MultiPolyOdd") # SA (Hartebeeshoek94) Lo.␣
↪Regions
multipoleven = kml.newmultigeometry(name="MultiPolyEven") # SA (Hartebeeshoek94)␣
↪Second Lo. Regions for styling
lolabels = kml.newfolder(name="Lo. Regions") # The labels of the Lo. Regions (17-33)
```

A lot is happening in the next section. There are 2 for loops that are generating all the latitude and longitude values for the shapes. These coordinates are then used to create the various features. The code will be further highlighted below, where necessary:

```
for x in range(16, 36, 2):
    linecoords = []
    if x < 34: # Label region
        lo = lolabels.newpoint(name=str(x+1), coords=[(x+1, -29)])
        lo.style.iconstyle.icon.href = "" # Remove the icons
    for y in range(-35, -19, 2):
        multipnt.newpoint(coords=[(x, y)])
        linecoords.append((x,y))
    multilin.newlinestring(coords=linecoords)
    polycoordsodd.append(linecoords)
    if len(polycoordsodd) == 2:
        end = polycoordsodd[1][:]
        end.reverse()
        multipolodd.newpolygon(outerboundaryis=polycoordsodd[0]+end)
        polycoordsodd = []
    if firstrun:
        firstrun = False
    else:
        polycoordseven.append(linecoords)
        if len(polycoordseven) == 2:
            end = polycoordseven[1][:]
            end.reverse()
            multipoleven.newpolygon(outerboundaryis=polycoordseven[0]+end)
            polycoordseven = []
```

This following section creates the points that are being used as labels for the regions. The points are added to the folder we created above (lolabels). You might be wondering why a MultiGeometry feature is not created to contain all the labels, this is because in KML all points in a MultiGeometry inherit the !MultiGeometries name for the name of the label. So, all the labels would end up being called "Lo. Regions", which is not what is wanted. The icon style's href of the points is also being made to equal "". This removes the icon completely and allows the name of the label to be centered on the origin of the point:

```
if x < 34: # Label region
    lo = lolabels.newpoint(name=str(x+1), coords=[(x+1, -29)])
    lo.style.iconstyle.icon.href = "" # Remove the icons
```

Here the longitude coordinates are generated. Now each of the intersections of the longitudes and latitudes can be drawn as a point in a MultiGeometry (multipnt). This is done by calling *newpoint* on the multipnt variable, and supplying the coordinates. After the loop finishes a new LineString is created from all the coordinates generated. this is done by calling *newlinestring* on the multilin variable:

```
for y in range(-35, -19, 2):
    multipnt.newpoint(coords=[(x, y)])
    linecoords.append((x,y))
multilin.newlinestring(coords=linecoords)
```

What the next step does is basically creates a polygon in multipolodd and multipoleven alternatively. Once all the coordinates are generated for either of the MultiGeometry collections *newpolygon* is called on the relative collection:

```
if len(polycoordsodd) == 2:
    end = polycoordsodd[1][:]
    end.reverse()
    multipolodd.newpolygon(outerboundaryis=polycoordsodd[0]+end)
    polycoordsodd = []
if firstrun:
    firstrun = False
else:
    polycoordseven.append(linecoords)
    if len(polycoordseven) == 2:
        end = polycoordseven[1][:]
        end.reverse()
```

Finally all the MultiGeometry features get styled. There a few things to note here.

- The labels' scale of the point collection is set to *0.0*. This is done to make all the labels disappear.
- The icon of the points is changed from the default pin to a circle by setting the icon styles href to the path of the circle image.
- Color is applied to the LineString (thick black).
- Color is applied to the MultiGeometry Polgon feature. Here we see the *simplekml.Color* class being utilized. The *simplekml.Color* class contains a list of named colors (from CSS and HTML). Here the orange and lightblue colors are used. The problem that occurs is that these colors are completely opaque, and makes the reference grid we are creating completely pointless, because we cannot see South Africa below the polygons. To remedy this, *simplekml.Color.changealpha()* of the *simplekml.Color* class is used. What this does is accept a Google Earth HEX string and an alpha value and returns the HEX string with the alpha value modified. It is a quick and convenient way of assigning any alpha value to the standard colors

```
multipnt.style.labelstyle.scale = 0.0 # Hide the labels of the points
multipnt.style.iconstyle.icon.href = "http://maps.google.com/mapfiles/kml/shapes/
↪placemark_circle.png"
multilin.style.linestyle.color = Color.black
```

(continues on next page)

(continued from previous page)

```
multilin.style.linestyle.width = 5
multipoleven.style.polystyle.color = Color.changealpha("77", Color.orange)
multipoleven.style.linestyle.color = Color.changealpha("77", Color.orange)
multipolodd.style.polystyle.color = Color.changealpha("77", Color.lightblue)
multipolodd.style.linestyle.color = Color.changealpha("77", Color.lightblue)
```

Complete Code Example

Here is the complete code:

```
from simplekml import Kml, Color
kml = Kml(open=1)

kml = simplekml.Kml(open=1)

# Creating MultiGeometry
multipnt = kml.newmultigeometry(name="MultiPoint") # SA (Hartebeeshoek94) Grid␣
↪Intersections
multilin = kml.newmultigeometry(name="MultiLine") # SA (Hartebeeshoek94) Lo. Lines
multipolodd = kml.newmultigeometry(name="MultiPolyOdd") # SA (Hartebeeshoek94) Lo.␣
↪Regions
multipoleven = kml.newmultigeometry(name="MultiPolyEven") # SA (Hartebeeshoek94)␣
↪Second Lo. Regions for styling
lolabels = kml.newfolder(name="Lo. Regions") # The labels of the Lo. Regions (17-33)

# Create all the coordinates to populate the South African Coordinate System
polycoordsodd = []
polycoordseven = []
firstrun = True
for x in range(16, 36, 2):
    linecoords = []
    if x < 34: # Label region
        lo = lolabels.newpoint(name=str(x+1), coords=[(x+1, -29)])
        lo.style.iconstyle.icon.href = "" # Remove the icons
    for y in range(-35, -19, 2):
        multipnt.newpoint(coords=[(x, y)])
        linecoords.append((x,y))
    multilin.newlinestring(coords=linecoords)
    polycoordsodd.append(linecoords)
    if len(polycoordsodd) == 2:
        end = polycoordsodd[1][:]
        end.reverse()
        multipolodd.newpolygon(outerboundaryis=polycoordsodd[0]+end)
        polycoordsodd = []
    if firstrun:
        firstrun = False
    else:
        polycoordseven.append(linecoords)
        if len(polycoordseven) == 2:
            end = polycoordseven[1][:]
            end.reverse()
            multipoleven.newpolygon(outerboundaryis=polycoordseven[0]+end)
            polycoordseven = []

# Style everything
```

(continues on next page)

(continued from previous page)

```
multipnt.style.labelstyle.scale = 0.0 # Hide the labels of the points
multipnt.style.iconstyle.icon.href = "http://maps.google.com/mapfiles/kml/shapes/
↪placemark_circle.png"
multilin.style.linestyle.color = Color.black
multilin.style.linestyle.width = 5
multipoleven.style.polystyle.color = Color.changealpha("77", Color.orange)
multipoleven.style.linestyle.color = Color.changealpha("77", Color.orange)
multipolodd.style.polystyle.color = Color.changealpha("77", Color.lightblue)
multipolodd.style.linestyle.color = Color.changealpha("77", Color.lightblue)

kml.save("Tut_MultiGeometry.kml")
```

2.5.4 Tour Tutorial

About

How to create a tour. We will create a KML that will reproduce the tour sample in the KML Reference with the edition of *simplekml.GxSoundCue*.

Creating the Code

First import simplekml and create a KML object:

```
import simplekml
kml = simplekml.Kml(open=1) # the document will be open in the table of contents
```

Next create a point feature and style it (note, we change the scale of the icon, this is going to be changed later during the tour:

```
pnt = kml.newpoint(name="New Zealand's Southern Alps", coords=[(170.144,-43.605)])
pnt.style.iconstyle.scale = 1.0
```

Now for the important part, creating the tour:

```
tour = kml.newgxtour(name="Play me!")
```

Once we have a tour we can create a playlist which will be contained inside of the tour:

```
playlist = tour.newgxplaylist()
```

A playlist is a collection of tour primitives (which are basically different events that happen during the tour), whose order is very important. The order that the tour primitives are added to the playlist is the order in which they play. There are five different tour primitives - *simplekml.GxAnimatedUpdate*, *simplekml.GxFlyTo*, *simplekml.GxSoundCue*, *simplekml.GxTourControl* and *simplekml.GxWait*. In the following code snippet we will create all of the tour primitives and add them to the playlist:

```
soundcue = playlist.newgxsoundcue()
soundcue.href = "http://code.google.com/p/simplekml/source/browse/samples/drum_roll_1.
↪wav"
soundcue.gxdelayedstart = 2

animatedupdate = playlist.newgxanimatedupdate(gxduration=6.5)
animatedupdate.update.change = '<IconStyle targetId="{0}"><scale>10.0</scale></
↪IconStyle>'.format(pnt.style.iconstyle.id)
```

(continues on next page)

(continued from previous page)

```
flyto = playlist.newgxflyto(gxduration=4.1)
flyto.camera.longitude = 170.157
flyto.camera.latitude = -43.671
flyto.camera.altitude = 9700
flyto.camera.heading = -6.333
flyto.camera.tilt = 33.5
flyto.camera.roll = 0

wait = playlist.newgxwait(gxduration=2.4)
```

The order in which we added the tour primitives to the playlist is important. If the *simplekml.GxSoundCue* was added after the *simplekml.GxFlyTo*, then Google Earth would wait for the *simplekml.GxFlyTo* to finish before playling the *simplekml.GxSoundCue*, but if the *simplekml.GxSoundCue* is added first it will play at the same time as the *simplekml.GxFlyTo*. It is best to have a look at the touring section of the KML Reference to familiarise yourself with what exactly is going on with tours. In this example the *simplekml.GxSoundCue* is delayed from playing by 2 second so the sound with stop playling at about the same time as the whole tour (the sound clip is about 4 seconds long).

Note: According the the KML Reference a tour needs either a *simplekml.GxFlyTo* or *simplekml.GxWait* to hold a tour open. If you just have an *simplekml.GxAnimatedUpdate* the tour plays for zero seconds in Google Earth. So, if you only want a `simplekml.GxAnimated` make sure you add a *simplekml.GxWait* tour primitive to the end of the tour with the same duration as the class:*simplekml.GxAnimated*.

And finally we save the kml:

```
kml.save("tut_tours.kml")
```

Complete Code Example

Here is the complete code:

```
import simplekml

kml = simplekml.Kml(name='9_tours', open=1)

pnt = kml.newpoint(name="New Zealand's Southern Alps", coords=[(170.144,-43.605)])
pnt.style.iconstyle.scale = 1.0

tour = kml.newgxtour(name="Play me!")
playlist = tour.newgxplaylist()

soundcue = playlist.newgxsoundcue()
soundcue.href = "http://code.google.com/p/simplekml/source/browse/samples/drum_roll_1.
↪wav"
soundcue.gxdelayedstart = 2

animatedupdate = playlist.newgxanimatedupdate(gxduration=6.5)
animatedupdate.update.change = '<IconStyle targetId="{0}"><scale>10.0</scale></
↪IconStyle>'.format(pnt.style.iconstyle.id)

flyto = playlist.newgxflyto(gxduration=4.1)
flyto.camera.longitude = 170.157
```

(continues on next page)

(continued from previous page)

```
flyto.camera.latitude = -43.671
flyto.camera.altitude = 9700
flyto.camera.heading = -6.333
flyto.camera.tilt = 33.5
flyto.camera.roll = 0

wait = playlist.newgxwait(gxduration=2.4)

kml.save("tut_tours.kml")
```

2.6 Release History

2.6.1 simplekml 1.3.4 - 02 April 2020

Fixes

- Changed pixel to pixels in class Units
- Removed a space in class RefreshMode

2.6.2 simplekml 1.3.3 - 28 January 2020

Fixes

- Changed cgi module to html for Python 3

2.6.3 simplekml 1.3.2 - 28 January 2020

Fixes

- Changed cgi module to html for Python 3

2.6.4 simplekml 1.3.1 - 08 August 2018

Changes

- Removed email from source files
- Changed PyPi classifications for Python 3 to all versions of 3
- Updated readme by removing expired websites

2.6.5 simplekml 1.3.0 - 18 March 2016

Fixes

- Made KmlElement class thread safe.
- Fixed formatting of gx:angles.

Changes

- Changed the license from GPL to LGPL

2.6.6 simplekml 1.2.8 - 07 June 2015

Fixes

- Fixed "global" styles repeating in each container.
- Fixed ampersand (&) not escaping correctly in urls.
- Fixed problem where files added via *simplekml.Kml.addfile()* were forgotten.

Changes

- Moved the method *simplekml.Kml.addfile()* from being available from all classes to being available only via *simplekml.Kml*

2.6.7 simplekml 1.2.7 - 08 February 2015

Fixes

- Fixed adding unnecessary tag (gxlabelvisibility) when it is not being used.

2.6.8 simplekml 1.2.6 - 08 February 2015

Fixes

- Fixed missing gxlabelvisibility property from *simplekml.LineStyle*.

2.6.9 simplekml 1.2.5 - 07 December 2014

Fixes

- Fixed syntax error for Python 3

2.6.10 simplekml 1.2.4 - 28 November 2014

Fixes

- Fixed missing gxvieweroptions property from *simplekml.Camera* and *simplekml.LookAt*.

2.6.11 simplekml 1.2.3 - 26 October 2013

Fixes

- Fixed unicode errors where some KML elements were causing an error when using unicode.

2.6.12 simplekml 1.2.2 - 07 June 2013

Fixes

- Changed added model to *simplekml.GxTrack*
- Added __version__ property

2.6.13 simplekml 1.2.1 - 16 December 2012

Fixes

- Changed newangle in *simplekml.GxTrack* to *simplekml.GxTrack.newgxangle()*

2.6.14 simplekml 1.2.0 - 03 December 2012

New Features

- Added a method *simplekml.Kml.addfile()*. This method adds additional files to a KMZ. Useful for adding images to the KMZ that you want to display in a description balloon.

Fixes

- Fixed documentation of coordinates where it incorrectly showed a coordinate being first latitude, then longitude, when it should have been the other way around
- Fixed paths included in a KMZ, changed backslashes to forward slashes

2.6.15 simplekml 1.1.2 - 17 September 2012

Fixes

- Fixed the import error regarding networklinkcontrol

2.6.16 simplekml 1.1.1 - 16 September 2012

New Features

- Added the property *gxballoonvisibility* to all features
- Added *simplekml.Kml.networklinkcontrol* to the *simplekml.Kml*. Thus, there is a new class called *simplekml.NetworkLinkControl* and relevant properties (including *simplekml.LinkSnippet*)

2.6.17 simplekml 1.1.0 - 09 August 2012

New Features

- Added methods to all container classes for querying features already created. The new methods are: features, allfeatures, geometries, allgeometries, containers, allcontainers, styles, allstyles, stylemaps, and allstylemaps
- Added a hint attribute to the Kml class that allows hints to be added to the kml tag, such as: *target=moon*

Enhancements

- The CDATA tags within text attributes are not escaped with the rest of the text and remain as is whether or not parsetext of the Kml class is set to True or False

Fixes

- FlyTo is now generating the Abstract View (Camera and LookAt) tag correctly

2.6.18 simplekml 1.0.0 - 24 July 2012

First production version release.

CHAPTER 3

Indices and tables

- genindex
- search

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z