pycollada Documentation

Release 0.4

Jeff Terrace

Nov 19, 2017

Contents

1 Source Code **1**

2 Tutorial **3**
- 2.1 Introduction . . . 3
- 2.2 Features . . . 3
- 2.3 Installation . . . 5
- 2.4 Loading A Collada Document . . . 5
- 2.5 Collada Object Structure . . . 6
- 2.6 Creating A Collada Object . . . 8
- 2.7 Changelog . . . 10

3 Reference **15**
- 3.1 API Summary . . . 15
- 3.2 API Reference . . . 109

4 Indices and tables **111**

Python Module Index **113**

CHAPTER 1

Source Code

The source code for pycollada lives on github

CHAPTER 2

Tutorial

2.1 Introduction

pycollada is a python module for creating, editing and loading COLLADA, which is a COLLAborative Design Activity for establishing an interchange file format for interactive 3D applications.

The library allows you to load a COLLADA file and interact with it as a python object. In addition, it supports creating a collada python object from scratch, as well as in-place editing.

pycollada uses lxml for XML loading, construction, and saving. numpy is used for numerical arrays. Both of these libraries are impleted in C/C++ which makes pycollada quite fast.

pycollada was originally written by Alejandro Conty Estevez of Scopia Visual Interfaces Systems in 2009. Since 2011, the library is now maintained by Jeff Terrace. For a list of additional contributors, see the AUTHORS file included with distribution.

2.2 Features

2.2.1 Geometry

- Triangles a set of triangles
- Polylist a set of polygons with no holes
- Polygons a set of polygons that can contain holes (holes unimplemented, currently an alias for Polylist)
- Lines a set of lines

2.2.2 Source Data

- Vertex
- Normals

- Multiple texture coordinate sets

2.2.3 Materials

- Shader types: phong, lambert, blinn, constant
- Effect attributes: emission, ambient, diffuse, specular, shininess, reflective, reflectivity, transparent, transparency
- Texture support: Can read from local file, zip archives, or a custom auxiliary file handler
- Loads texture images with PIL if available

2.2.4 Lights

- Directional
- Ambient
- Point
- Spot

2.2.5 Cameras

- Perspective

2.2.6 Scenes

- Full scene construction
- Transformations: rotate, scale, translate, matrix, lookat (for cameras)
- Supports iterating through a scene, yielding transformed geometry

2.2.7 Controllers

- Currently experimental (more support coming)
- Morph
- Skin

2.2.8 Additional Features

- Fast triangulation of polygons
- Fast computation of normals

2.3 Installation

2.3.1 github

The source code for pycollada is available on github here: https://github.com/pycollada/pycollada

To pull a read-only copy, you can clone the repository:

```
git clone git://github.com/pycollada/pycollada.git pycollada
```

2.3.2 Python Package Index

pycollada is available as a package at: http://pypi.python.org/pypi/pycollada/

You can also use easy_install:

```
easy_install pycollada
```

On Mac OS X, try this if you get an error installing lxml:

```
export ARCHFLAGS="arch i386 -arch x86_64"
easy_install pycollada
```

On Ubuntu, install these dependencies first:

```
apt-get install python-lxml python-numpy python-dateutil
easy_install pycollada
```

2.4 Loading A Collada Document

Collada documents can be loaded with the *Collada* class:

```
mesh = Collada('file.dae')
```

Zip archives are also supported. The archive will be searched for a dae file.

The constructor can also accept a file-like object:

```
f = open('file.dae')
mesh = Collada(f)
```

Note that this will also work with the *StringIO* module. When loading from non-file sources, the *aux_file_loader* parameter can be passed to the constructor. This is useful if loading from an unusual source, like a database:

```
dae_file = open('file.dae')
dae_data = dae_file.read()
texture_file = open('texture.jpg')
texture_data = texture_file.read()

def my_aux_loader(filename):
    if filename == 'texture.jpg':
        return texture_data
    return None
```

```
mesh = Collada(StringIO(dae_data), aux_file_loader=my_aux_loader)
```

When using the Collada object (see *Collada Object Structure*), if you try and read a texture, the *my_aux_loader* function will be invoked.

Loading a collada document can result in an exception being thrown. For a list of possible exceptions, see *Exceptions*. Sometimes, you may want to ignore some exceptions and let the loader try to continue loading the file. For example, the following will ignore errors about broken references and features that pycollada doesn't support:

```
mesh = Collada('file.dae', ignore=[DaeUnsupportedError, DaeBrokenRefError])
```

If any errors occurred during the load, you can find them in *Collada.errors*.

2.5 Collada Object Structure

After loading a collada document, all of the information about the file is stored within the Collada object. For example, consider the following code:

```
>>> from collada import *
>>> mesh = Collada('duck_triangles.dae')
>>> mesh
<Collada geometries=1>
```

This sample file is located in *collada/tests/data* of the pycollada distribution. We can now explore the attributes of the *Collada* class.

Let's see what *Collada.geometries* it has:

```
>>> mesh.geometries
[<Geometry id=LOD3spShape-lib, 1 primitives>]
```

Each geometry has a number of *Source* objects that contain raw source data like an array of floats. It then has a number of *Primitive* objects contained. Let's inspect them:

```
>>> geom = mesh.geometries[0]
>>> geom.primitives
[<TriangleSet length=4212>]
```

In this case, there is only a single primitive contained in the geometry and it's a set of triangles. The *TriangleSet* object lets us get at the vertex, normal, and texture coordinate information. There are index properties that index into the source arrays, and the sources are also automatically mapped for you. You can iterate over the triangle set to yield individual *Triangle* objects:

```
>>> triset = geom.primitives[0]
>>> trilist = list(triset)
>>> len(trilist)
4212
>>> trilist[0]
<Triangle ([-23.93639946  11.53530025  30.61249924], [-18.72640038  10.1079998   26.
↪6814003 ], [-15.69919968  11.42780018  34.23210144], "blinn3SG")>
```

The triangle object has the vertex, normal, and texture coordinate data associated with the triangle, as well as the material it references. Iterating over the triangle set is convenient, but it can be slow for large meshes. Instead, you can

access the numpy arrays in the set. For example, to get the vertex, normal, and texture coordinate for the first triangle in the set:

```
>>> triset.vertex[triset.vertex_index][0]
array([[-23.93639946,  11.53530025,  30.61249924],
       [-18.72640038,  10.1079998 ,  26.6814003 ],
       [-15.69919968,  11.42780018,  34.23210144]], dtype=float32)
>>> triset.normal[triset.normal_index][0]
array([[-0.192109  , -0.934569  ,  0.299458  ],
       [-0.06315   , -0.99362302,  0.093407  ],
       [-0.11695   , -0.92131299,  0.37081599]], dtype=float32)
>>> triset.texcoordset[0][triset.texcoord_indexset[0]][0]
array([[ 0.866606  ,  0.39892399],
       [ 0.87138402,  0.39761901],
       [ 0.87415999,  0.398826  ]], dtype=float32)
```

These are numpy arrays which allows for fast retrieval and computations.

The collada object also has arrays for accessing *Camera*, *Light*, *Effect*, *Material*, and *Scene* objects:

```
>>> mesh.cameras
[<Camera id=cameraShape1>]
>>> mesh.lights
[<DirectionalLight id=directionalLightShape1-lib>]
>>> mesh.effects
[<Effect id=blinn3-fx type=blinn>]
>>> mesh.materials
[<Material id=blinn3 effect=blinn3-fx>]
>>> mesh.scenes
[<Scene id=VisualSceneNode nodes=3>]
```

A collada scene is a graph that contains nodes. Each node can have transformations and a list of child nodes. A child node can be another node or an instance of a geometry, light, camera, etc. The default scene is contained in the *Collada.scene* attribute. Let's take a look:

```
>>> mesh.scene
<Scene id=VisualSceneNode nodes=3>
>>> mesh.scene.nodes
[<Node transforms=3, children=1>, <Node transforms=4, children=1>, <Node transforms=4,
↪ children=1>]
```

We could write code to iterate through the scene, applying transformations on bound objects, but the Scene object already does this for you via its *Scene.objects()* method. For example, to find all of the instantiated geometries in a scene and have them bound to a material and transformation:

```
>>> boundgeoms = list(mesh.scene.objects('geometry'))
>>> boundgeoms
[<BoundGeometry id=LOD3spShape-lib, 1 primitives>]
```

Notice that we get a *BoundGeometry* here. We can also pass in *light*, *camera*, or *controller* to get back a *BoundLight*, *BoundCamera*, or *BoundController*, respectively. The bound geometry is very similar to the geometry we looked through above. We can use the iterative method:

```
>>> boundprims = list(boundgeoms[0].primitives())
>>> boundprims
[<BoundTriangleSet length=4212>]
>>> boundtrilist = list(boundprims[0])
```

```
>>> boundtrilist[0]
<Triangle ([-23.93639946 -30.61249924  11.53530025], [-18.72640038 -26.6814003   10.
↪1079998 ], [-15.69919968 -34.23210144  11.42780018], "<Material id=blinn3␣
↪effect=blinn3-fx>")>
```

or by accessing the numpy arrays directly:

```
>>> boundprims[0].vertex[boundprims[0].vertex_index][0]
array([[-23.93639946, -30.61249924,  11.53530025],
       [-18.72640038, -26.6814003 ,  10.1079998 ],
       [-15.69919968, -34.23210144,  11.42780018]], dtype=float32)
```

In this case, the triangle is identical to above. This is because the collada duck example only has identity transformations. We can inspect these in the scene:

```
>>> mesh.scene.nodes[0].transforms
[<RotateTransform (0.0, 0.0, 1.0) angle=0.0>, <RotateTransform (0.0, 1.0, 0.0)␣
↪angle=0.0>, <RotateTransform (1.0, 0.0, 0.0) angle=0.0>]
>>> mesh.scene.nodes[0].children
[<GeometryNode geometry=LOD3spShape-lib>]
```

2.6 Creating A Collada Object

In this section, we outline how to create a collada document from scratch. First, let's create an empy collada document:

```
>>> from collada import *
>>> mesh = Collada()
```

We could save this out, but it would be completely blank. Let's first add a *Material* to the document:

```
>>> mesh = Collada()
>>> effect = material.Effect("effect0", [], "phong", diffuse=(1,0,0), specular=(0,1,
↪0))
>>> mat = material.Material("material0", "mymaterial", effect)
>>> mesh.effects.append(effect)
>>> mesh.materials.append(mat)
```

Note that the second argument to *Effect* is for parameters. These are used for textures. We omit textures for simplicity here.

Next, let's first create some source arrays. These are going to be used to create a triangle set later:

```
>>> import numpy
>>> vert_floats = [-50,50,50,50,50,50,-50,-50,50,50,
...     -50,50,-50,50,-50,50,50,-50,-50,-50,-50,50,-50,-50]
>>> normal_floats = [0,0,1,0,0,1,0,0,1,0,0,1,0,1,0,
...     0,1,0,0,1,0,0,1,0,0,-1,0,0,-1,0,0,-1,0,0,-1,0,0,
...     -1,0,0,-1,0,0,-1,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0,0,-1,
...     0,0,-1,0,0,-1,0,0,-1]
>>> vert_src = source.FloatSource("cubeverts-array", numpy.array(vert_floats), ('X',
↪'Y', 'Z'))
>>> normal_src = source.FloatSource("cubenormals-array", numpy.array(normal_floats), (
↪'X', 'Y', 'Z'))
```

Now that we have some sources, let's create a *Geometry* and add the sources to it:

```
>>> geom = geometry.Geometry(mesh, "geometry0", "mycube", [vert_src, normal_src])
```

To add a triangle set to the geometry, we can call the *Geometry.createTriangleSet()* method. To do this, we need to define the inputs to the triangle set. In this case, we are going to input the arrays we previously defined:

```
>>> input_list = source.InputList()
>>> input_list.addInput(0, 'VERTEX', "#cubeverts-array")
>>> input_list.addInput(1, 'NORMAL', "#cubenormals-array")
```

This says to use the source with identifier *cubeverts-array* as the vertex source and source with identifier *cubenormals-array* as the normal source. The offsets indicate that the vertex data is the first offset in the index array and the normal data is the second offset in the index array. Let's now create the index array:

```
>>> indices = numpy.array([0,0,2,1,3,2,0,0,3,2,1,3,0,4,1,5,5,6,0,
...     4,5,6,4,7,6,8,7,9,3,10,6,8,3,10,2,11,0,12,
...     4,13,6,14,0,12,6,14,2,15,3,16,7,17,5,18,3,
...     16,5,18,1,19,5,20,7,21,6,22,5,20,6,22,4,23])
```

Now that we have an index array, an input list, and a material, we can create a triangle set and add it to the geometry's list of primitives. We then add it to the list of geometries in the mesh:

```
>>> triset = geom.createTriangleSet(indices, input_list, "materialref")
>>> geom.primitives.append(triset)
>>> mesh.geometries.append(geom)
```

We now have everything we need in the object except for a scene. To get the geometry to show up, we have to create a scene. First, we instantiate the geometry into a scene node, mapping it to a material:

```
>>> matnode = scene.MaterialNode("materialref", mat, inputs=[])
>>> geomnode = scene.GeometryNode(geom, [matnode])
>>> node = scene.Node("node0", children=[geomnode])
```

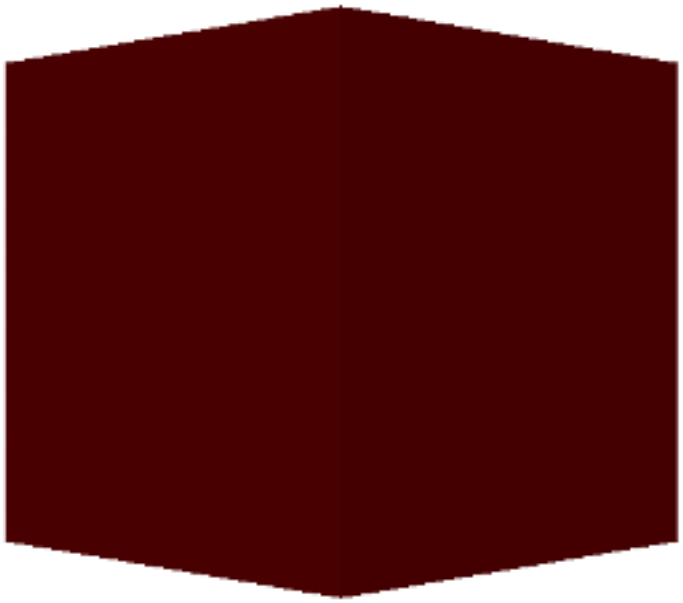
Now that we have the scene node, we can create a scene, add the node to the scene, add the scene to the document, and then set our scene as the default:

```
>>> myscene = scene.Scene("myscene", [node])
>>> mesh.scenes.append(myscene)
>>> mesh.scene = myscene
```

We can now save the document to a file:

```
>>> mesh.write('/tmp/test.dae')
```

If you load this file, it should look like a red cube. Here's a screenshot:



2.7 Changelog

2.7.1 0.4 (2012-07-31)

Backwards Compatibility Notes

- Python 2.5 is no longer supported. Supported versions are now 2.6, 2.7 and 3.2.

New Features

- Added support for reading the opaque attribute from <transparent> tag.
- Normals and texture coordinate indices are now available in shapes (Triangle and Polygon).
- Library is now compatible with python's built-in ElementTree API instead of requiring lxml. lxml is still recommended.
- Added support for Python 3.2. Supported versions are now 2.6, 2.7 and 3.2.
- Added support for index_of_refraction in <effect>.
- Added optional parameter to Collada that does XML schema validation when saving.
- Automatically corrects broken files that don't have correct xfov, yfov, and aspect ratio in cameras.

Bug Fixes

- Fix the default value for transparency in Effect. Now correctly defaults to 1.0 when opaque mode is A_ONE, and 0.0 when opaque mode is RGB_ZERO.
- Fixed bug where BoundPolylist was not returning the correct length value.

- Removed support for RGB from Effect since it's not valid in the spec. If an RGB is given, a fourth A channel is automatically added as 1.0.
- Made instance_geometry not write an empty bind_material if it's empty since it breaks validation.
- Made saving strip out empty <library_*> tags since it breaks validation.

2.7.2 0.3 (2011-08-31)

Backwards Compatibility Notes

- If using the old Camera object, this has been changed to an abstract class with types for PerspectiveCamera and OrthographicCamera
- If using the old Collada.assetInfo dictionary to read asset information, this has been changed to an object. See documentation for more information.

New Features

- Added support for bump maps inside the extra tag of an effect
- Added texbinormal and textangent to triangle sets
- Added a method to generate texture tangents and binormals
- Added detection for double_sided
- Added an optional parameter to specify what filename inside an archive to use when loading from zip
- Added support for loading multiple sets of library_* nodes
- Refactored asset information into a separate module. Fixed #12
- Refactored Camera into PerspectiveCamera and OrthographicCamera, inheriting from Camera

Bug Fixes

- Changed Collada IndexedLists attributes to be properties. Fixed Issue #14
- Updated scene to use a local scope when nodes are instantiated inside a scene
- Changed parsing to raise DaeMalformedError when an lxml parser exception is thrown
- Fixed bug when loading an <image> tag local to an <effect> not showing up in Collada.images
- Fixed bug when loading an empty <polygons>
- Fixed bug in if statement when loading morph controllers
- Fixed bug when triangulating a length-0 polylist
- Updated install instructions for OS X and Ubuntu problems
- Fixed bugs in IndexedList from Issue #13
- Fixed a bug where using the same map twice in an effect would cause incorrrect output
- Changed geometry export to delete any sources in the vertices tag that no longer exist
- Changed library output to not output emtpy library nodes so validator doesn't complain
- Add same checks in scene loading that was done in library_nodes loading so that if nodes are not found yet while loading, it will keep trying

- Changed the way library_nodes is loaded so that if a referenced node from instance_node is not loaded yet, it will keep trying
- Fixed bug where a triangles xml node would try to set an attribute to None
- Fixed bug in handling joints that influence 0 vertices

2.7.3 0.2.2 (2011-05-03)

- Changed the way instance_node is handled to actually maintain the mapping so it's not lost when saving
- Added setdata function to CImage and made Effect compare only image path
- Fixed a bug when rewriting geometry sources
- Change primitive sources to point to the <vertices> tag when possible since other importers don't like not having a <vertices> tag
- Export source data with only 7 decimal precision for better file size
- Prevent NaN from being the result of a normalize_v3 call
- Fixed bug where effect was not correctly reading all four color values
- Fixed a bug where a triangleset would not create its xml node when generated from a polylist
- Big speed increases for converting numpy data to strings
- Moved getInputs function to Primitive
- Added functions to triangleset to generate normals and get an input list
- Fixed bug in saving a scene node if there was no id
- Fixed some bugs/optimizations with saving
- Added function to test if an Effect is almost equal to another Effect
- Adding dynamic dependencies to setup.py

2.7.4 0.2.1 (2011-04-15)

- Fixed bug with saving existing files that didn't have some library_ tags.

2.7.5 0.2 (2011-04-15)

- Many bugfixes
- polylist support
- polygons support without holes
- lines support
- blinn and constant material support
- More effect attributes
- Better support for auxiliary texture files
- Lights (directional, ambient, point, spot)
- lookat transform

- Experimental controller support (skin, morph)
- polygons/polylist can be triangulated
- Automatic computation of per-vertex normals

2.7.6 0.1 (2009-02-08)

- Initial release
- Triangles geometry
- Reads vertices and normals
- Multiple texture coordinate channels
- Phong and Lambert Materials
- Texture support using PIL
- Scene suppport for geometry, material and camera instances
- Transforms (matrix, rotate, scale, translate)

CHAPTER 3

Reference

3.1 API Summary

3.1.1 Main

collada.Collada	This is the main class used to create and load collada documents
collada.common.DaeObject	This class is the abstract interface to all collada objects.

collada.Collada

class collada.**Collada**(*filename=None*, *ignore=None*, *aux_file_loader=None*, *zip_filename=None*, *validate_output=False*)

Bases: object

This is the main class used to create and load collada documents

__init__(*filename=None*, *ignore=None*, *aux_file_loader=None*, *zip_filename=None*, *validate_output=False*)

Load collada data from filename or file like object.

Parameters

- **filename** – String containing path to filename to open or file-like object. Uncompressed .dae files are supported, as well as zip file archives. If this is set to None, a new collada instance is created.
- **ignore** (*list*) – A list of *common.DaeError* types that should be ignored when loading the collada document. Instances of these types will be added to *errors* after loading but won't be raised. Only used if *filename* is not None.
- **aux_file_loader** (*function*) – Referenced files (e.g. texture images) are loaded from disk when reading from the local filesystem and from the zip archive when loading from a zip file. If these files are coming from another source (e.g. database) and/or you're

loading with StringIO, set this to a function that given a filename, returns the binary data in the file. If *filename* is `None`, you must set this parameter if you want to load auxiliary files.

- **zip_filename** (*str*) – If the file being loaded is a zip archive, you can set this parameter to indicate the file within the archive that should be loaded. If not set, a file that ends with .dae will be searched.
- **validate_output** (*bool*) – If set to True, the XML written when calling *save()* will be validated against the COLLADA 1.4.1 schema. If validation fails, the *common.DaeSaveValidationError* exception will be thrown.

Methods

__init__([filename, ignore, ...])	Load collada data from filename or file like object.
handleError(error)	
ignoreErrors(*args)	Add exceptions to the mask for ignoring or clear the mask if None given.
save()	Saves the collada document back to *xmlnode*

geometries
A list of *collada.geometry.Geometry* objects. Can also be indexed by id

controllers
A list of *collada.controller.Controller* objects. Can also be indexed by id

animations
A list of `collada.animation.Animation` objects. Can also be indexed by id

lights
A list of *collada.light.Light* objects. Can also be indexed by id

cameras
A list of *collada.camera.Camera* objects. Can also be indexed by id

images
A list of *collada.material.CImage* objects. Can also be indexed by id

effects
A list of *collada.material.Effect* objects. Can also be indexed by id

materials
A list of *collada.material.Effect* objects. Can also be indexed by id

nodes
A list of *collada.scene.Node* objects. Can also be indexed by id

scenes
A list of *collada.scene.Scene* objects. Can also be indexed by id

errors = None
List of `common.common.DaeError` objects representing errors encountered while loading collada file

scene = None
The default scene. This is either an instance of *collada.scene.Scene* or *None*.

assetInfo = None
Instance of *collada.asset.Asset* containing asset information

xmlnode = None
ElementTree representation of the collada document

ignoreErrors(**args*)
Add exceptions to the mask for ignoring or clear the mask if None given.

You call c.ignoreErrors(e1, e2, ...) if you want the loader to ignore those exceptions and continue loading whatever it can. If you want to empty the mask so all exceptions abort the load just call c.ignoreErrors(None).

save()
Saves the collada document back to *xmlnode*

write(*fp*)
Writes out the collada document to a file. Note that this also calls *save()* so avoid calling both methods to save performance.

Parameters **file** – Either the file name to write to or a file-like object

collada.common.DaeObject

class collada.common.**DaeObject**
Bases: object

This class is the abstract interface to all collada objects.

Every <tag> in a COLLADA that we recognize and load has mirror class deriving from this one. All instances will have at least a *load()* method which creates the object from an xml node and an attribute called *xmlnode* with the ElementTree representation of the data. Even if it was created on the fly. If the object is not read-only, it will also have a *save()* method which saves the object's information back to the *xmlnode* attribute.

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

load(collada, localscope, node)	Load and return a class instance from an XML node.
save()	Put all the data to the internal xml node (xmlnode) so it can be serialized.

xmlnode = None
ElementTree representation of the data.

static load(*collada*, *localscope*, *node*)
Load and return a class instance from an XML node.

Inspect the data inside node, which must match this class tag and create an instance out of it.

Parameters

- **collada** (collada.Collada) – The collada file object where this object lives
- **localscope** (*dict*) – If there is a local scope where we should look for local ids (sid) this is the dictionary. Otherwise empty dict ({})
- **node** – An Element from python's ElementTree API

save()
Put all the data to the internal xml node (xmlnode) so it can be serialized.

3.1.2 Asset

collada.asset.Contributor	Defines authoring information for asset management
collada.asset.Asset	Defines asset-management information
collada.asset.UP_AXIS	The up-axis of the collada document.

collada.asset.Contributor

class collada.asset.**Contributor**(*author=None*, *authoring_tool=None*, *comments=None*, *copyright=None*, *source_data=None*, *xmlnode=None*)
Bases: *collada.common.DaeObject*

Defines authoring information for asset management

__init__(*author=None*, *authoring_tool=None*, *comments=None*, *copyright=None*, *source_data=None*, *xmlnode=None*)
Create a new contributor

Parameters

- **author** (*str*) – The author's name
- **authoring_tool** (*str*) – Name of the authoring tool
- **comments** (*str*) – Comments from the contributor
- **copyright** (*str*) – Copyright information
- **source_data** (*str*) – URI referencing the source data
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__([author, authoring_tool, comments, ...])	Create a new contributor
load(collada, localscope, node)	
save()	Saves the contributor info back to *xmlnode*

author = None
Contains a string with the author's name.

authoring_tool = None
Contains a string with the name of the authoring tool.

comments = None
Contains a string with comments from this contributor.

copyright = None
Contains a string with copyright information.

source_data = None
Contains a string with a URI referencing the source data for this asset.

xmlnode = None
ElementTree representation of the contributor.

save()
Saves the contributor info back to *xmlnode*

collada.asset.Asset

class collada.asset.**Asset**(*created=None*, *modified=None*, *title=None*, *subject=None*, *revision=None*, *keywords=None*, *unitname=None*, *unitmeter=None*, *upaxis=None*, *contributors=None*, *xmlnode=None*)
Bases: *collada.common.DaeObject*

Defines asset-management information

__init__(*created=None*, *modified=None*, *title=None*, *subject=None*, *revision=None*, *keywords=None*, *unitname=None*, *unitmeter=None*, *upaxis=None*, *contributors=None*, *xmlnode=None*)
Create a new set of information about an asset

Parameters

- **created** (*datetime.datetime*) – When the asset was created. If None, this will be set to the current date and time.
- **modified** (*datetime.datetime*) – When the asset was modified. If None, this will be set to the current date and time.
- **title** (*str*) – The title of the asset
- **subject** (*str*) – The description of the topical subject of the asset
- **revision** (*str*) – Revision information about the asset
- **keywords** (*str*) – A list of words used for search criteria for the asset
- **unitname** (*str*) – The name of the unit of distance for this asset
- **unitmeter** (*float*) – How many real-world meters are in one distance unit
- **upaxis** (collada.asset.UP_AXIS) – The up-axis of the asset. If None, this will be set to Y_UP
- **contributors** (*list*) – The list of contributors for the asset
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__([created, modified, title, ...])	Create a new set of information about an asset
load(collada, localscope, node)	
save()	Saves the asset info back to *xmlnode*

created = None
Instance of datetime.datetime indicating when the asset was created

modified = None
Instance of datetime.datetime indicating when the asset was modified

title = None

String containing the title of the asset

subject = None
String containing the description of the topical subject of the asset

revision = None
String containing revision information about the asset

keywords = None
String containing a list of words used for search criteria for the asset

unitname = None
String containing the name of the unit of distance for this asset

unitmeter = None
Float containing how many real-world meters are in one distance unit

upaxis = None
Instance of type *collada.asset.UP_AXIS* indicating the up-axis of the asset

contributors = None
A list of instances of *collada.asset.Contributor*

xmlnode = None
ElementTree representation of the asset.

save()
Saves the asset info back to *xmlnode*

collada.asset.UP_AXIS

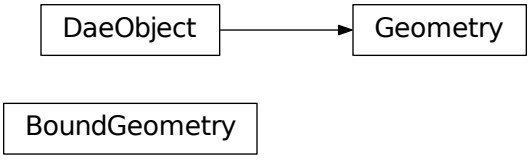
class collada.asset.**UP_AXIS**
The up-axis of the collada document.

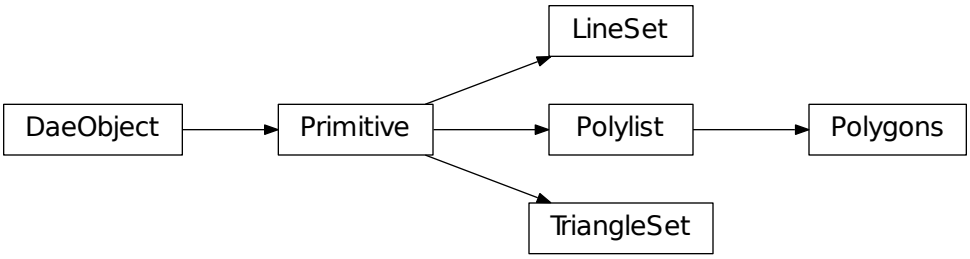
X_UP = 'X_UP'
Indicates X direction is up

Y_UP = 'Y_UP'
Indicates Y direction is up

Z_UP = 'Z_UP'
Indicates Z direction is up

3.1.3 Geometry





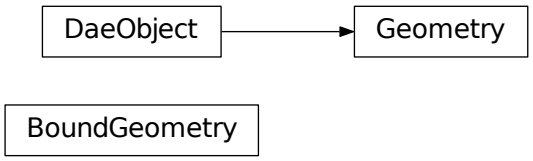
collada.geometry.Geometry	A class containing the data coming from a COLLADA <geometry> tag
collada.primitive.Primitive	Base class for all primitive sets like TriangleSet, LineSet, Polylist, etc.
collada.triangleset.TriangleSet	Class containing the data COLLADA puts in a <triangles> tag, a collection of triangles.
collada.lineset.LineSet	Class containing the data COLLADA puts in a <lines> tag, a collection of lines.
collada.polylist.Polylist	Class containing the data COLLADA puts in a <polylist> tag, a collection of polygons.
collada.polygons.Polygons	Class containing the data COLLADA puts in a <polygons> tag, a collection of polygons that can have holes.

collada.geometry.Geometry

class collada.geometry.**Geometry**(*collada*, *id*, *name*, *sourcebyid*, *primitives=None*, *xmlnode=None*, *double_sided=False*)

Bases: *collada.common.DaeObject*

A class containing the data coming from a COLLADA <geometry> tag



__init__(*collada*, *id*, *name*, *sourcebyid*, *primitives=None*, *xmlnode=None*, *double_sided=False*)

Create a geometry instance

Parameters

- **collada** (collada.Collada) – The collada object this geometry belongs to

- **id** (*str*) – A unique string identifier for the geometry
- **name** (*str*) – A text string naming the geometry
- **sourcebyid** – A list of *collada.source.Source* objects or a dictionary mapping source ids to the actual objects
- **primitives** (*list*) – List of primitive objects contained within the geometry. Do not set this argument manually. Instead, create a *collada.geometry.Geometry* first and then append to *primitives* with the *create** functions.
- **xmlnode** – When loaded, the xmlnode it comes from.
- **double_sided** (*bool*) – Whether or not the geometry should be rendered double sided

Methods

__init__(collada, id, name, sourcebyid[, ...])	Create a geometry instance
bind(matrix, materialnodebysymbol)	Binds this geometry to a transform matrix and material mapping.
createLineSet(indices, inputlist, materialid)	Create a set of lines for use in this geometry instance.
createPolylist(indices, vcounts, inputlist, ...)	Create a polylist for use with this geometry instance.
createPolygons(indices, inputlist, materialid)	Create a polygons for use with this geometry instance.
createTriangleSet(indices, inputlist, materialid)	Create a set of triangles for use in this geometry instance.
load(collada, localscope, node)	
save()	Saves the geometry back to *xmlnode*

collada = None
The *collada.Collada* object this geometry belongs to

id = None
The unique string identifier for the geometry

name = None
The text string naming the geometry

double_sided = None
A boolean indicating whether or not the geometry should be rendered double sided

sourceById = None
A dictionary containing *collada.source.Source* objects indexed by their id.

primitives = None
List of primitives (base type *collada.primitive.Primitive*) inside this geometry.

xmlnode = None
ElementTree representation of the geometry.

createLineSet(*indices*, *inputlist*, *materialid*)
Create a set of lines for use in this geometry instance.

Parameters

- **indices** (*numpy.array*) – unshaped numpy array that contains the indices for the inputs referenced in inputlist
- **inputlist** (collada.source.InputList) – The inputs for this primitive

- **materialid** (*str*) – A string containing a symbol that will get used to bind this lineset to a material when instantiating into a scene

Return type *collada.lineset.LineSet*

createTriangleSet(*indices*, *inputlist*, *materialid*)
Create a set of triangles for use in this geometry instance.

Parameters

- **indices** (*numpy.array*) – unshaped numpy array that contains the indices for the inputs referenced in inputlist
- **inputlist** (collada.source.InputList) – The inputs for this primitive
- **materialid** (*str*) – A string containing a symbol that will get used to bind this triangleset to a material when instantiating into a scene

Return type *collada.triangleset.TriangleSet*

createPolylist(*indices*, *vcounts*, *inputlist*, *materialid*)
Create a polylist for use with this geometry instance.

Parameters

- **indices** (*numpy.array*) – unshaped numpy array that contains the indices for the inputs referenced in inputlist
- **vcounts** (*numpy.array*) – unshaped numpy array that contains the vertex count for each polygon in this polylist
- **inputlist** (collada.source.InputList) – The inputs for this primitive
- **materialid** (*str*) – A string containing a symbol that will get used to bind this polylist to a material when instantiating into a scene

Return type *collada.polylist.Polylist*

createPolygons(*indices*, *inputlist*, *materialid*)
Create a polygons for use with this geometry instance.

Parameters

- **indices** (*numpy.array*) – list of unshaped numpy arrays that each contain the indices for a single polygon
- **inputlist** (collada.source.InputList) – The inputs for this primitive
- **materialid** (*str*) – A string containing a symbol that will get used to bind this polygons to a material when instantiating into a scene

Return type *collada.polygons.Polygons*

save()
Saves the geometry back to *xmlnode*

bind(*matrix*, *materialnodebysymbol*)
Binds this geometry to a transform matrix and material mapping. The geometry's points get transformed by the given matrix and its inputs get mapped to the given materials.

Parameters

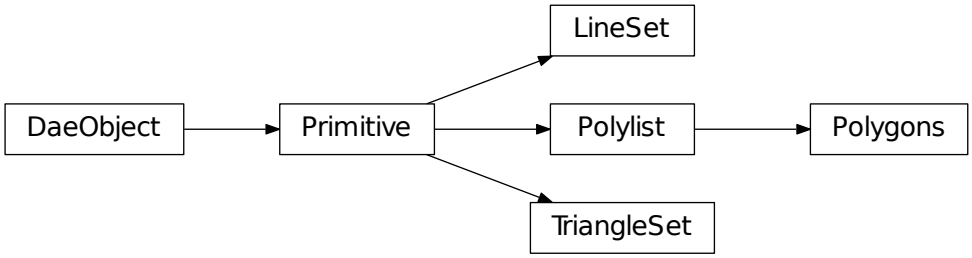
- **matrix** (*numpy.array*) – A 4x4 numpy float matrix
- **materialnodebysymbol** (*dict*) – A dictionary with the material symbols inside the primitive assigned to *collada.scene.MaterialNode* defined in the scene

Return type *collada.geometry.BoundGeometry*

collada.primitive.Primitive

class collada.primitive.**Primitive**
Bases: *collada.common.DaeObject*

Base class for all primitive sets like TriangleSet, LineSet, Polylist, etc.



__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

bind(matrix, materialnodebysymbol)	Binds this primitive to a transform matrix and material mapping.
load(collada, localscope, node)	Load and return a class instance from an XML node.
save()	

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.
vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array.

textangentset
Read-only tuple of texture tangent arrays. Each value is a numpy.array of size Nx3 where N is the number of texture tangents in the primitive's source array.

texbinormalset
Read-only tuple of texture binormal arrays. Each value is a numpy.array of size Nx3 where N is the number of texture binormals in the primitive's source array.

vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`.

texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates.

textangent_indexset
Read-only tuple of texture tangent index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture tangents, one can use the array to select into the textangentset array, e.g. `textangentset[0][textangent_indexset[0]]` would select the first set of texture tangents.

texbinormal_indexset
Read-only tuple of texture binormal index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture binormals, one can use the array to select into the texbinormalset array, e.g. `texbinormalset[0][texbinormal_indexset[0]]` would select the first set of texture binormals.

bind(*matrix*, *materialnodebysymbol*)
Binds this primitive to a transform matrix and material mapping. The primitive's points get transformed by the given matrix and its inputs get mapped to the given materials.

Parameters

- **matrix** (*numpy.array*) – A 4x4 numpy float matrix
- **materialnodebysymbol** (*dict*) – A dictionary with the material symbols inside the primitive assigned to *collada.scene.MaterialNode* defined in the scene

Return type *collada.primitive.Primitive*

getInputList()
Gets a *collada.source.InputList* representing the inputs from a primitive

load(*collada*, *localscope*, *node*)
Load and return a class instance from an XML node.

Inspect the data inside node, which must match this class tag and create an instance out of it.

Parameters

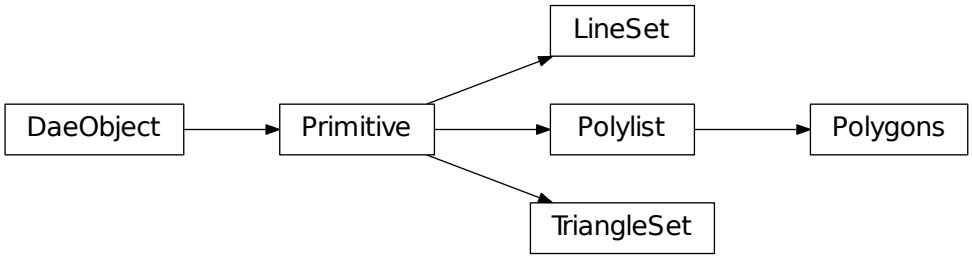
- **collada** (collada.Collada) – The collada file object where this object lives
- **localscope** (*dict*) – If there is a local scope where we should look for local ids (sid) this is the dictionary. Otherwise empty dict ({})
- **node** – An Element from python's ElementTree API

collada.triangleset.TriangleSet

class collada.triangleset.**TriangleSet**(*sources*, *material*, *index*, *xmlnode=None*)
Bases: *collada.primitive.Primitive*

Class containing the data COLLADA puts in a <triangles> tag, a collection of triangles.

- The TriangleSet object is read-only. To modify a TriangleSet, create a new instance using *collada.geometry.Geometry.createTriangleSet()*.
- If T is an instance of *collada.triangleset.TriangleSet*, then len(T) returns the number of triangles in the set. T[i] returns the i[th] triangle in the set.



__init__(*sources*, *material*, *index*, *xmlnode=None*)
A TriangleSet should not be created manually. Instead, call the *collada.geometry.Geometry.createTriangleSet()* method after creating a geometry instance.

Methods

__init__(sources, material, index[, xmlnode])	A TriangleSet should not be created manually.
bind(matrix, materialnodebysymbol)	Create a bound triangle set from this triangle set, transform and material mapping
load(collada, localscope, node)	
save()	

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.
vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

bind(*matrix*, *materialnodebysymbol*)
Create a bound triangle set from this triangle set, transform and material mapping

generateNormals()
If `normals` is *None* or you wish for normals to be recomputed, call this method to recompute them.

generateTexTangentsAndBinormals()
If there are no texture tangents, this method will compute them. Texture coordinates must exist and it uses the first texture coordinate set.

getInputList()
Gets a *collada.source.InputList* representing the inputs from a primitive

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`.

texbinormal_indexset
Read-only tuple of texture binormal index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture binormals, one can use the array to select into the texbinormalset array, e.g. `texbinormalset[0][texbinormal_indexset[0]]` would select the first set of texture binormals.

texbinormalset
Read-only tuple of texture binormal arrays. Each value is a numpy.array of size Nx3 where N is the number of texture binormals in the primitive's source array.

texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array.

textangent_indexset
Read-only tuple of texture tangent index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture tangents, one can use the array to select into the textangentset array, e.g. `textangentset[0][textangent_indexset[0]]` would select the first set of texture tangents.

textangentset
Read-only tuple of texture tangent arrays. Each value is a numpy.array of size Nx3 where N is the number of texture tangents in the primitive's source array.

vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.

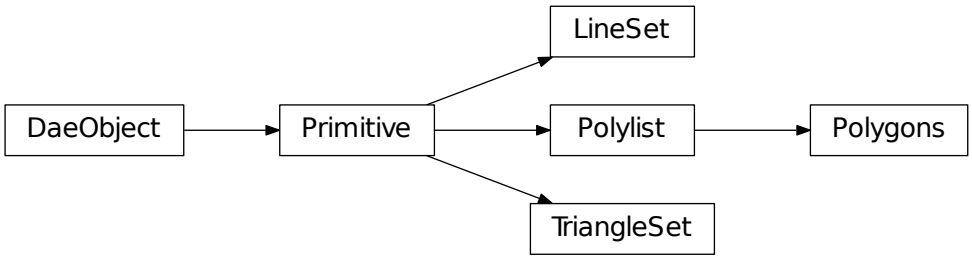
vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`.

collada.lineset.LineSet

class `collada.lineset.`**LineSet**(*sources*, *material*, *index*, *xmlnode=None*)
Bases: *collada.primitive.Primitive*

Class containing the data COLLADA puts in a <lines> tag, a collection of lines. The LineSet object is read-only. To modify a LineSet, create a new instance using *collada.geometry.Geometry.createLineSet()*.

- If `L` is an instance of *collada.lineset.LineSet*, then `len(L)` returns the number of lines in the set. `L[i]` returns the i^{th} line in the set.



__init__(*sources*, *material*, *index*, *xmlnode=None*)
A LineSet should not be created manually. Instead, call the *collada.geometry.Geometry.createLineSet()* method after creating a geometry instance.

Methods

__init__(sources, material, index[, xmlnode])	A LineSet should not be created manually.
bind(matrix, materialnodebysymbol)	Create a bound line set from this line set, transform and material mapping

Continued on next page

Table 3.13 – continued from previous page

`load`(collada, localscope, node)	
`save`()	

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.
vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

xmlnode = None
ElementTree representation of the line set.

bind(*matrix*, *materialnodebysymbol*)
Create a bound line set from this line set, transform and material mapping

getInputList()
Gets a *collada.source.InputList* representing the inputs from a primitive

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`.

texbinormal_indexset
Read-only tuple of texture binormal index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture binormals, one can use the array to select into the texbinormalset array, e.g. `texbinormalset[0][texbinormal_indexset[0]]` would select the first set of texture binormals.

texbinormalset
Read-only tuple of texture binormal arrays. Each value is a numpy.array of size Nx3 where N is the number of texture binormals in the primitive's source array.

texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array.

textangent_indexset
Read-only tuple of texture tangent index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture tangents, one can use the array to select into the textangentset array, e.g. `textangentset[0][textangent_indexset[0]]` would select the first set of texture tangents.

textangentset
Read-only tuple of texture tangent arrays. Each value is a numpy.array of size Nx3 where N is the number of texture tangents in the primitive's source array.

vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.

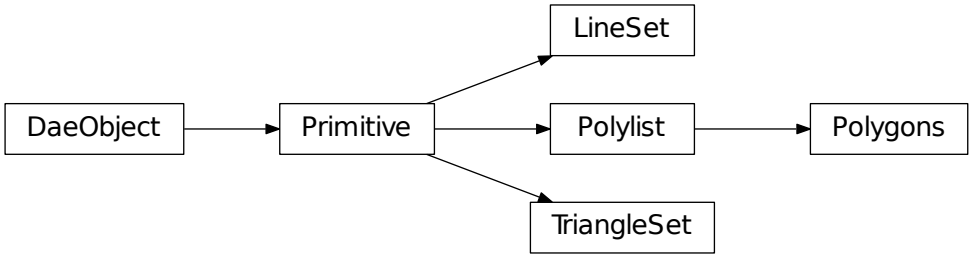
vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`.

collada.polylist.Polylist

class `collada.polylist.`**Polylist**(*sources*, *material*, *index*, *vcounts*, *xmlnode=None*)
Bases: *collada.primitive.Primitive*

Class containing the data COLLADA puts in a <polylist> tag, a collection of polygons. The Polylist object is read-only. To modify a Polylist, create a new instance using *collada.geometry.Geometry.createPolylist()*.

- If `P` is an instance of *collada.polylist.Polylist*, then `len(P)` returns the number of polygons in the set. `P[i]` returns the i^{th} polygon in the set.



__init__(*sources*, *material*, *index*, *vcounts*, *xmlnode=None*)
A Polylist should not be created manually. Instead, call the *collada.geometry.Geometry.createPolylist()* method after creating a geometry instance.

Methods

__init__(sources, material, index, vcounts)	A Polylist should not be created manually.

Continued on next page

Table 3.15 – continued from previous page

bind(matrix, materialnodebysymbol)	Create a bound polylist from this polylist, transform and material mapping
load(collada, localscope, node)	
save()	
triangleset()	This performs a simple triangulation of the polylist using the fanning method.

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.
vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

xmlnode = None
ElementTree representation of the line set.

triangleset()
This performs a simple triangulation of the polylist using the fanning method.

Return type *collada.triangleset.TriangleSet*

bind(*matrix*, *materialnodebysymbol*)
Create a bound polylist from this polylist, transform and material mapping

getInputList()
Gets a *collada.source.InputList* representing the inputs from a primitive

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`.

texbinormal_indexset
Read-only tuple of texture binormal index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture binormals, one can use the array to select into the texbinormalset array, e.g. `texbinormalset[0][texbinormal_indexset[0]]` would select the first set of texture binormals.

texbinormalset
Read-only tuple of texture binormal arrays. Each value is a numpy.array of size Nx3 where N is the number of texture binormals in the primitive's source array.

texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array.

textangent_indexset
Read-only tuple of texture tangent index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture tangents, one can use the array to select into the textangentset array, e.g. `textangentset[0][textangent_indexset[0]]` would select the first set of texture tangents.

textangentset
Read-only tuple of texture tangent arrays. Each value is a numpy.array of size Nx3 where N is the number of texture tangents in the primitive's source array.

vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.

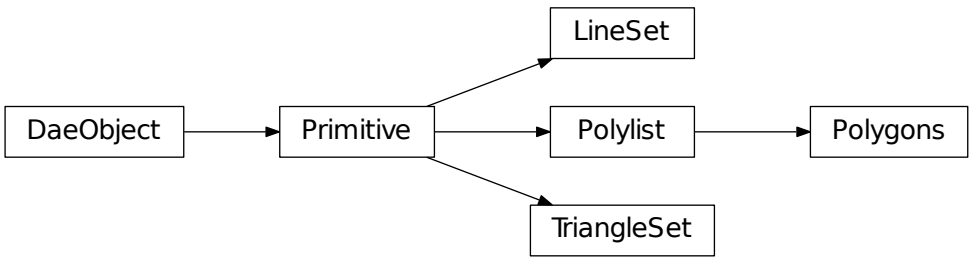
vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`.

collada.polygons.Polygons

class `collada.polygons.`**Polygons**(*sources*, *material*, *polygons*, *xmlnode=None*)
Bases: *collada.polylist.Polylist*

Class containing the data COLLADA puts in a <polygons> tag, a collection of polygons that can have holes.

- The Polygons object is read-only. To modify a Polygons, create a new instance using *collada.geometry.Geometry.createPolygons()*.
- Polygons with holes are not currently supported, so for right now, this class is essentially the same as a *collada.polylist.Polylist*. Use a polylist instead if your polygons don't have holes.



__init__(*sources*, *material*, *polygons*, *xmlnode=None*)
A Polygons should not be created manually. Instead, call the *collada.geometry.Geometry.createPolygons()* method after creating a geometry instance.

Methods

__init__(sources, material, polygons[, xmlnode])	A Polygons should not be created manually.
bind(matrix, materialnodebysymbol)	Create a bound polygons from this polygons, transform and material mapping
load(collada, localscope, node)	
save()	
triangleset()	This performs a simple triangulation of the polylist using the fanning method.

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.
vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

bind(*matrix*, *materialnodebysymbol*)
Create a bound polygons from this polygons, transform and material mapping

getInputList()
Gets a *collada.source.InputList* representing the inputs from a primitive

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`.

texbinormal_indexset
Read-only tuple of texture binormal index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture binormals, one can use the array to select into the texbinormalset array, e.g. `texbinormalset[0][texbinormal_indexset[0]]` would select the first set of texture binormals.

texbinormalset
Read-only tuple of texture binormal arrays. Each value is a numpy.array of size Nx3 where N is the number of texture binormals in the primitive's source array.

texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array.

textangent_indexset
Read-only tuple of texture tangent index arrays. Each value is a numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual texture tangents, one can use the array to select into the textangentset array, e.g. `textangentset[0][textangent_indexset[0]]` would select the first set of texture tangents.

textangentset
Read-only tuple of texture tangent arrays. Each value is a numpy.array of size Nx3 where N is the number of texture tangents in the primitive's source array.

triangleset()
This performs a simple triangulation of the polylist using the fanning method.

Return type *collada.triangleset.TriangleSet*

vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.

vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`.

3.1.4 Shapes

collada.triangleset.Triangle	Single triangle representation.
collada.polylist.Polygon	Single polygon representation.
collada.lineset.Line	Single line representation.

collada.triangleset.Triangle

class collada.triangleset.**Triangle**(*indices*, *vertices*, *normal_indices*, *normals*, *texcoord_indices*, *texcoords*, *material*)

Bases: object

Single triangle representation.

__init__(*indices*, *vertices*, *normal_indices*, *normals*, *texcoord_indices*, *texcoords*, *material*)
A triangle should not be created manually.

Methods

__init__(indices, vertices, normal_indices, ...)	A triangle should not be created manually.

vertices = None
A (3, 3) float array for points in the triangle

texcoords = None
A tuple with (3, 2) float arrays with the texture coordinates for the points in the triangle

material = None
If coming from an unbound *collada.triangleset.TriangleSet*, contains a string with the material symbol. If coming from a bound *collada.triangleset.BoundTriangleSet*, contains the actual *collada.material.Effect* the triangle is bound to.

indices = None
A (3,) int array with vertex indexes of the 3 vertices in the vertex array

normal_indices = None
A (3,) int array with normal indexes of the 3 vertices in the normal array

texcoord_indices = None
A (3,2) int array with texture coordinate indexes of the 3 vertices in the texcoord array.

normals = None
A (3, 3) float array with the normals for points in the triangle. If the triangle didn't have normals, they will be computed.

collada.polylist.Polygon

class collada.polylist.**Polygon**(*indices*, *vertices*, *normal_indices*, *normals*, *texcoord_indices*, *texcoords*, *material*)

Bases: object

Single polygon representation. Represents a polygon of N points.

__init__(*indices*, *vertices*, *normal_indices*, *normals*, *texcoord_indices*, *texcoords*, *material*)
A Polygon should not be created manually.

Methods

__init__(indices, vertices, normal_indices, ...)	A Polygon should not be created manually.
triangles()	This triangulates the polygon using a simple fanning method.

vertices = None
A (N, 3) float array containing the points in the polygon.

normals = None
A (N, 3) float array with the normals for points in the polygon. Can be None.

texcoords = None
A tuple where entries are numpy float arrays of size (N, 2) containing the texture coordinates for the points in the polygon for each texture coordinate set. Can be length 0 if there are no texture coordinates.

material = None
If coming from an unbound *collada.polylist.Polylist*, contains a string with the material symbol. If coming from a bound *collada.polylist.BoundPolylist*, contains the actual *collada.material.Effect* the line is bound to.

indices = None

A (N,) int array containing the indices for the vertices of the N points in the polygon.

normal_indices = None
A (N,) int array containing the indices for the normals of the N points in the polygon

texcoord_indices = None
A (N,2) int array with texture coordinate indexes for the texcoords of the N points in the polygon

triangles()
This triangulates the polygon using a simple fanning method.

Return type generator of *collada.polylist.Polygon*

collada.lineset.Line

class collada.lineset.**Line**(*indices*, *vertices*, *normals*, *texcoords*, *material*)
Bases: object

Single line representation. Represents the line between two points (x0,y0,z0) and (x1,y1,z1). A Line is read-only.

__init__(*indices*, *vertices*, *normals*, *texcoords*, *material*)
A Line should not be created manually.

Methods

__init__(indices, vertices, normals, ...)	A Line should not be created manually.

vertices = None
A (2, 3) numpy float array containing the endpoints of the line

normals = None
A (2, 3) numpy float array with the normals for the endpoints of the line. Can be None.

texcoords = None
A tuple where entries are numpy float arrays of size (2, 2) containing the texture coordinates for the endpoints of the line for each texture coordinate set. Can be length 0 if there are no texture coordinates.

material = None
If coming from an unbound *collada.lineset.LineSet*, contains a string with the material symbol. If coming from a bound *collada.lineset.BoundLineSet*, contains the actual *collada.material.Effect* the line is bound to.

3.1.5 Controller

collada.controller.Controller	Base controller class holding data from <controller> tags.
collada.controller.Skin	Class containing data collada holds in the <skin> tag
collada.controller.Morph	Class containing data collada holds in the <morph> tag

collada.controller.Controller

class collada.controller.**Controller**
Bases: *collada.common.DaeObject*

Base controller class holding data from <controller> tags.

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

bind(matrix, materialnodebysymbol)	
load(collada, localscope, node)	
save()	Put all the data to the internal xml node (xmlnode) so it can be serialized.

save()
Put all the data to the internal xml node (xmlnode) so it can be serialized.

collada.controller.Skin

class collada.controller.**Skin**(*sourcebyid*, *bind_shape_matrix*, *joint_source*, *joint_matrix_source*, *weight_source*, *weight_joint_source*, *vcounts*, *vertex_weight_index*, *offsets*, *geometry*, *controller_node=None*, *skin_node=None*)
Bases: *collada.controller.Controller*

Class containing data collada holds in the <skin> tag

__init__(*sourcebyid*, *bind_shape_matrix*, *joint_source*, *joint_matrix_source*, *weight_source*, *weight_joint_source*, *vcounts*, *vertex_weight_index*, *offsets*, *geometry*, *controller_node=None*, *skin_node=None*)
Create a skin.

Parameters

sourceById A dict mapping id's to a collada source

bind_shape_matrix A numpy array of floats (pre-shape)

joint_source The string id for the joint source

joint_matrix_source The string id for the joint matrix source

weight_source The string id for the weight source

weight_joint_source The string id for the joint source of weights

vcounts A list with the number of influences on each vertex

vertex_weight_index An array with the indexes as they come from <v> array

offsets A list with the offsets in the weight index array for each source in (joint, weight)

geometry The source geometry this should be applied to (geometry.Geometry)

controller_node XML node of the <controller> tag which is the parent of this

skin_node XML node of the <skin> tag if this is from there

Methods

__init__(sourcebyid, bind_shape_matrix, ...)	Create a skin.
bind(matrix, materialnodebysymbol)	Create a bound morph from this one, transform and material mapping
load(collada, localscope, skinnode, ...)	
save()	Put all the data to the internal xml node (xmlnode) so it can be serialized.

bind(*matrix*, *materialnodebysymbol*)
Create a bound morph from this one, transform and material mapping

save()
Put all the data to the internal xml node (xmlnode) so it can be serialized.

collada.controller.Morph

class collada.controller.**Morph**(*source_geometry*, *target_list*, *xmlnode=None*)
Bases: *collada.controller.Controller*

Class containing data collada holds in the <morph> tag

__init__(*source_geometry*, *target_list*, *xmlnode=None*)
Create a morph instance

Parameters

source_geometry The source geometry (Geometry)

targets A list of tuples where each tuple (g,w) contains a Geometry (g) and a float weight value (w)

xmlnode When loaded, the xmlnode it comes from

Methods

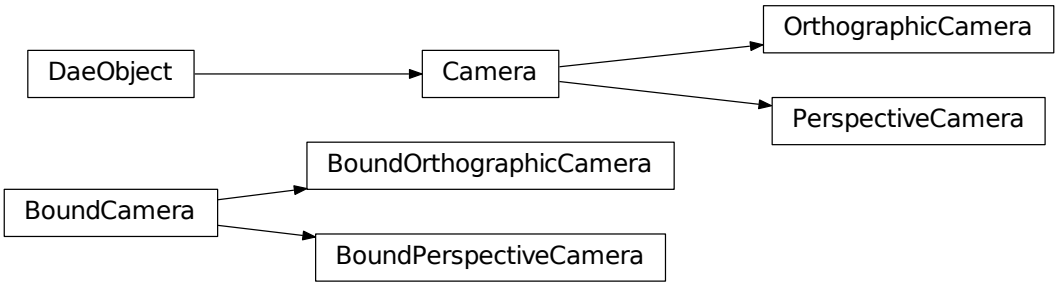
__init__(source_geometry, target_list[, xmlnode])	Create a morph instance
bind(matrix, materialnodebysymbol)	Create a bound morph from this one, transform and material mapping
load(collada, localscope, morphnode, ...)	
save()	

source_geometry = None
The source geometry (Geometry)

target_list = None
A list of tuples where each tuple (g,w) contains a Geometry (g) and a float weight value (w)

bind(*matrix*, *materialnodebysymbol*)
Create a bound morph from this one, transform and material mapping

3.1.6 Camera



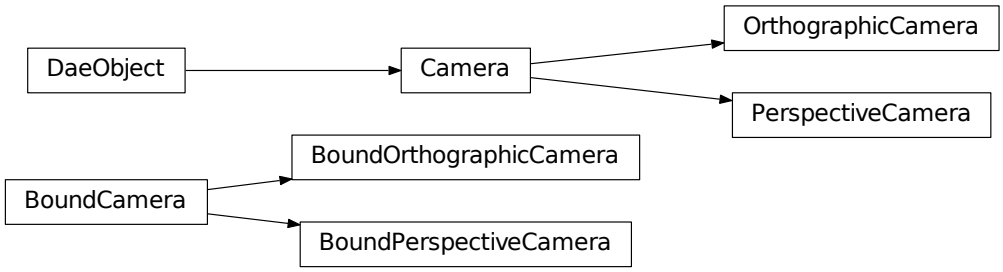
collada.camera.Camera	Base camera class holding data from <camera> tags.
collada.camera.PerspectiveCamera	Perspective camera as defined in COLLADA tag <perspective>.
collada.camera.OrthographicCamera	Orthographic camera as defined in COLLADA tag <orthographic>.

collada.camera.Camera

class collada.camera.**Camera**

Bases: *collada.common.DaeObject*

Base camera class holding data from <camera> tags.



__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

load(collada, localscope, node)	
save()	Put all the data to the internal xml node (xmlnode) so it can be serialized.

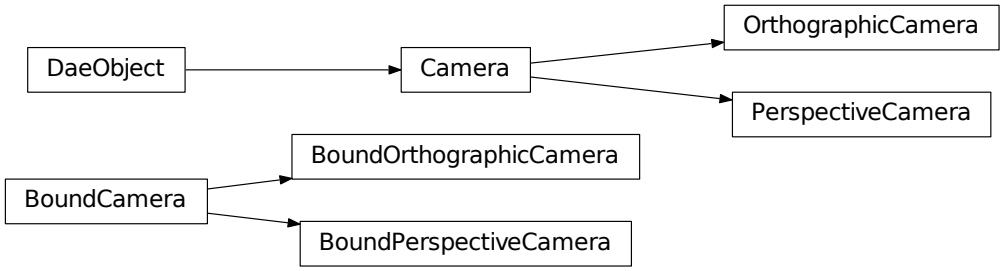
save()
Put all the data to the internal xml node (xmlnode) so it can be serialized.

collada.camera.PerspectiveCamera

class collada.camera.**PerspectiveCamera**(*id*, *znear*, *zfar*, *xfov=None*, *yfov=None*, *aspect_ratio=None*, *xmlnode=None*)

Bases: *collada.camera.Camera*

Perspective camera as defined in COLLADA tag <perspective>.



__init__(*id*, *znear*, *zfar*, *xfov=None*, *yfov=None*, *aspect_ratio=None*, *xmlnode=None*)
Create a new perspective camera.

Note: `aspect_ratio = tan(0.5*xfov) / tan(0.5*yfov)`

You can specify one of:

- *xfov* alone
- *yfov* alone
- *xfov* and *yfov*
- *xfov* and *aspect_ratio*
- *yfov* and *aspect_ratio*

Any other combination will raise *collada.common.DaeMalformedError*

Parameters

- **id** (*str*) – Identifier for the camera
- **znear** (*float*) – Distance to the near clipping plane
- **zfar** (*float*) – Distance to the far clipping plane
- **xfov** (*float*) – Horizontal field of view, in degrees

- **yfov** (*float*) – Vertical field of view, in degrees
- **aspect_ratio** (*float*) – Aspect ratio of the field of view
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, znear, zfar[, xfov, yfov, ...])	Create a new perspective camera.
bind(matrix)	Create a bound camera of itself based on a transform matrix.
load(collada, localscope, node)	
save()	Saves the perspective camera's properties back to xmlnode

id = None
Identifier for the camera

xfov = None
Horizontal field of view, in degrees

yfov = None
Vertical field of view, in degrees

aspect_ratio = None
Aspect ratio of the field of view

znear = None
Distance to the near clipping plane

zfar = None
Distance to the far clipping plane

xmlnode = None
ElementTree representation of the data.

save()
Saves the perspective camera's properties back to xmlnode

bind(*matrix*)
Create a bound camera of itself based on a transform matrix.

Parameters **matrix** (*numpy.array*) – A numpy transformation matrix of size 4x4

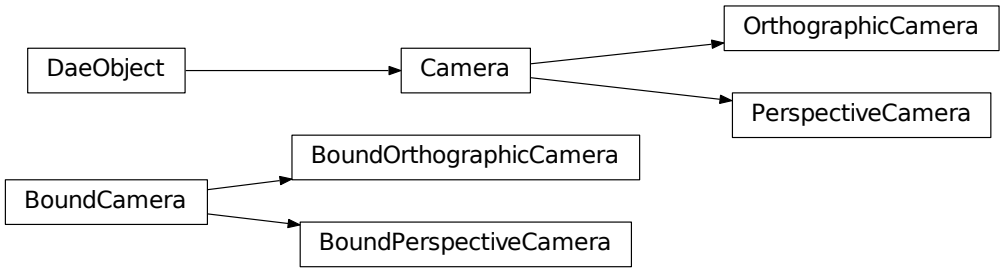
Return type *collada.camera.BoundPerspectiveCamera*

collada.camera.OrthographicCamera

class collada.camera.**OrthographicCamera**(*id*, *znear*, *zfar*, *xmag=None*, *ymag=None*, *aspect_ratio=None*, *xmlnode=None*)

Bases: *collada.camera.Camera*

Orthographic camera as defined in COLLADA tag <orthographic>.



__init__(*id*, *znear*, *zfar*, *xmag=None*, *ymag=None*, *aspect_ratio=None*, *xmlnode=None*)

Create a new orthographic camera.

Note: `aspect_ratio = xmag / ymag`

You can specify one of:

- *xmag* alone
- *ymag* alone
- *xmag* and *ymag*
- *xmag* and *aspect_ratio*
- *ymag* and *aspect_ratio*

Any other combination will raise *collada.common.DaeMalformedError*

Parameters

- **id** (*str*) – Identifier for the camera
- **znear** (*float*) – Distance to the near clipping plane
- **zfar** (*float*) – Distance to the far clipping plane
- **xmag** (*float*) – Horizontal magnification of the view
- **ymag** (*float*) – Vertical magnification of the view
- **aspect_ratio** (*float*) – Aspect ratio of the field of view
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, znear, zfar[, xmag, ymag, ...])	Create a new orthographic camera.
bind(matrix)	Create a bound camera of itself based on a transform matrix.
load(collada, localscope, node)	
save()	Saves the orthographic camera's properties back to xmlnode

id = None
Identifier for the camera

xmag = None
Horizontal magnification of the view

ymag = None
Vertical magnification of the view

aspect_ratio = None
Aspect ratio of the field of view

znear = None
Distance to the near clipping plane

zfar = None
Distance to the far clipping plane

xmlnode = None
ElementTree representation of the data.

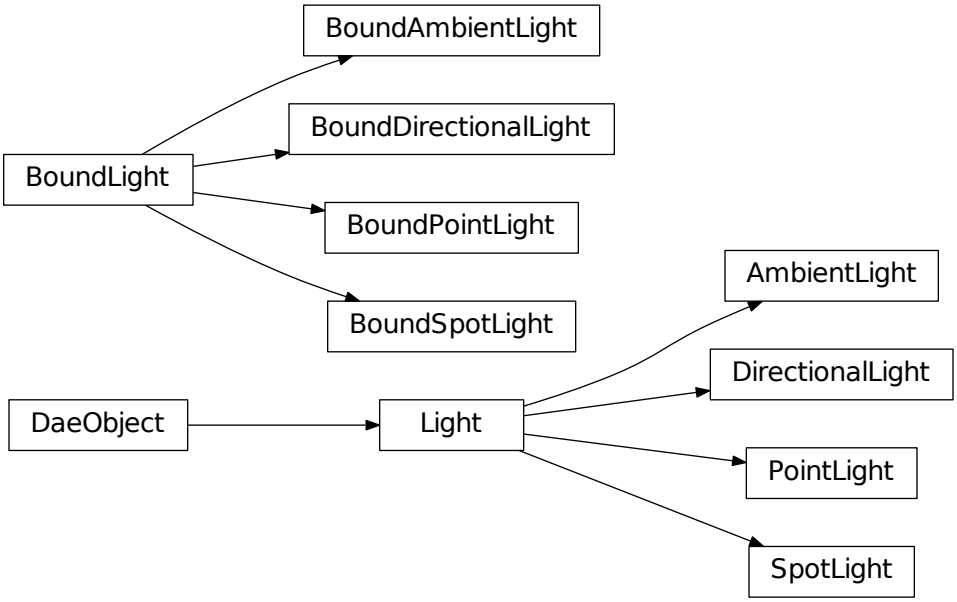
save()
Saves the orthographic camera's properties back to xmlnode

bind(*matrix*)
Create a bound camera of itself based on a transform matrix.

Parameters **matrix** (*numpy.array*) – A numpy transformation matrix of size 4x4

Return type *collada.camera.BoundOrthographicCamera*

3.1.7 Light



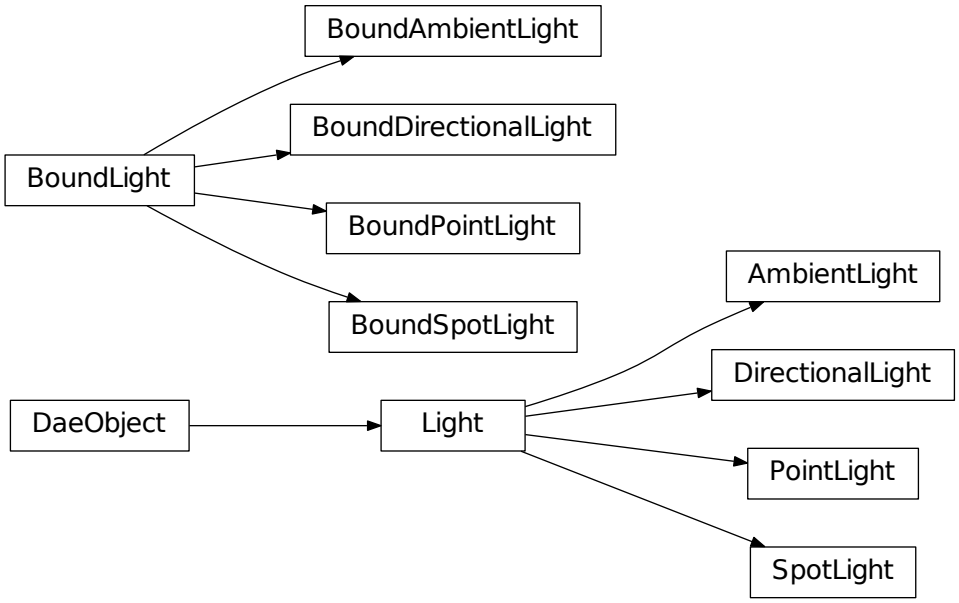
collada.light.Light	Base light class holding data from <light> tags.
collada.light.DirectionalLight	Directional light as defined in COLLADA tag <directional> tag.
collada.light.AmbientLight	Ambient light as defined in COLLADA tag <ambient>.
collada.light.PointLight	Point light as defined in COLLADA tag <point>.
collada.light.SpotLight	Spot light as defined in COLLADA tag <spot>.

collada.light.Light

class collada.light.**Light**

Bases: *collada.common.DaeObject*

Base light class holding data from <light> tags.



__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

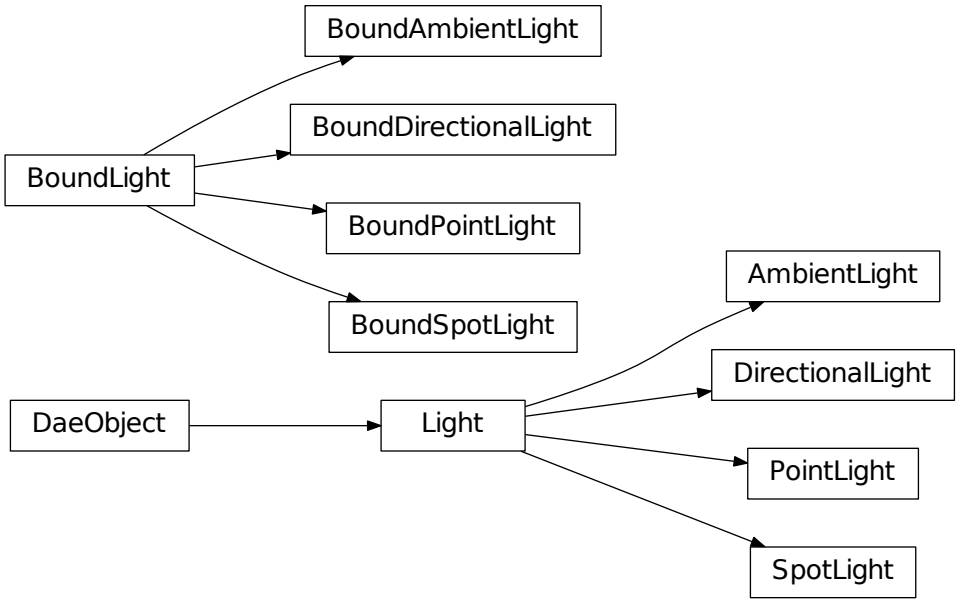
load(collada, localscope, node)	
save()	Put all the data to the internal xml node (xmlnode) so it can be serialized.

save()
Put all the data to the internal xml node (xmlnode) so it can be serialized.

collada.light.DirectionalLight

class collada.light.**DirectionalLight**(*id*, *color*, *xmlnode=None*)
Bases: *collada.light.Light*

Directional light as defined in COLLADA tag <directional> tag.



__init__(*id*, *color*, *xmlnode=None*)
Create a new directional light.

Parameters

- **id** (*str*) – A unique string identifier for the light
- **color** (*tuple*) – Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, color[, xmlnode])	Create a new directional light.
bind(matrix)	Binds this light to a transform matrix.
load(collada, localscope, node)	
save()	Saves the light's properties back to *xmlnode*

id = None
The unique string identifier for the light

color = None
Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light

xmlnode = None
ElementTree representation of the light.

save()
Saves the light's properties back to *xmlnode*

bind(*matrix*)
Binds this light to a transform matrix.

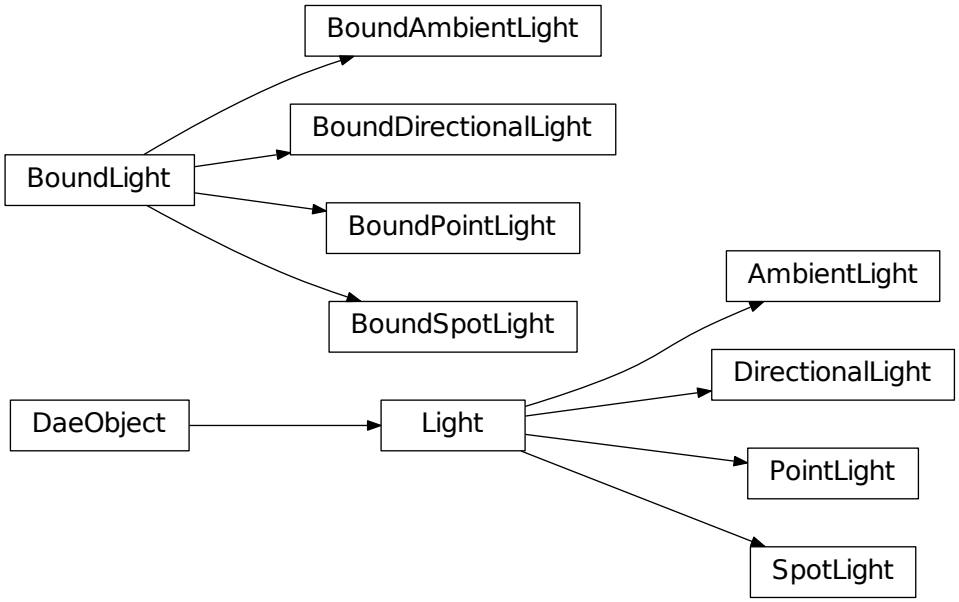
Parameters **matrix** (*numpy.array*) – A 4x4 numpy float matrix

Return type *collada.light.BoundDirectionalLight*

collada.light.AmbientLight

class collada.light.**AmbientLight**(*id*, *color*, *xmlnode=None*)
Bases: *collada.light.Light*

Ambient light as defined in COLLADA tag <ambient>.



__init__(*id*, *color*, *xmlnode=None*)
Create a new ambient light.

Parameters

- **id** (*str*) – A unique string identifier for the light
- **color** (*tuple*) – Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, color[, xmlnode])	Create a new ambient light.
bind(matrix)	Binds this light to a transform matrix.
load(collada, localscope, node)	
save()	Saves the light's properties back to *xmlnode*

id = None
The unique string identifier for the light

color = None
Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light

xmlnode = None
ElementTree representation of the light.

save()
Saves the light's properties back to *xmlnode*

bind(*matrix*)
Binds this light to a transform matrix.

Parameters **matrix** (*numpy.array*) – A 4x4 numpy float matrix

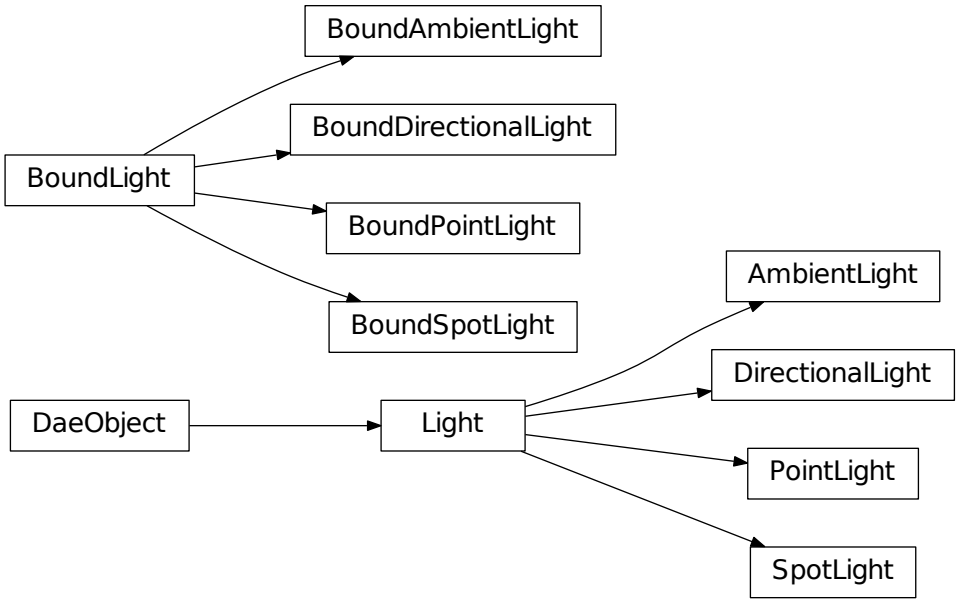
Return type *collada.light.BoundAmbientLight*

collada.light.PointLight

class collada.light.**PointLight**(*id*, *color*, *constant_att=None*, *linear_att=None*, *quad_att=None*, *zfar=None*, *xmlnode=None*)

Bases: *collada.light.Light*

Point light as defined in COLLADA tag <point>.



__init__(*id*, *color*, *constant_att=None*, *linear_att=None*, *quad_att=None*, *zfar=None*, *xmlnode=None*)

Create a new sun light.

Parameters

- **id** (*str*) – A unique string identifier for the light
- **color** (*tuple*) – Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light
- **constant_att** (*float*) – Constant attenuation factor
- **linear_att** (*float*) – Linear attenuation factor
- **quad_att** (*float*) – Quadratic attenuation factor
- **zfar** (*float*) – Distance to the far clipping plane
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, color[, constant_att, ...])	Create a new sun light.
bind(matrix)	Binds this light to a transform matrix.
load(collada, localscope, node)	
save()	Saves the light's properties back to *xmlnode*

id = None

The unique string identifier for the light

color = None
Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light

constant_att = None
Constant attenuation factor.

linear_att = None
Linear attenuation factor.

quad_att = None
Quadratic attenuation factor.

zfar = None
Distance to the far clipping plane

xmlnode = None
ElementTree representation of the light.

save()
Saves the light's properties back to *xmlnode*

bind(*matrix*)
Binds this light to a transform matrix.

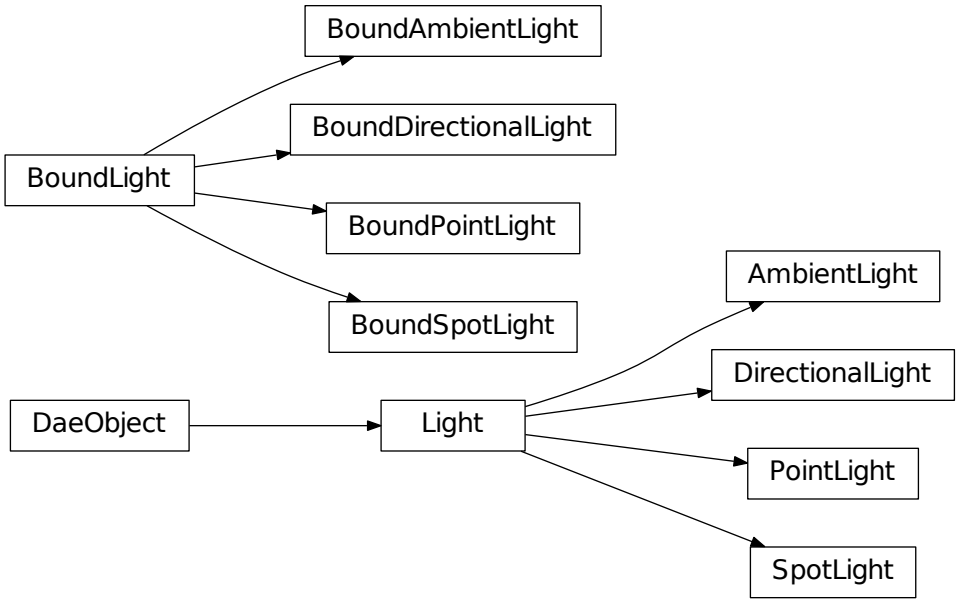
Parameters **matrix** (*numpy.array*) – A 4x4 numpy float matrix

Return type *collada.light.BoundPointLight*

collada.light.SpotLight

class collada.light.**SpotLight**(*id*, *color*, *constant_att=None*, *linear_att=None*, *quad_att=None*, *falloff_ang=None*, *falloff_exp=None*, *xmlnode=None*)
Bases: *collada.light.Light*

Spot light as defined in COLLADA tag <spot>.



__init__(*id*, *color*, *constant_att=None*, *linear_att=None*, *quad_att=None*, *falloff_ang=None*, *falloff_exp=None*, *xmlnode=None*)

Create a new spot light.

Parameters

- **id** (*str*) – A unique string identifier for the light
- **color** (*tuple*) – Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light
- **constant_att** (*float*) – Constant attenuation factor
- **linear_att** (*float*) – Linear attenuation factor
- **quad_att** (*float*) – Quadratic attenuation factor
- **falloff_ang** (*float*) – Falloff angle
- **falloff_exp** (*float*) – Falloff exponent
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, color[, constant_att, ...])	Create a new spot light.
bind(matrix)	Binds this light to a transform matrix.
load(collada, localscope, node)	
save()	Saves the light's properties back to *xmlnode*

id = None
The unique string identifier for the light

color = None
Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light

constant_att = None
Constant attenuation factor.

linear_att = None
Linear attenuation factor.

quad_att = None
Quadratic attenuation factor.

falloff_ang = None
Falloff angle

falloff_exp = None
Falloff exponent

xmlnode = None
ElementTree representation of the light.

save()
Saves the light's properties back to *xmlnode*

bind(*matrix*)
Binds this light to a transform matrix.

Parameters **matrix** (*numpy.array*) – A 4x4 numpy float matrix

Return type *collada.light.BoundSpotLight*

3.1.8 Material

collada.material.Material	Class containing data coming from a <material> tag.
collada.material.Effect	Class containing data coming from an <effect> tag.
collada.material.CImage	Class containing data coming from a <image> tag.
collada.material.Surface	Class containing data coming from a <surface> tag.
collada.material.Sampler2D	Class containing data coming from <sampler2D> tag in material.
collada.material.Map	Class containing data coming from <texture> tag inside material.
collada.material.OPAQUE_MODE	The opaque mode of an effect.

collada.material.Material

class collada.material.**Material**(*id*, *name*, *effect*, *xmlnode=None*)
Bases: *collada.common.DaeObject*

Class containing data coming from a <material> tag.

Right now, this just stores a reference to the effect which is instantiated in the material. The effect instance can have parameters, but this is rarely used in the wild, so it is not yet implemented.

__init__(*id*, *name*, *effect*, *xmlnode=None*)
Creates a material.

Parameters

- **id** (*str*) – A unique string identifier for the material
- **name** (*str*) – A name for the material
- **effect** (collada.material.Effect) – The effect instantiated in this material
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, name, effect[, xmlnode])	Creates a material.
load(collada, localscope, node)	
save()	Saves the material data back to *xmlnode*

id = None
The unique string identifier for the material

name = None
The name for the material

effect = None
The *collada.material.Effect* instantiated in this material

xmlnode = None
ElementTree representation of the surface.

save()
Saves the material data back to *xmlnode*

collada.material.Effect

class collada.material.**Effect**(*id*, *params*, *shadingtype*, *bumpmap=None*, *double_sided=False*, *emission=(0.0, 0.0, 0.0, 1.0)*, *ambient=(0.0, 0.0, 0.0, 1.0)*, *diffuse=(0.0, 0.0, 0.0, 1.0)*, *specular=(0.0, 0.0, 0.0, 1.0)*, *shininess=0.0*, *reflective=(0.0, 0.0, 0.0, 1.0)*, *reflectivity=0.0*, *transparent=(0.0, 0.0, 0.0, 1.0)*, *transparency=None*, *index_of_refraction=None*, *opaque_mode=None*, *xmlnode=None*)

Bases: *collada.common.DaeObject*

Class containing data coming from an <effect> tag.

__init__(*id*, *params*, *shadingtype*, *bumpmap=None*, *double_sided=False*, *emission=(0.0, 0.0, 0.0, 1.0)*, *ambient=(0.0, 0.0, 0.0, 1.0)*, *diffuse=(0.0, 0.0, 0.0, 1.0)*, *specular=(0.0, 0.0, 0.0, 1.0)*, *shininess=0.0*, *reflective=(0.0, 0.0, 0.0, 1.0)*, *reflectivity=0.0*, *transparent=(0.0, 0.0, 0.0, 1.0)*, *transparency=None*, *index_of_refraction=None*, *opaque_mode=None*, *xmlnode=None*)

Create an effect instance out of properties.

Parameters

- **id** (*str*) – A string identifier for the effect
- **params** (*list*) – A list containing elements of type *collada.material.Sampler2D* and *collada.material.Surface*

- **shadingtype** (*str*) – The type of shader to be used for this effect. Right now, we only supper the shaders listed in *shaders*
- **bumpmap** (collada.material.Map) – The bump map for this effect, or None if there isn't one
- **double_sided** (*bool*) – Whether or not the material should be rendered double sided
- **emission** – Either an RGBA-format tuple of four floats or an instance of *collada.material.Map*
- **ambient** – Either an RGBA-format tuple of four floats or an instance of *collada.material.Map*
- **diffuse** – Either an RGBA-format tuple of four floats or an instance of *collada.material.Map*
- **specular** – Either an RGBA-format tuple of four floats or an instance of *collada.material.Map*
- **shininess** – Either a single float or an instance of *collada.material.Map*
- **reflective** – Either an RGBA-format tuple of four floats or an instance of *collada.material.Map*
- **reflectivity** – Either a single float or an instance of *collada.material.Map*
- **transparent** (*tuple*) – Either an RGBA-format tuple of four floats or an instance of *collada.material.Map*
- **transparency** – Either a single float or an instance of *collada.material.Map*
- **index_of_refraction** (*float*) – A single float indicating the index of refraction for perfectly refracted light
- **opaque_mode** (collada.material.OPAQUE_MODE) – The opaque mode for the effect. If not specified, defaults to A_ONE.
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, params, shadingtype[, bumpmap, ...])	Create an effect instance out of properties.
load(collada, localscope, node)	
save()	Saves the effect back to *xmlnode*

supported = ['emission', 'ambient', 'diffuse', 'specular', 'shininess', 'reflective', 'reflectivity', 'transparent', 'transparen
Supported material properties list.

shaders = ['phong', 'lambert', 'blinn', 'constant']
Supported shader list.

id = None
The string identifier for the effect

params = None
A list containing elements of type *collada.material.Sampler2D* and *collada.material.Surface*

shadingtype = None
String with the type of the shading.

bumpmap = None
Either the bump map of the effect of type *collada.material.Map* or None if there is none.

double_sided = None
A boolean indicating whether or not the material should be rendered double sided

emission = None
Either an RGB-format tuple of three floats or an instance of *collada.material.Map*

ambient = None
Either an RGB-format tuple of three floats or an instance of *collada.material.Map*

diffuse = None
Either an RGB-format tuple of three floats or an instance of *collada.material.Map*

specular = None
Either an RGB-format tuple of three floats or an instance of *collada.material.Map*

shininess = None
Either a single float or an instance of *collada.material.Map*

reflective = None
Either an RGB-format tuple of three floats or an instance of *collada.material.Map*

reflectivity = None
Either a single float or an instance of *collada.material.Map*

transparent = None
Either an RGB-format tuple of three floats or an instance of *collada.material.Map*

index_of_refraction = None
A single float indicating the index of refraction for perfectly refracted light

opaque_mode = None
The opaque mode for the effect. An instance of *collada.material.OPAQUE_MODE*.

transparency = None
Either a single float or an instance of *collada.material.Map*

xmlnode = None
ElementTree representation of the effect

save()
Saves the effect back to *xmlnode*

almostEqual(*other*)
Checks if this effect is almost equal (within float precision) to the given effect.

Parameters **other** (collada.material.Effect) – Effect to compare to

Return type bool

collada.material.CImage

class collada.material.**CImage**(*id*, *path*, *collada=None*, *xmlnode=None*)
Bases: *collada.common.DaeObject*

Class containing data coming from a <image> tag.

Basically is just the path to the file, but we give an extended functionality if PIL is available. You can in that case get the image object or numpy arrays in both int and float format. We named it CImage to avoid confusion with PIL's Image class.

__init__(*id*, *path*, *collada=None*, *xmlnode=None*)
Create an image object.

Parameters

- **id** (*str*) – A unique string identifier for the image
- **path** (*str*) – Path relative to the collada document where the image is located
- **collada** (collada.Collada) – The collada object this image belongs to
- **xmlnode** – If loaded from xml, the node this data comes from

Methods

__init__(id, path[, collada, xmlnode])	Create an image object.
load(collada, localspace, node)	
save()	Saves the image back to *xmlnode*.

Attributes

data	Raw binary image file data if the file is readable.
floatarray	Numpy float array (height, width, nchannels) with the image data normalized to 1.0.
pilimage	PIL Image object if PIL is available and the file is readable.
uintarray	Numpy array (height, width, nchannels) in integer format.

id = None
The unique string identifier for the image

path = None
Path relative to the collada document where the image is located

xmlnode = None
ElementTree representation of the image.

data
Raw binary image file data if the file is readable. If *aux_file_loader* was passed to *collada.Collada.__init__()*, this function will be called to retrieve the data. Otherwise, if the file came from the local disk, the path will be interpreted from the local file system. If the file was a zip archive, the archive will be searched.

pilimage
PIL Image object if PIL is available and the file is readable.

uintarray
Numpy array (height, width, nchannels) in integer format.

floatarray
Numpy float array (height, width, nchannels) with the image data normalized to 1.0.

save()
Saves the image back to *xmlnode*. Only the *id* attribute is saved. The image itself will have to be saved to its original source to make modifications.

collada.material.Surface

class collada.material.**Surface**(*id*, *img*, *format=None*, *xmlnode=None*)
Bases: *collada.common.DaeObject*

Class containing data coming from a <surface> tag.

Collada materials use this to access to the <image> tag. The only extra information we store right now is the image format. In theory, this enables many more features according to the collada spec, but no one seems to actually use them in the wild, so for now, it's unimplemented.

__init__(*id*, *img*, *format=None*, *xmlnode=None*)
Creates a surface.

Parameters

- **id** (*str*) – A string identifier for the surface within the local scope of the material
- **img** (collada.material.CImage) – The image object
- **format** (*str*) – The format of the image
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, img[, format, xmlnode])	Creates a surface.
load(collada, localscope, node)	
save()	Saves the surface data back to *xmlnode*

id = None
The string identifier for the surface within the local scope of the material

image = None
collada.material.CImage object from the image library.

format = None
Format string.

xmlnode = None
ElementTree representation of the surface.

save()
Saves the surface data back to *xmlnode*

collada.material.Sampler2D

class collada.material.**Sampler2D**(*id*, *surface*, *minfilter=None*, *magfilter=None*, *xmlnode=None*)
Bases: *collada.common.DaeObject*

Class containing data coming from <sampler2D> tag in material.

Collada uses the <sampler2D> tag to map to a <surface>. The only information we store about the sampler right now is minfilter and magfilter. Theoretically, the collada spec has many more parameters here, but no one seems to be using them in the wild, so they are currently unimplemented.

__init__(*id*, *surface*, *minfilter=None*, *magfilter=None*, *xmlnode=None*)
Create a Sampler2D object.

Parameters

- **id** (*str*) – A string identifier for the sampler within the local scope of the material
- **surface** (collada.material.Surface) – Surface instance that this object samples from
- **minfilter** (*str*) – Minification filter string id, see collada spec for details
- **magfilter** (*str*) – Maximization filter string id, see collada spec for details
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(id, surface[, minfilter, ...])	Create a Sampler2D object.
load(collada, localscope, node)	
save()	Saves the sampler data back to *xmlnode*

id = None
The string identifier for the sampler within the local scope of the material

surface = None
Surface instance that this object samples from

minfilter = None
Minification filter string id, see collada spec for details

magfilter = None
Maximization filter string id, see collada spec for details

xmlnode = None
ElementTree representation of the sampler.

save()
Saves the sampler data back to *xmlnode*

collada.material.Map

class collada.material.**Map**(*sampler*, *texcoord*, *xmlnode=None*)
Bases: *collada.common.DaeObject*

Class containing data coming from <texture> tag inside material.

When a material defines its properties like *diffuse*, it can give you a color or a texture. In the latter, the texture is mapped with a sampler and a texture coordinate channel. If a material defined a texture for one of its properties, you'll find an object of this class in the corresponding attribute.

__init__(*sampler*, *texcoord*, *xmlnode=None*)
Create a map instance to a sampler using a texcoord channel.

Parameters

- **sampler** (collada.material.Sampler2D) – A sampler object to map
- **texcoord** (*str*) – Texture coordinate channel symbol to use
- **xmlnode** – If loaded from xml, the xml node

Methods

__init__(sampler, texcoord[, xmlnode])	Create a map instance to a sampler using a texcoord channel.
load(collada, localscope, node)	
save()	Saves the map back to *xmlnode*

sampler = None
collada.material.Sampler2D object to map

texcoord = None
Texture coordinate channel symbol to use

xmlnode = None
ElementTree representation of the map

save()
Saves the map back to *xmlnode*

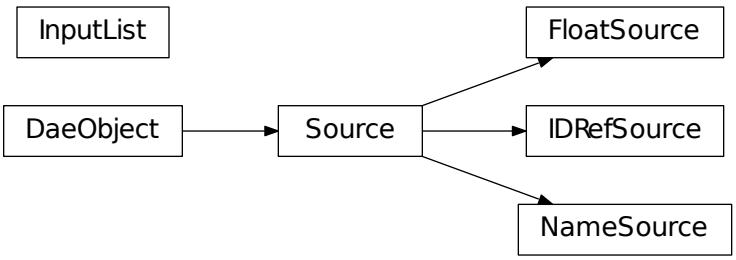
collada.material.OPAQUE_MODE

class collada.material.**OPAQUE_MODE**
The opaque mode of an effect.

A_ONE = 'A_ONE'
Takes the transparency information from the color's alpha channel, where the value 1.0 is opaque (default).

RGB_ZERO = 'RGB_ZERO'
Takes the transparency information from the color's red, green, and blue channels, where the value 0.0 is opaque, with each channel modulated independently.

3.1.9 Source



collada.source.InputList	Used for defining input sources to a geometry.
collada.source.Source	Abstract class for loading source arrays

Continued on next page

Table 3.45 – continued from previous page

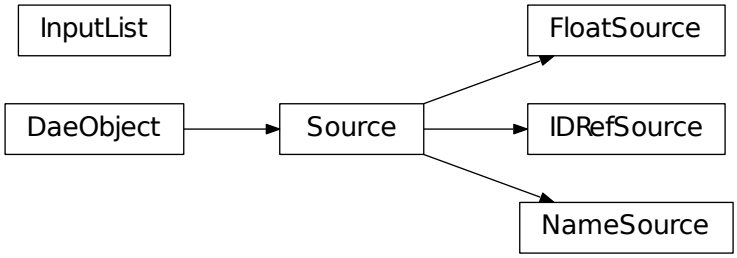
collada.source.FloatSource	Contains a source array of floats, as defined in the collada <float_array> inside a <source>.
collada.source.NameSource	Contains a source array of strings, as defined in the collada <Name_array> inside a <source>.
collada.source.IDRefSource	Contains a source array of ID references, as defined in the collada <IDREF_array> inside a <source>.

collada.source.InputList

class collada.source.**InputList**

Bases: object

Used for defining input sources to a geometry.



__init__()

Create an input list

Methods

__init__()	Create an input list
addInput(offset, semantic, src[, set])	Add an input source to this input list.
getList()	Returns a list of tuples of the source in the form (offset, semantic, source, set)

addInput(*offset*, *semantic*, *src*, *set=None*)

Add an input source to this input list.

Parameters

- **offset** (*int*) – Offset for this source within the geometry's indices
- **semantic** (*str*) –

  The semantic for the input source. Currently supported options are:

  - VERTEX
  - NORMAL

- TEXCOORD
- TEXBINORMAL
- TEXTANGENT
- COLOR
- TANGENT
- BINORMAL

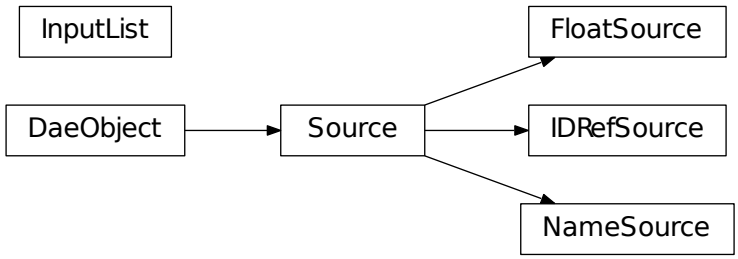
- **src** (*str*) – A string identifier of the form *#srcid* where *srcid* is a source within the geometry's *sourceById* array.
- **set** (*str*) – Indicates a set number for the source. This is used, for example, when there are multiple texture coordinate sets.

getList()
Returns a list of tuples of the source in the form (offset, semantic, source, set)

collada.source.Source

class collada.source.**Source**
Bases: *collada.common.DaeObject*

Abstract class for loading source arrays



__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

load(collada, localscope, node)	
save()	Put all the data to the internal xml node (xmlnode) so it can be serialized.

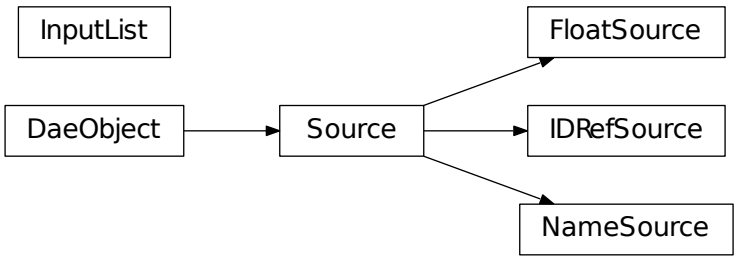
save()
Put all the data to the internal xml node (xmlnode) so it can be serialized.

collada.source.FloatSource

class collada.source.**FloatSource**(*id*, *data*, *components*, *xmlnode=None*)
Bases: *collada.source.Source*

Contains a source array of floats, as defined in the collada <float_array> inside a <source>.

If `f` is an instance of *collada.source.FloatSource*, then `len(f)` is the length of the shaped source. `len(f)*len(f.components)` would give you the number of values in the source. `f[i]` is the i^{th} item in the source array.



__init__(*id*, *data*, *components*, *xmlnode=None*)
Create a float source instance.

Parameters

- **id** (*str*) – A unique string identifier for the source
- **data** (*numpy.array*) – Numpy array (unshaped) with the source values
- **components** (*tuple*) – Tuple of strings describing the semantic of the data, e.g. `('X','Y','Z')` would cause *data* to be reshaped as `(-1, 3)`
- **xmlnode** – When loaded, the xmlnode it comes from.

Methods

__init__(id, data, components[, xmlnode])	Create a float source instance.
load(collada, localscope, node)	
save()	Saves the source back to *xmlnode*

id = None
The unique string identifier for the source

data = None
Numpy array with the source values. This will be shaped as `(-1,N)` where `N = len(self.components)`

components = None
Tuple of strings describing the semantic of the data, e.g. `('X','Y','Z')`

xmlnode = None

ElementTree representation of the source.

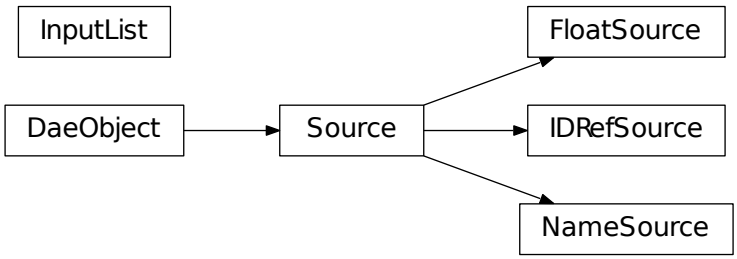
save()
Saves the source back to *xmlnode*

collada.source.NameSource

class collada.source.**NameSource**(*id*, *data*, *components*, *xmlnode=None*)
Bases: *collada.source.Source*

Contains a source array of strings, as defined in the collada <Name_array> inside a <source>.

If `n` is an instance of *collada.source.NameSource*, then `len(n)` is the length of the shaped source. `len(n)*len(n.components)` would give you the number of values in the source. `n[i]` is the i^{th} item in the source array.



__init__(*id*, *data*, *components*, *xmlnode=None*)
Create a name source instance.

Parameters

- **id** (*str*) – A unique string identifier for the source
- **data** (*numpy.array*) – Numpy array (unshaped) with the source values
- **components** (*tuple*) – Tuple of strings describing the semantic of the data, e.g. `('JOINT')` would cause *data* to be reshaped as `(-1, 1)`
- **xmlnode** – When loaded, the xmlnode it comes from.

Methods

__init__(id, data, components[, xmlnode])	Create a name source instance.
load(collada, localscope, node)	
save()	Saves the source back to *xmlnode*

id = None
The unique string identifier for the source

data = None
Numpy array with the source values. This will be shaped as `(-1,N)` where `N = len(self.`

components)

components = None
Tuple of strings describing the semantic of the data, e.g. `('JOINT')`

xmlnode = None
ElementTree representation of the source.

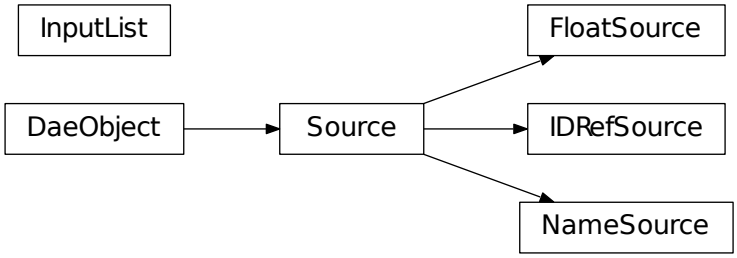
save()
Saves the source back to *xmlnode*

collada.source.IDRefSource

class collada.source.**IDRefSource**(*id*, *data*, *components*, *xmlnode=None*)
Bases: *collada.source.Source*

Contains a source array of ID references, as defined in the collada <IDREF_array> inside a <source>.

If `r` is an instance of *collada.source.IDRefSource*, then `len(r)` is the length of the shaped source. `len(r)*len(r.components)` would give you the number of values in the source. `r[i]` is the i^{th} item in the source array.



__init__(*id*, *data*, *components*, *xmlnode=None*)
Create an id ref source instance.

Parameters

- **id** (*str*) – A unique string identifier for the source
- **data** (*numpy.array*) – Numpy array (unshaped) with the source values
- **components** (*tuple*) – Tuple of strings describing the semantic of the data, e.g. `('MORPH_TARGET')` would cause *data* to be reshaped as `(-1, 1)`
- **xmlnode** – When loaded, the xmlnode it comes from.

Methods

__init__(id, data, components[, xmlnode])	Create an id ref source instance.
load(collada, localscope, node)	
save()	Saves the source back to *xmlnode*

id = None
The unique string identifier for the source

data = None
Numpy array with the source values. This will be shaped as `(-1,N)` where `N = len(self.components)`

components = None
Tuple of strings describing the semantic of the data, e.g. `('MORPH_TARGET')`

xmlnode = None
ElementTree representation of the source.

save()
Saves the source back to *xmlnode*

3.1.10 Scene

collada.scene.Scene	The root object for a scene, as defined in a collada <scene> tag
collada.scene.SceneNode	Abstract base class for all nodes within a scene.
collada.scene.Node	Represents a node object, which is a point on the scene graph, as defined in the collada <node> tag.
collada.scene.NodeNode	Represents a node being instantiated in a scene, as defined in the collada <instande_node> tag.
collada.scene.GeometryNode	Represents a geometry instance in a scene, as defined in the collada <instance_geometry> tag.
collada.scene.ControllerNode	Represents a controller instance in a scene, as defined in the collada <instance_controller> tag.
collada.scene.MaterialNode	Represents a material being instantiated in a scene, as defined in the collada <instance_material> tag.
collada.scene.LightNode	Represents a light being instantiated in a scene, as defined in the collada <instance_light> tag.
collada.scene.CameraNode	Represents a camera being instantiated in a scene, as defined in the collada <instance_camera> tag.
collada.scene.ExtraNode	Represents extra information in a scene, as defined in a collada <extra> tag.
collada.scene.Transform	Base class for all transformation types
collada.scene.TranslateTransform	Contains a translation transformation as defined in the collada <translate> tag.
collada.scene.RotateTransform	Contains a rotation transformation as defined in the collada <rotate> tag.
collada.scene.ScaleTransform	Contains a scale transformation as defined in the collada <scale> tag.
collada.scene.MatrixTransform	Contains a matrix transformation as defined in the collada <matrix> tag.
collada.scene.LookAtTransform	Contains a transformation for aiming a camera as defined in the collada <lookat> tag.

collada.scene.Scene

class collada.scene.**Scene**(*id*, *nodes*, *xmlnode=None*, *collada=None*)
Bases: *collada.common.DaeObject*

The root object for a scene, as defined in a collada <scene> tag

__init__(*id*, *nodes*, *xmlnode=None*, *collada=None*)
Create a scene

Parameters

- **id** (*str*) – A unique string identifier for the scene
- **nodes** (*list*) – A list of type *collada.scene.Node* representing the nodes in the scene
- **xmlnode** – When loaded, the xmlnode it comes from
- **collada** – The collada instance this is part of

Methods

__init__(id, nodes[, xmlnode, collada])	Create a scene
load(collada, node)	
objects(tipo)	Iterate through all objects in the scene that match *tipo*.
save()	Saves the scene back to *xmlnode*

id = None
The unique string identifier for the scene

nodes = None
A list of type *collada.scene.Node* representing the nodes in the scene

collada = None
The collada instance this is part of

xmlnode = None
ElementTree representation of the scene node.

objects(*tipo*)
Iterate through all objects in the scene that match *tipo*. The objects will be bound and transformed via the scene transformations.

Parameters **tipo** (*str*) – A string for the desired object type. This can be one of 'geometry', 'camera', 'light', or 'controller'.

Return type generator that yields the type specified

save()
Saves the scene back to *xmlnode*

collada.scene.SceneNode

class collada.scene.**SceneNode**
Bases: *collada.common.DaeObject*

Abstract base class for all nodes within a scene.

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

load(collada, localscope, node)	Load and return a class instance from an XML node.
objects(tipo[, matrix])	Iterate through all objects under this node that match *tipo*.
save()	Put all the data to the internal xml node (xmlnode) so it can be serialized.

objects (*tipo*, *matrix=None*)
Iterate through all objects under this node that match *tipo*. The objects will be bound and transformed via the scene transformations.

Parameters

- **tipo** (*str*) – A string for the desired object type. This can be one of 'geometry', 'camera', 'light', or 'controller'.
- **matrix** (*numpy.matrix*) – An optional transformation matrix

Return type generator that yields the type specified

load (*collada*, *localscope*, *node*)
Load and return a class instance from an XML node.

Inspect the data inside node, which must match this class tag and create an instance out of it.

Parameters

- **collada** (*collada.Collada*) – The collada file object where this object lives
- **localscope** (*dict*) – If there is a local scope where we should look for local ids (sid) this is the dictionary. Otherwise empty dict ({})
- **node** – An Element from python's ElementTree API

save ()
Put all the data to the internal xml node (xmlnode) so it can be serialized.

collada.scene.Node

class collada.scene.**Node** (*id*, *children=None*, *transforms=None*, *xmlnode=None*)
Bases: *collada.scene.SceneNode*

Represents a node object, which is a point on the scene graph, as defined in the collada <node> tag.

Contains the list of transformations effecting the node as well as any children.

__init__ (*id*, *children=None*, *transforms=None*, *xmlnode=None*)
Create a node in the scene graph.

Parameters

- **id** (*str*) – A unique string identifier for the node
- **children** (*list*) – A list of child nodes of this node. This can contain any object that inherits from *collada.scene.SceneNode*
- **transforms** (*list*) – A list of transformations effecting the node. This can contain any object that inherits from *collada.scene.Transform*
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

`__init__`(id[, children, transforms, xmlnode])	Create a node in the scene graph.
`load`(collada, node, localscope)	
`objects`(tipo[, matrix])	Iterate through all objects under this node that match *tipo*.
`save`()	Saves the geometry back to `xmlnode`.

id = None
The unique string identifier for the node

children = None
A list of child nodes of this node. This can contain any object that inherits from `collada.scene.SceneNode`

matrix = None
A numpy.array of size 4x4 containing a transformation matrix that combines all the transformations in `transforms`. This will only be updated after calling `save()`.

xmlnode = None
ElementTree representation of the transform.

objects(*tipo*, *matrix=None*)
Iterate through all objects under this node that match *tipo*. The objects will be bound and transformed via the scene transformations.

Parameters

- **tipo** (*str*) – A string for the desired object type. This can be one of 'geometry', 'camera', 'light', or 'controller'.
- **matrix** (*numpy.matrix*) – An optional transformation matrix

Return type generator that yields the type specified

save()
Saves the geometry back to `xmlnode`. Also updates `matrix` if `transforms` has been modified.

collada.scene.NodeNode

class collada.scene.**NodeNode**(*node*, *xmlnode=None*)
Bases: `collada.scene.Node`

Represents a node being instantiated in a scene, as defined in the collada <instande_node> tag.

__init__(*node*, *xmlnode=None*)
Creates a node node

Parameters

- **node** (collada.scene.Node) – A node to instantiate in the scene
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(id[, children, transforms, xmlnode])	Create a node in the scene graph.
load(collada, node, localscope)	
objects(tipo[, matrix])	Iterate through all objects under this node that match *tipo*.
save()	Saves the geometry back to *xmlnode*.

node = None
An object of type *collada.scene.Node* representing the node to bind in the scene

xmlnode = None
ElementTree representation of the node node.

save()
Saves the node node back to *xmlnode*

collada.scene.GeometryNode

class collada.scene.**GeometryNode**(*geometry*, *materials=None*, *xmlnode=None*)
Bases: *collada.scene.SceneNode*

Represents a geometry instance in a scene, as defined in the collada <instance_geometry> tag.

__init__(*geometry*, *materials=None*, *xmlnode=None*)
Creates a geometry node

Parameters

- **geometry** (collada.geometry.Geometry) – A geometry to instantiate in the scene
- **materials** (*list*) – A list containing items of type *collada.scene.MaterialNode*. Each of these represents a material that the geometry should be bound to.
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(geometry[, materials, xmlnode])	Creates a geometry node
load(collada, node)	
objects(tipo[, matrix])	Yields a *collada.geometry.BoundGeometry* if tipo=='geometry'
save()	Saves the geometry node back to *xmlnode*

geometry = None
An object of type *collada.geometry.Geometry* representing the geometry to bind in the scene

materials = None
A list containing items of type *collada.scene.MaterialNode*. Each of these represents a material that the geometry is bound to.

xmlnode = None
ElementTree representation of the geometry node.

objects(*tipo*, *matrix=None*)

Yields a *collada.geometry.BoundGeometry* if `tipo=='geometry'`

save()
Saves the geometry node back to *xmlnode*

collada.scene.ControllerNode

class collada.scene.**ControllerNode**(*controller*, *materials*, *xmlnode=None*)
Bases: *collada.scene.SceneNode*

Represents a controller instance in a scene, as defined in the collada <instance_controller> tag. **This class is highly experimental. More support will be added in version 0.4.**

__init__(*controller*, *materials*, *xmlnode=None*)
Creates a controller node

Parameters

- **controller** (*collada.controller.Controller*) – A controller to instantiate in the scene
- **materials** (*list*) – A list containing items of type *collada.scene.MaterialNode*. Each of these represents a material that the controller should be bound to.
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(controller, materials[, xmlnode])	Creates a controller node
load(collada, node)	
objects(tipo[, matrix])	Yields a *collada.controller.BoundController* if `tipo=='controller'`
save()	Saves the controller node back to *xmlnode*

controller = None
An object of type *collada.controller.Controller* representing the controller being instantiated in the scene

materials = None
A list containing items of type *collada.scene.MaterialNode*. Each of these represents a material that the controller is bound to.

xmlnode = None
ElementTree representation of the controller node.

objects(*tipo*, *matrix=None*)
Yields a *collada.controller.BoundController* if `tipo=='controller'`

save()
Saves the controller node back to *xmlnode*

collada.scene.MaterialNode

class collada.scene.**MaterialNode**(*symbol*, *target*, *inputs*, *xmlnode=None*)
Bases: *collada.scene.SceneNode*

Represents a material being instantiated in a scene, as defined in the collada <instance_material> tag.

__init__(*symbol*, *target*, *inputs*, *xmlnode=None*)
Creates a material node

Parameters

- **symbol** (*str*) – The symbol within a geometry this material should be bound to
- **target** (collada.material.Material) – The material object being bound to
- **inputs** (*list*) – A list of tuples of the form `(semantic, input_semantic, set)` mapping texcoords or other inputs to material input channels, e.g. `('TEX0', 'TEXCOORD', '0')` would map the effect parameter `'TEX0'` to the `'TEXCOORD'` semantic of the geometry, using texture coordinate set `0`.
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(symbol, target, inputs[, xmlnode])	Creates a material node
load(collada, node)	
objects()	
save()	Saves the material node back to *xmlnode*

symbol = None
The symbol within a geometry this material should be bound to

target = None
An object of type *collada.material.Material* representing the material object being bound to

inputs = None
A list of tuples of the form `(semantic, input_semantic, set)` mapping texcoords or other inputs to material input channels, e.g. `('TEX0', 'TEXCOORD', '0')` would map the effect parameter `'TEX0'` to the `'TEXCOORD'` semantic of the geometry, using texture coordinate set `0`.

xmlnode = None
ElementTree representation of the material node.

save()
Saves the material node back to *xmlnode*

collada.scene.LightNode

class collada.scene.**LightNode**(*light*, *xmlnode=None*)
Bases: *collada.scene.SceneNode*

Represents a light being instantiated in a scene, as defined in the collada <instance_light> tag.

__init__(*light*, *xmlnode=None*)
Create a light instance

Parameters

- **light** (collada.light.Light) – The light being instantiated
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(light[, xmlnode])	Create a light instance
load(collada, node)	
objects(tipo[, matrix])	Yields a *collada.light.BoundLight* if tipo=='light'
save()	Saves the light node back to *xmlnode*

light = None
An object of type *collada.light.Light* representing the instantiated light

xmlnode = None
ElementTree representation of the light node.

objects(*tipo*, *matrix=None*)
Yields a *collada.light.BoundLight* if tipo=='light'

save()
Saves the light node back to *xmlnode*

collada.scene.CameraNode

class collada.scene.**CameraNode**(*camera*, *xmlnode=None*)
Bases: *collada.scene.SceneNode*

Represents a camera being instantiated in a scene, as defined in the collada <instance_camera> tag.

__init__(*camera*, *xmlnode=None*)
Create a camera instance

Parameters

- **camera** (collada.camera.Camera) – The camera being instantiated
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(camera[, xmlnode])	Create a camera instance
load(collada, node)	
objects(tipo[, matrix])	Yields a *collada.camera.BoundCamera* if tipo=='camera'
save()	Saves the camera node back to *xmlnode*

camera = None
An object of type *collada.camera.Camera* representing the instantiated camera

xmlnode = None
ElementTree representation of the camera node.

objects(*tipo*, *matrix=None*)
Yields a *collada.camera.BoundCamera* if tipo=='camera'

save()
Saves the camera node back to *xmlnode*

collada.scene.ExtraNode

class collada.scene.**ExtraNode**(*xmlnode*)
Bases: *collada.scene.SceneNode*

Represents extra information in a scene, as defined in a collada <extra> tag.

__init__(*xmlnode*)
Create an extra node which stores arbitrary xml

Parameters **xmlnode** – Should be an ElementTree instance of tag type <extra>

Methods

__init__(xmlnode)	Create an extra node which stores arbitrary xml
load(collada, node)	
objects(tipo[, matrix])	
save()	

xmlnode = None
ElementTree representation of the extra node.

collada.scene.Transform

class collada.scene.**Transform**
Bases: *collada.common.DaeObject*

Base class for all transformation types

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

load(collada, localscope, node)	Load and return a class instance from an XML node.
save()	

load(*collada*, *localscope*, *node*)
Load and return a class instance from an XML node.

Inspect the data inside node, which must match this class tag and create an instance out of it.

Parameters

- **collada** (collada.Collada) – The collada file object where this object lives
- **localscope** (*dict*) – If there is a local scope where we should look for local ids (sid) this is the dictionary. Otherwise empty dict ({})
- **node** – An Element from python's ElementTree API

collada.scene.TranslateTransform

class collada.scene.**TranslateTransform**(*x*, *y*, *z*, *xmlnode=None*)
Bases: *collada.scene.Transform*

Contains a translation transformation as defined in the collada <translate> tag.

__init__(*x*, *y*, *z*, *xmlnode=None*)
Creates a translation transformation

Parameters

- **x** (*float*) – x coordinate
- **y** (*float*) – y coordinate
- **z** (*float*) – z coordinate
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(x, y, z[, xmlnode])	Creates a translation transformation
load(collada, node)	
save()	

x = None
x coordinate

y = None
y coordinate

z = None
z coordinate

matrix = None
The resulting transformation matrix. This will be a numpy.array of size 4x4.

xmlnode = None
ElementTree representation of the transform.

collada.scene.RotateTransform

class collada.scene.**RotateTransform**(*x*, *y*, *z*, *angle*, *xmlnode=None*)
Bases: *collada.scene.Transform*

Contains a rotation transformation as defined in the collada <rotate> tag.

__init__(*x*, *y*, *z*, *angle*, *xmlnode=None*)
Creates a rotation transformation

Parameters

- **x** (*float*) – x coordinate
- **y** (*float*) – y coordinate
- **z** (*float*) – z coordinate
- **angle** (*float*) – angle of rotation, in radians

- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(x, y, z, angle[, xmlnode])	Creates a rotation transformation
load(collada, node)	
save()	

x = None
x coordinate

y = None
y coordinate

z = None
z coordinate

angle = None
angle of rotation, in radians

matrix = None
The resulting transformation matrix. This will be a numpy.array of size 4x4.

xmlnode = None
ElementTree representation of the transform.

collada.scene.ScaleTransform

class collada.scene.**ScaleTransform**(*x*, *y*, *z*, *xmlnode=None*)
Bases: *collada.scene.Transform*

Contains a scale transformation as defined in the collada <scale> tag.

__init__(*x*, *y*, *z*, *xmlnode=None*)
Creates a scale transformation

Parameters

- **x** (*float*) – x coordinate
- **y** (*float*) – y coordinate
- **z** (*float*) – z coordinate
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(x, y, z[, xmlnode])	Creates a scale transformation
load(collada, node)	
save()	

x = None
x coordinate

y = None
y coordinate

z = None
z coordinate

matrix = None
The resulting transformation matrix. This will be a numpy.array of size 4x4.

xmlnode = None
ElementTree representation of the transform.

collada.scene.MatrixTransform

class collada.scene.**MatrixTransform**(*matrix*, *xmlnode=None*)
Bases: *collada.scene.Transform*

Contains a matrix transformation as defined in the collada <matrix> tag.

__init__(*matrix*, *xmlnode=None*)
Creates a matrix transformation

Parameters

- **matrix** (*numpy.array*) – This should be an unshaped numpy array of floats of length 16
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(matrix[, xmlnode])	Creates a matrix transformation
load(collada, node)	
save()	

matrix = None
The resulting transformation matrix. This will be a numpy.array of size 4x4.

xmlnode = None
ElementTree representation of the transform.

collada.scene.LookAtTransform

class collada.scene.**LookAtTransform**(*eye*, *interest*, *upvector*, *xmlnode=None*)
Bases: *collada.scene.Transform*

Contains a transformation for aiming a camera as defined in the collada <lookat> tag.

__init__(*eye*, *interest*, *upvector*, *xmlnode=None*)
Creates a lookat transformation

Parameters

- **eye** (*numpy.array*) – An unshaped numpy array of floats of length 3 containing the position of the eye

- **interest** (*numpy.array*) – An unshaped numpy array of floats of length 3 containing the point of interest
- **upvector** (*numpy.array*) – An unshaped numpy array of floats of length 3 containing the up-axis direction
- **xmlnode** – When loaded, the xmlnode it comes from

Methods

__init__(eye, interest, upvector[, xmlnode])	Creates a lookat transformation
load(collada, node)	
save()	

eye = None
A numpy array of length 3 containing the position of the eye

interest = None
A numpy array of length 3 containing the point of interest

upvector = None
A numpy array of length 3 containing the up-axis direction

matrix = None
The resulting transformation matrix. This will be a numpy.array of size 4x4.

xmlnode = None
ElementTree representation of the transform.

3.1.11 Bound

collada.geometry.BoundGeometry	A geometry bound to a transform matrix and material mapping.
collada.primitive.BoundPrimitive	A *collada.primitive.Primitive* bound to a transform matrix and material mapping.
collada.triangleset.BoundTriangleSet	A triangle set bound to a transform matrix and materials mapping.
collada.lineset.BoundLineSet	A line set bound to a transform matrix and materials mapping.
collada.polylist.BoundPolylist	A polylist bound to a transform matrix and materials mapping.
collada.polygons.BoundPolygons	Polygons bound to a transform matrix and materials mapping.
collada.camera.BoundCamera	Base class for bound cameras
collada.camera.BoundPerspectiveCamera	Perspective camera bound to a scene with a transform.
collada.camera.BoundOrthographicCamera	Orthographic camera bound to a scene with a transform.
collada.controller.BoundController	Base class for a controller bound to a transform matrix and materials mapping.
collada.controller.BoundMorph	A morph bound to a transform matrix and materials mapping.
collada.controller.BoundSkin	A skin bound to a transform matrix and materials mapping.

Continued on next page

Table 3.68 – continued from previous page

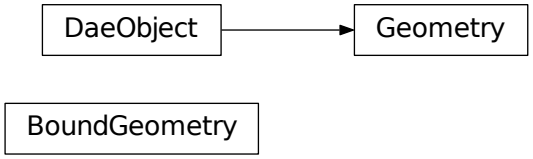
collada.controller.BoundSkinPrimitive	A bound skin bound to a primitive.
collada.light.BoundLight	Base class for bound lights
collada.light.BoundAmbientLight	Ambient light bound to a scene with transformation.
collada.light.BoundDirectionalLight	Directional light bound to a scene with transformation.
collada.light.BoundPointLight	Point light bound to a scene with transformation.
collada.light.BoundSpotLight	Spot light bound to a scene with transformation.
collada.primitive.BoundPrimitive	A *collada.primitive.Primitive* bound to a transform matrix and material mapping.

collada.geometry.BoundGeometry

class collada.geometry.**BoundGeometry**(*geom*, *matrix*, *materialnodebysymbol*)

Bases: object

A geometry bound to a transform matrix and material mapping. This gets created when a geometry is instantiated in a scene. Do not create this manually.



__init__(*geom*, *matrix*, *materialnodebysymbol*)

Methods

__init__(geom, matrix, materialnodebysymbol)	
primitives()	Returns an iterator that iterates through the primitives in the bound geometry.

matrix = None
The matrix bound to

materialnodebysymbol = None
Dictionary with the material symbols inside the primitive assigned to *collada.scene.MaterialNode* defined in the scene

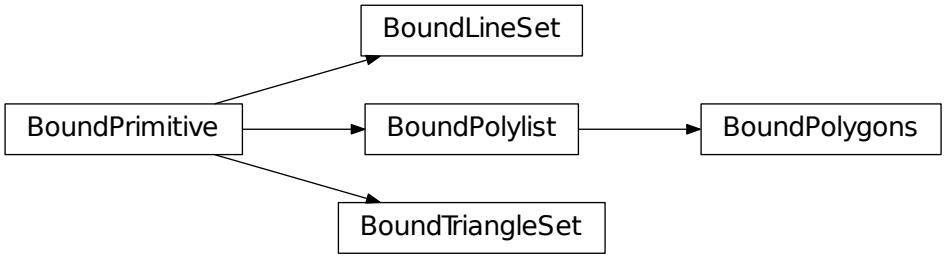
original = None
The original *collada.geometry.Geometry* object this is bound to

primitives()
Returns an iterator that iterates through the primitives in the bound geometry. Each value returned will be of base type *collada.primitive.BoundPrimitive*

collada.primitive.BoundPrimitive

class collada.primitive.**BoundPrimitive**
Bases: object

A *collada.primitive.Primitive* bound to a transform matrix and material mapping.



__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

Methods

shapes()	Iterate through the items in this primitive.

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.
vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

shapes()
Iterate through the items in this primitive. The shape returned depends on the primitive type. Examples: Triangle, Polygon.

vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array. The values will be transformed according to the bound transformation matrix.

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array. The values will be transformed according to the bound transformation matrix.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array. The values will be transformed according to the bound transformation matrix.

vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`. The values will be transformed according to the bound transformation matrix.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`. The values will be transformed according to the bound transformation matrix.

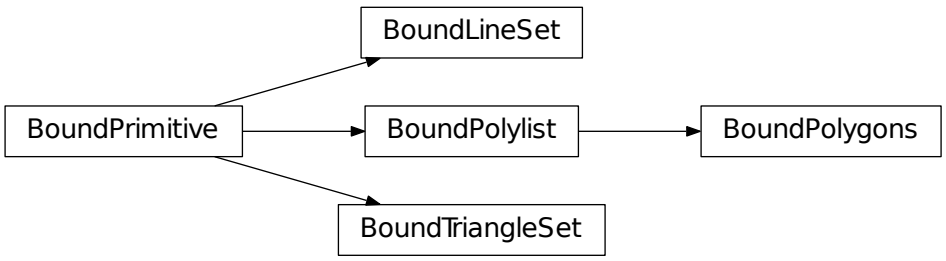
texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates. The values will be transformed according to the bound transformation matrix.

collada.triangleset.BoundTriangleSet

class `collada.triangleset.`**BoundTriangleSet**(*ts*, *matrix*, *materialnodebysymbol*)
Bases: *collada.primitive.BoundPrimitive*

A triangle set bound to a transform matrix and materials mapping.

- If `T` is an instance of *collada.triangleset.BoundTriangleSet*, then `len(T)` returns the number of triangles in the set. `T[i]` returns the i^{th} triangle in the set.



__init__(*ts*, *matrix*, *materialnodebysymbol*)
Create a bound triangle set from a triangle set, transform and material mapping. This gets created when a triangle set is instantiated in a scene. Do not create this manually.

Methods

__init__(ts, matrix, materialnodebysymbol)	Create a bound triangle set from a triangle set, transform and material mapping.
generateNormals()	If `normals` is *None* or you wish for normals to be recomputed, call this method to recompute them.
shapes()	Iterate through all the triangles contained in the set.
triangles()	Iterate through all the triangles contained in the set.

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.
vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array. The values will be transformed according to the bound transformation matrix.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`. The values will be transformed according to the bound transformation matrix.

texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates. The values will be transformed according to the bound transformation matrix.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array. The values will be transformed according to the bound transformation matrix.

vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array. The values will be transformed according to the bound transformation matrix.

vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`. The values will be transformed according to the bound transformation matrix.

triangles()
Iterate through all the triangles contained in the set.

Return type generator of *collada.triangleset.Triangle*

shapes()
Iterate through all the triangles contained in the set.

Return type generator of *collada.triangleset.Triangle*

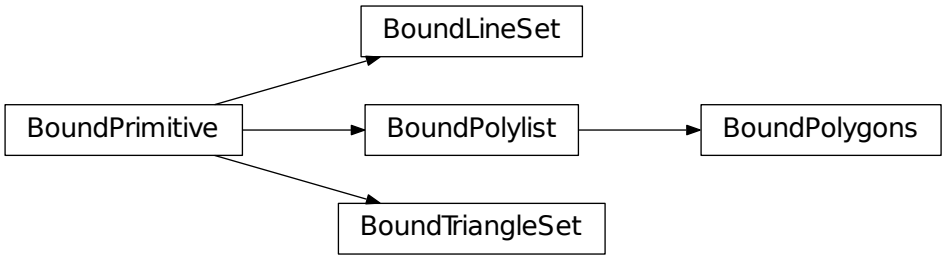
generateNormals()
If `normals` is *None* or you wish for normals to be recomputed, call this method to recompute them.

collada.lineset.BoundLineSet

class collada.lineset.**BoundLineSet**(*ls*, *matrix*, *materialnodebysymbol*)
Bases: *collada.primitive.BoundPrimitive*

A line set bound to a transform matrix and materials mapping.

- If `bs` is an instance of *collada.lineset.BoundLineSet*, `len(bs)` returns the number of lines in the set and `bs[i]` returns the i^{th} line in the set.



__init__(*ls*, *matrix*, *materialnodebysymbol*)
Create a bound line set from a line set, transform and material mapping. This gets created when a line set is instantiated in a scene. Do not create this manually.

Methods

__init__(ls, matrix, materialnodebysymbol)	Create a bound line set from a line set, transform and material mapping.
lines()	Iterate through all the lines contained in the set.
shapes()	Iterate through all the lines contained in the set.

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.
vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array. The values will be transformed according to the bound transformation matrix.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`. The values will be transformed according to the bound transformation matrix.

texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates. The values will be transformed according to the bound transformation matrix.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array. The values will be transformed according to the bound transformation matrix.

vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array. The values will be transformed according to the bound transformation matrix.

vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`. The values will be transformed according to the bound transformation matrix.

lines()
Iterate through all the lines contained in the set.

Return type generator of *collada.lineset.Line*

shapes()
Iterate through all the lines contained in the set.

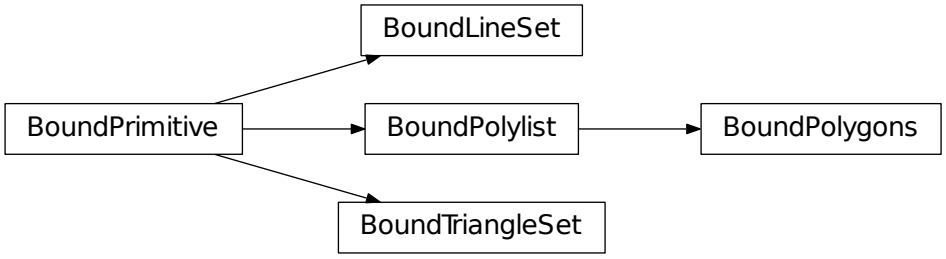
Return type generator of *collada.lineset.Line*

collada.polylist.BoundPolylist

class collada.polylist.**BoundPolylist**(*pl*, *matrix*, *materialnodebysymbol*)
Bases: *collada.primitive.BoundPrimitive*

A polylist bound to a transform matrix and materials mapping.

- If `P` is an instance of *collada.polylist.BoundPolylist*, then `len(P)` returns the number of polygons in the set. `P[i]` returns the i^{th} polygon in the set.



__init__(*pl*, *matrix*, *materialnodebysymbol*)
Create a bound polylist from a polylist, transform and material mapping. This gets created when a polylist is instantiated in a scene. Do not create this manually.

Methods

__init__(pl, matrix, materialnodebysymbol)	Create a bound polylist from a polylist, transform and material mapping.
polygons()	Iterate through all the polygons contained in the set.
shapes()	Iterate through all the polygons contained in the set.
triangleset()	This performs a simple triangulation of the polylist using the fanning method.

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.

Continued on next page

Table 3.77 – continued from previous page

vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array. The values will be transformed according to the bound transformation matrix.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`. The values will be transformed according to the bound transformation matrix.

texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates. The values will be transformed according to the bound transformation matrix.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array. The values will be transformed according to the bound transformation matrix.

vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array. The values will be transformed according to the bound transformation matrix.

vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`. The values will be transformed according to the bound transformation matrix.

triangleset()
This performs a simple triangulation of the polylist using the fanning method.

Return type *collada.triangleset.BoundTriangleSet*

polygons()
Iterate through all the polygons contained in the set.

Return type generator of *collada.polylist.Polygon*

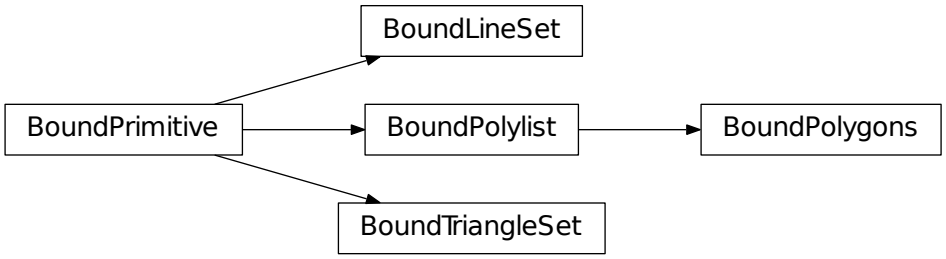
shapes()
Iterate through all the polygons contained in the set.

Return type generator of *collada.polylist.Polygon*

collada.polygons.BoundPolygons

class collada.polygons.**BoundPolygons**(*pl*, *matrix*, *materialnodebysymbol*)
Bases: *collada.polylist.BoundPolylist*

Polygons bound to a transform matrix and materials mapping.



__init__ (*pl*, *matrix*, *materialnodebysymbol*)
Create a BoundPolygons from a Polygons, transform and material mapping

Methods

__init__(pl, matrix, materialnodebysymbol)	Create a BoundPolygons from a Polygons, transform and material mapping
polygons()	Iterate through all the polygons contained in the set.
shapes()	Iterate through all the polygons contained in the set.
triangleset()	This performs a simple triangulation of the polylist using the fanning method.

Attributes

normal	Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array.
normal_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.
texcoord_indexset	Read-only tuple of texture coordinate index arrays.
texcoordset	Read-only tuple of texture coordinate arrays.
vertex	Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array.
vertex_index	Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive.

normal
Read-only numpy.array of size Nx3 where N is the number of normal values in the primitive's normal source array. The values will be transformed according to the bound transformation matrix.

normal_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual normal values, one can use this array to select into the normals array, e.g. `normal[normal_index]`. The values will be transformed according to the bound transformation matrix.

polygons()
Iterate through all the polygons contained in the set.

Return type generator of *collada.polylist.Polygon*

shapes()
Iterate through all the polygons contained in the set.

Return type generator of *collada.polylist.Polygon*

texcoord_indexset
Read-only tuple of texture coordinate index arrays. Each value is a numpy.array of size Nx2 where N is the number of vertices in the primitive. To get the actual texture coordinates, one can use the array to select into the texcoordset array, e.g. `texcoordset[0][texcoord_indexset[0]]` would select the first set of texture coordinates. The values will be transformed according to the bound transformation matrix.

texcoordset
Read-only tuple of texture coordinate arrays. Each value is a numpy.array of size Nx2 where N is the number of texture coordinates in the primitive's source array. The values will be transformed according to the bound transformation matrix.

triangleset()
This performs a simple triangulation of the polylist using the fanning method.

Return type *collada.triangleset.BoundTriangleSet*

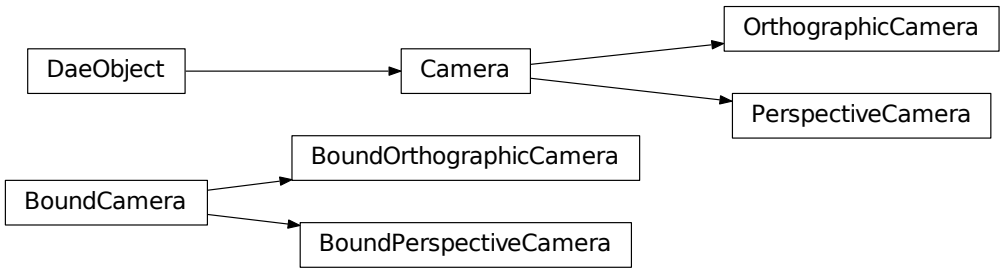
vertex
Read-only numpy.array of size Nx3 where N is the number of vertex points in the primitive's vertex source array. The values will be transformed according to the bound transformation matrix.

vertex_index
Read-only numpy.array of size Nx3 where N is the number of vertices in the primitive. To get the actual vertex points, one can use this array to select into the vertex array, e.g. `vertex[vertex_index]`. The values will be transformed according to the bound transformation matrix.

collada.camera.BoundCamera

class collada.camera.**BoundCamera**
Bases: object

Base class for bound cameras

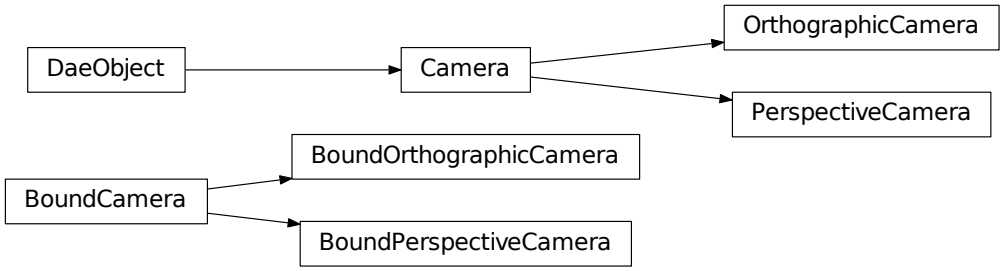


__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

collada.camera.BoundPerspectiveCamera

class collada.camera.**BoundPerspectiveCamera**(*cam*, *matrix*)
Bases: *collada.camera.BoundCamera*

Perspective camera bound to a scene with a transform. This gets created when a camera is instantiated in a scene. Do not create this manually.



__init__(*cam*, *matrix*)

Methods

__init__(cam, matrix)

xfov = None
Horizontal field of view, in degrees

yfov = None
Vertical field of view, in degrees

aspect_ratio = None
Aspect ratio of the field of view

znear = None
Distance to the near clipping plane

zfar = None
Distance to the far clipping plane

matrix = None
The matrix bound to

position = None
The position of the camera

direction = None
The direction the camera is facing

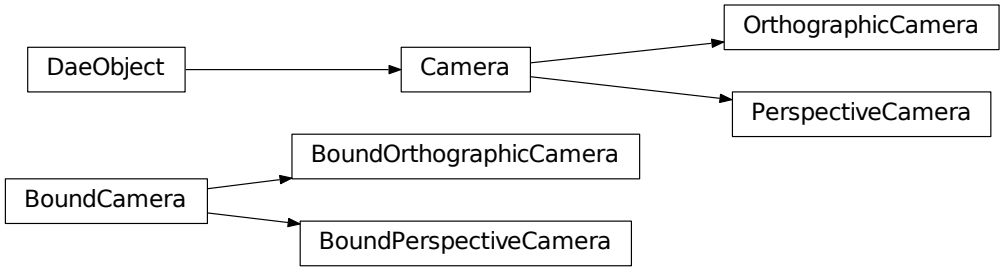
up = None
The up vector of the camera

original = None
Original *collada.camera.PerspectiveCamera* object this is bound to.

collada.camera.BoundOrthographicCamera

class collada.camera.**BoundOrthographicCamera**(*cam*, *matrix*)
Bases: *collada.camera.BoundCamera*

Orthographic camera bound to a scene with a transform. This gets created when a camera is instantiated in a scene. Do not create this manually.



__init__(*cam*, *matrix*)

Methods

__init__(cam, matrix)

xmag = None
Horizontal magnification of the view

ymag = None
Vertical magnification of the view

aspect_ratio = None
Aspect ratio of the field of view

znear = None
Distance to the near clipping plane

zfar = None
Distance to the far clipping plane

matrix = None
The matrix bound to

position = None
The position of the camera

direction = None
The direction the camera is facing

up = None
The up vector of the camera

original = None
Original *collada.camera.OrthographicCamera* object this is bound to.

collada.controller.BoundController

class collada.controller.**BoundController**
Bases: object

Base class for a controller bound to a transform matrix and materials mapping.

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

collada.controller.BoundMorph

class collada.controller.**BoundMorph**(*morph*, *matrix*, *materialnodebysymbol*)
Bases: *collada.controller.BoundController*

A morph bound to a transform matrix and materials mapping.

__init__(*morph*, *matrix*, *materialnodebysymbol*)

Methods

__init__(morph, matrix, materialnodebysymbol)

collada.controller.BoundSkin

class collada.controller.**BoundSkin**(*skin*, *matrix*, *materialnodebysymbol*)
Bases: *collada.controller.BoundController*

A skin bound to a transform matrix and materials mapping.

__init__(*skin*, *matrix*, *materialnodebysymbol*)

Methods

__init__(skin, matrix, materialnodebysymbol)
getJoint(i)
getWeight(i)
primitives()

collada.controller.BoundSkinPrimitive

class collada.controller.**BoundSkinPrimitive**(*primitive*, *boundskin*)
Bases: object

A bound skin bound to a primitive.

__init__ (*primitive*, *boundskin*)

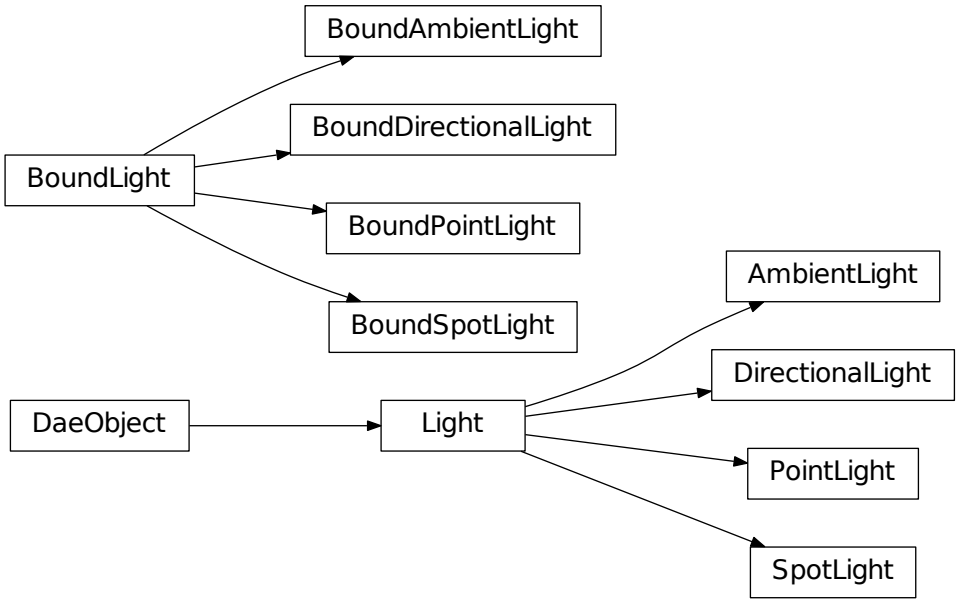
Methods

__init__(primitive, boundskin)
shapes()

collada.light.BoundLight

class collada.light.**BoundLight**
Bases: object

Base class for bound lights

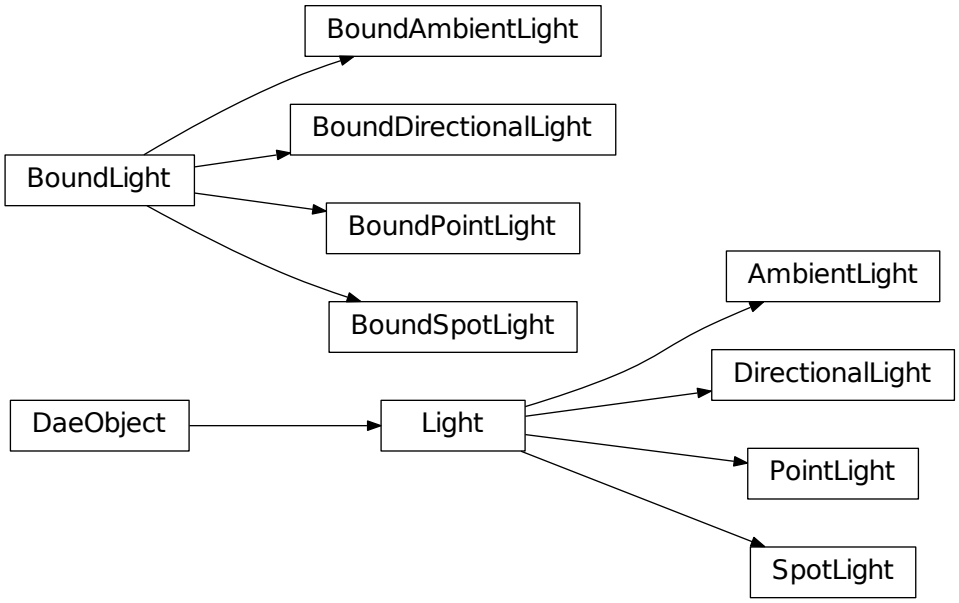


__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

collada.light.BoundAmbientLight

class collada.light.**BoundAmbientLight** (*alight*, *matrix*)
Bases: *collada.light.BoundLight*

Ambient light bound to a scene with transformation. This gets created when a light is instantiated in a scene. Do not create this manually.



__init__ (*alight*, *matrix*)

Methods

__init__(alight, matrix)

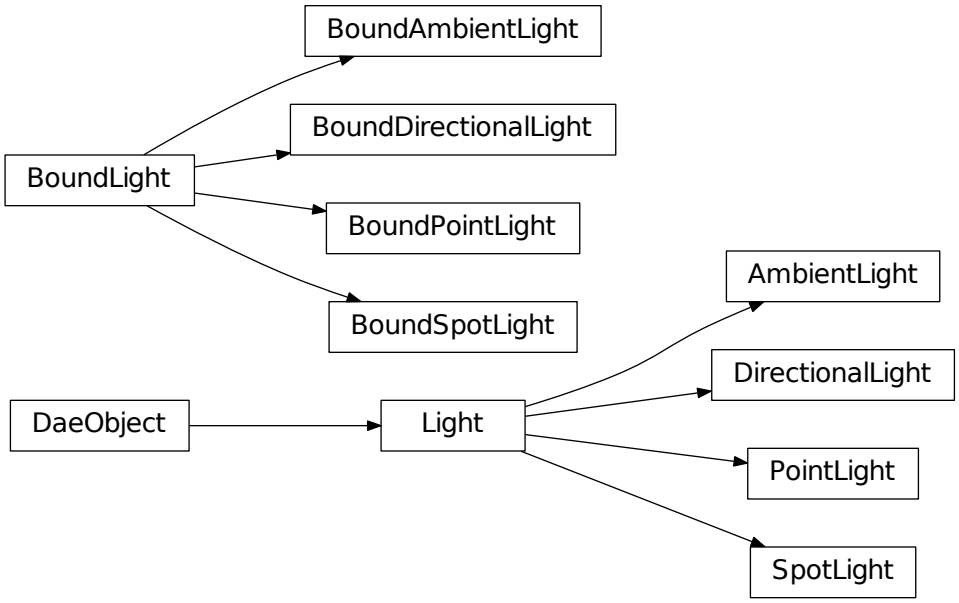
color = None
Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light

original = None
The original *collada.light.AmbientLight* this is bound to

collada.light.BoundDirectionalLight

class collada.light.**BoundDirectionalLight** (*dlight*, *matrix*)
Bases: *collada.light.BoundLight*

Directional light bound to a scene with transformation. This gets created when a light is instantiated in a scene. Do not create this manually.



__init__ (*dlight*, *matrix*)

Methods

__init__(dlight, matrix)

direction = None
Numpy array of length 3 representing the direction of the light in the scene

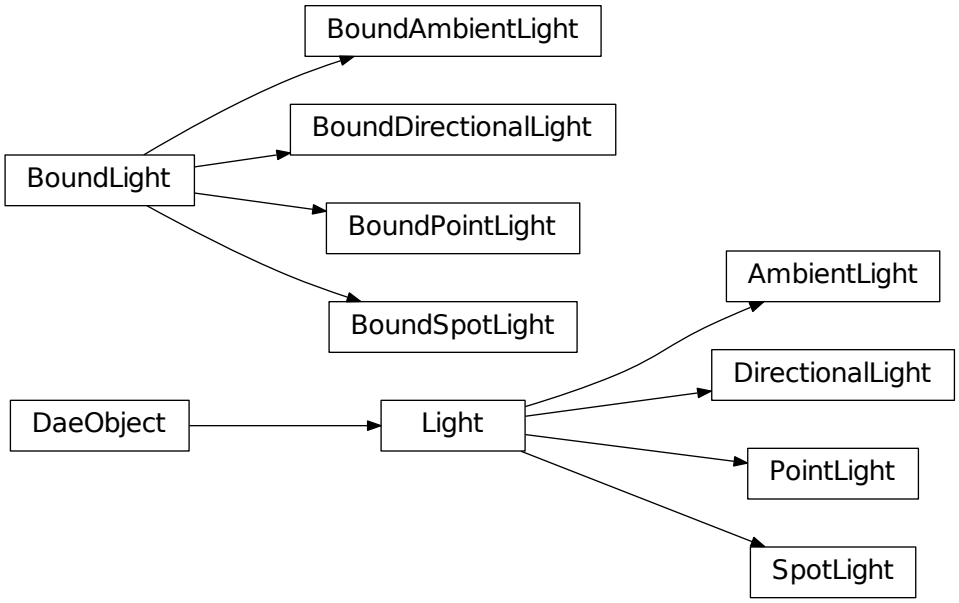
color = None
Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light

original = None
The original *collada.light.DirectionalLight* this is bound to

collada.light.BoundPointLight

class collada.light.**BoundPointLight** (*plight*, *matrix*)
Bases: *collada.light.BoundLight*

Point light bound to a scene with transformation. This gets created when a light is instantiated in a scene. Do not create this manually.



__init__ (*plight*, *matrix*)

Methods

__init__(plight, matrix)

position = None
Numpy array of length 3 representing the position of the light in the scene

color = None
Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light

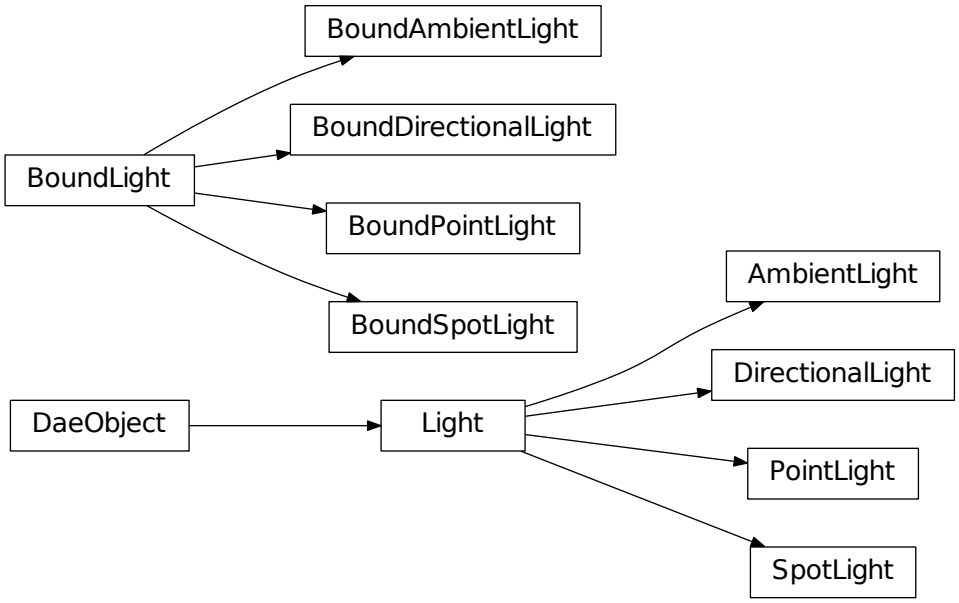
zfar = None
Distance to the far clipping plane

original = None
The original *collada.light.PointLight* this is bound to

collada.light.BoundSpotLight

class collada.light.**BoundSpotLight** (*slight*, *matrix*)
Bases: *collada.light.BoundLight*

Spot light bound to a scene with transformation. This gets created when a light is instantiated in a scene. Do not create this manually.



__init__ (*slight*, *matrix*)

Methods

__init__(slight, matrix)

position = None
Numpy array of length 3 representing the position of the light in the scene

direction = None
Direction of the spot light

up = None
Up vector of the spot light

matrix = None
Transform matrix for the bound light

color = None
Either a tuple of size 3 containing the RGB color value of the light or a tuple of size 4 containing the RGBA color value of the light

original = None
The original *collada.light.SpotLight* this is bound to

3.1.12 Exceptions

collada.common.DaeError	General DAE exception.
collada.common.DaeIncompleteError	Raised when needed data for an object isn't there.
collada.common.DaeBrokenRefError	Raised when a referenced object is not found in the scope.
collada.common.DaeMalformedError	Raised when data is found to be corrupted in some way.
collada.common.DaeUnsupportedError	Raised when some unexpectedly unsupported feature is found.
collada.common.DaeSaveValidationError	Raised when XML validation fails when saving.

collada.common.DaeError

exception collada.common.**DaeError**(*msg*)
Bases: exceptions.Exception

General DAE exception.

collada.common.DaeIncompleteError

exception collada.common.**DaeIncompleteError**(*msg*)
Bases: *collada.common.DaeError*

Raised when needed data for an object isn't there.

collada.common.DaeBrokenRefError

exception collada.common.**DaeBrokenRefError**(*msg*)
Bases: *collada.common.DaeError*

Raised when a referenced object is not found in the scope.

collada.common.DaeMalformedError

exception collada.common.**DaeMalformedError**(*msg*)
Bases: *collada.common.DaeError*

Raised when data is found to be corrupted in some way.

collada.common.DaeUnsupportedError

exception collada.common.**DaeUnsupportedError**(*msg*)
Bases: *collada.common.DaeError*

Raised when some unexpectedly unsupported feature is found.

collada.common.DaeSaveValidationError

exception collada.common.**DaeSaveValidationError**(*msg*)
Bases: *collada.common.DaeError*

Raised when XML validation fails when saving.

3.1.13 Util

collada.util.toUnitVec	Converts the given vector to a unit vector
collada.util.checkSource	Check if a source objects complies with the needed *components* and has the needed length
collada.util.normalize_v3	Normalize a numpy array of 3 component vectors with shape (N,3)
collada.util.IndexedList	Class that combines a list and a dict into a single class

collada.util.toUnitVec

util.**toUnitVec**(*vec*)

Converts the given vector to a unit vector

Parameters **vec** (*numpy.array*) – The vector to transform to unit length

Return type numpy.array

collada.util.checkSource

util.**checkSource**(*source*, *components*, *maxindex*)

Check if a source objects complies with the needed *components* and has the needed length

Parameters

- **source** (collada.source.Source) – A source instance to check
- **components** (*tuple*) – A tuple describing the needed channels, e.g. `('X','Y','Z')`
- **maxindex** (*int*) – The maximum index that refers to this source

collada.util.normalize_v3

util.**normalize_v3**(*arr*)

Normalize a numpy array of 3 component vectors with shape (N,3)

Parameters **arr** (*numpy.array*) – The numpy array to normalize

Return type numpy.array

collada.util.IndexedList

class collada.util.**IndexedList**(*items*, *attrs*)

Bases: list

Class that combines a list and a dict into a single class

- Written by Hugh Bothwell (http://stackoverflow.com/users/33258/hugh-bothwell)
- **Original source available at:** http://stackoverflow.com/questions/5332841/python-list-dict-property-best-practice/5334686#5334686
- Modifications by Jeff Terrace

Given an object, obj, that has a property x, this allows you to create an IndexedList like so: L = IndexedList([], ('x')) o = obj() o.x = 'test' L.append(o) L[0] # = o L['test'] # = o

__init__(*items*, *attrs*)

Methods

__init__(items, attrs)	
append(obj)	
extend(newList)	
get(key[, default])	
insert(ind, new_obj)	
pop([ind])	
remove(ind_or_obj)	

Attributes

count(...)	
index((value, [start, ...)	Raises ValueError if the value is not present.
reverse	L.reverse() – reverse *IN PLACE*
sort	L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*;

count (*value*) → integer – return number of occurrences of value

index (*value*[, *start*[, *stop*]]) → integer – return first index of value.
Raises ValueError if the value is not present.

reverse ()
L.reverse() – reverse *IN PLACE*

sort ()
L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*; cmp(x, y) -> -1, 0, 1

3.1.14 Modules

collada	Main module for collada (pycollada) package.
collada.camera	Contains objects for representing cameras
collada.common	
collada.controller	Contains objects representing controllers.
collada.geometry	Contains objects for representing a geometry.
collada.light	Contains objects for representing lights.
collada.lineset	Module containing classes and functions for the <lines> primitive.
collada.material	Module for material, effect and image loading
collada.polygons	Module containing classes and functions for the <polygons> primitive.
collada.polylist	Module containing classes and functions for the <polylist> primitive.
collada.primitive	Module containing the base class for primitives
collada.scene	This module contains several classes related to the scene graph.
collada.source	Module for managing data sources defined in geometry tags.

Continued on next page

Table 3.93 – continued from previous page

collada.triangleset	Module containing classes and functions for the <triangles> primitive.
collada.util	This module contains utility functions

collada

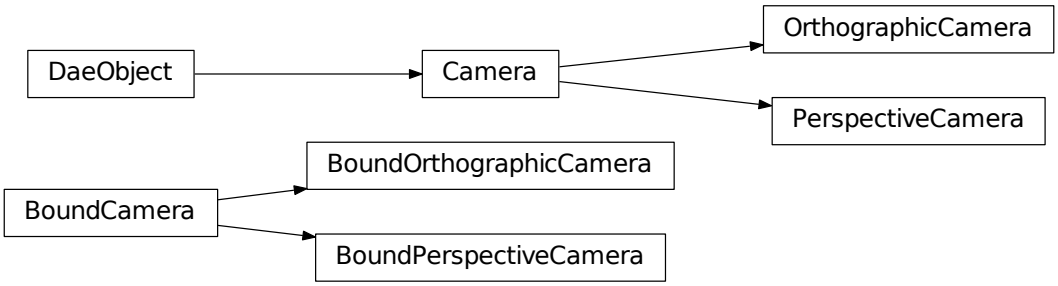
Main module for collada (pycollada) package.

You will find here class *Collada* which is the one to use to access collada file, and some exceptions that are raised in case the input file is not what is expected.

Members

collada.Collada	This is the main class used to create and load collada documents
collada.common.DaeBrokenRefError	Raised when a referenced object is not found in the scope.
collada.common.DaeError	General DAE exception.
collada.common.DaeIncompleteError	Raised when needed data for an object isn't there.
collada.common.DaeMalformedError	Raised when data is found to be corrupted in some way.
collada.common.DaeObject	This class is the abstract interface to all collada objects.
collada.common.DaeUnsupportedError	Raised when some unexpectedly unsupported feature is found.

collada.camera



Contains objects for representing cameras

Members

collada.camera.BoundCamera	Base class for bound cameras
collada.camera.Camera	Base camera class holding data from <camera> tags.

collada.common

Members

collada.common.DaeObject	This class is the abstract interface to all collada objects.
collada.common.DaeError	General DAE exception.
collada.common.DaeIncompleteError	Raised when needed data for an object isn't there.
collada.common.DaeBrokenRefError	Raised when a referenced object is not found in the scope.
collada.common.DaeMalformedError	Raised when data is found to be corrupted in some way.
collada.common.DaeUnsupportedError	Raised when some unexpectedly unsupported feature is found.
collada.common.DaeSaveValidationError	Raised when XML validation fails when saving.

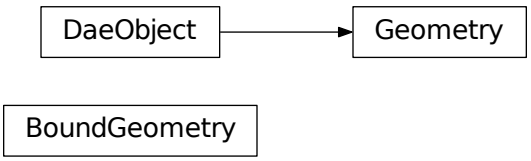
collada.controller

Contains objects representing controllers. Currently has partial support for loading Skin and Morph. **This module is highly experimental. More support will be added in version 0.4.**

Members

collada.controller.BoundController	Base class for a controller bound to a transform matrix and materials mapping.
collada.controller.BoundMorph	A morph bound to a transform matrix and materials mapping.
collada.controller.BoundSkin	A skin bound to a transform matrix and materials mapping.
collada.controller.BoundSkinPrimitive	A bound skin bound to a primitive.
collada.controller.Controller	Base controller class holding data from <controller> tags.
collada.controller.Morph	Class containing data collada holds in the <morph> tag
collada.controller.Skin	Class containing data collada holds in the <skin> tag

collada.geometry

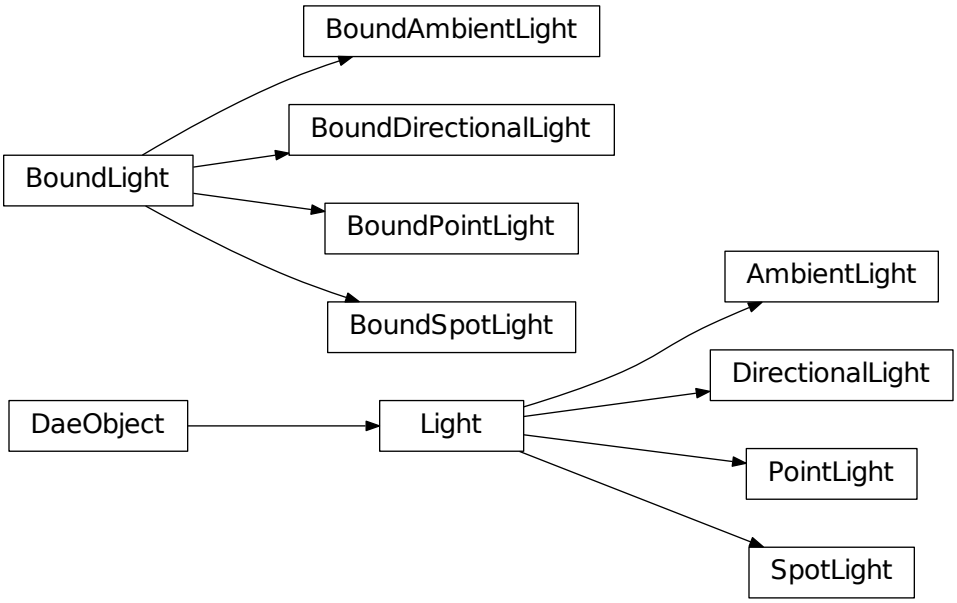


Contains objects for representing a geometry.

Members

collada.geometry.Geometry	A class containing the data coming from a COLLADA <geometry> tag
collada.geometry.BoundGeometry	A geometry bound to a transform matrix and material mapping.

collada.light

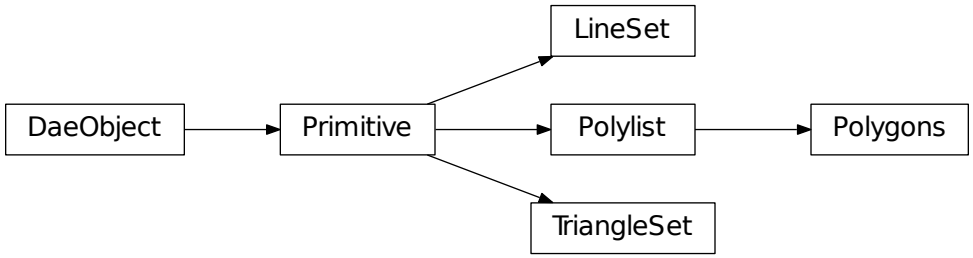


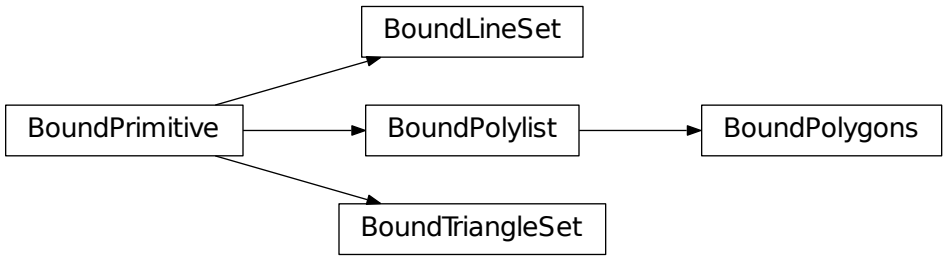
Contains objects for representing lights.

Members

collada.light.AmbientLight	Ambient light as defined in COLLADA tag <ambient>.
collada.light.BoundAmbientLight	Ambient light bound to a scene with transformation.
collada.light.BoundDirectionalLight	Directional light bound to a scene with transformation.
collada.light.BoundLight	Base class for bound lights
collada.light.BoundPointLight	Point light bound to a scene with transformation.
collada.light.BoundSpotLight	Spot light bound to a scene with transformation.
collada.light.DirectionalLight	Directional light as defined in COLLADA tag <directional> tag.
collada.light.Light	Base light class holding data from <light> tags.
collada.light.PointLight	Point light as defined in COLLADA tag <point>.
collada.light.SpotLight	Spot light as defined in COLLADA tag <spot>.

collada.lineset





Module

containing classes and functions for the <lines> primitive.

Members

collada.lineset.BoundLineSet	A line set bound to a transform matrix and materials mapping.
collada.lineset.Line	Single line representation.
collada.lineset.LineSet	Class containing the data COLLADA puts in a <lines> tag, a collection of lines.

collada.material

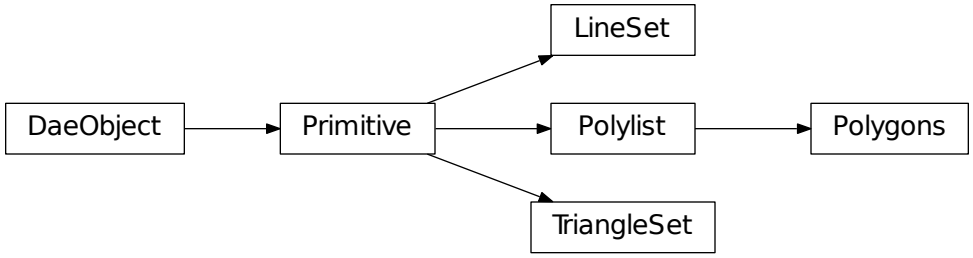
Module for material, effect and image loading

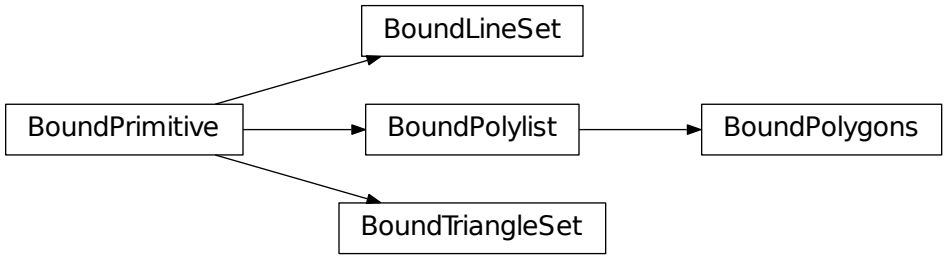
This module contains all the functionality to load and manage: - Images in the image library - Surfaces and samplers2D in effects - Effects (that are now used as materials)

Members

collada.material.CImage	Class containing data coming from a <image> tag.
collada.material.Effect	Class containing data coming from an <effect> tag.
collada.material.Map	Class containing data coming from <texture> tag inside material.
collada.material.Material	Class containing data coming from a <material> tag.
collada.material.Sampler2D	Class containing data coming from <sampler2D> tag in material.
collada.material.Surface	Class containing data coming from a <surface> tag.

collada.polygons



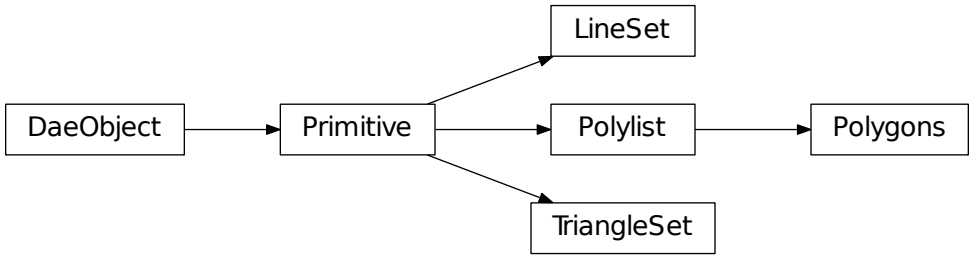


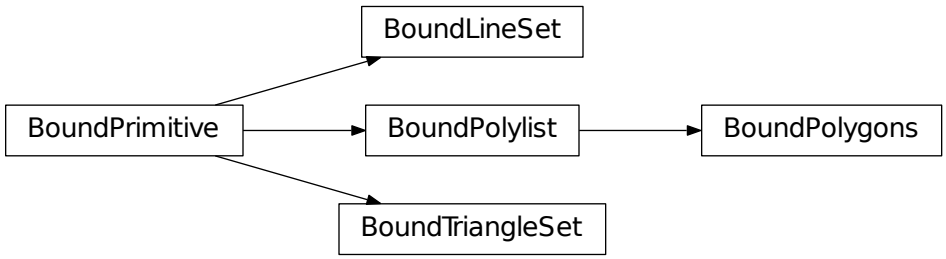
Module containing classes and functions for the <polygons> primitive.

Members

collada.polygons.BoundPolygons	Polygons bound to a transform matrix and materials mapping.
collada.polygons.Polygons	Class containing the data COLLADA puts in a <polygons> tag, a collection of polygons that can have holes.

collada.polylist



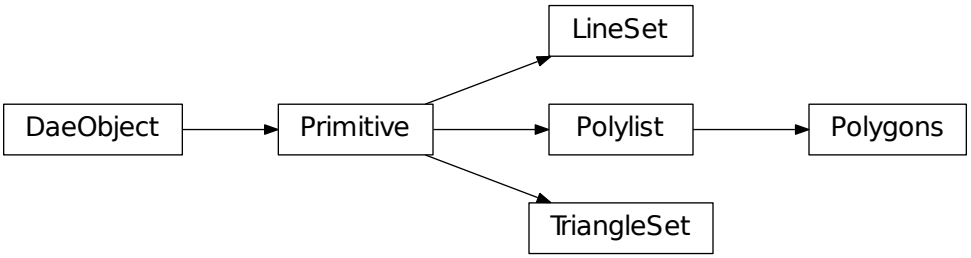


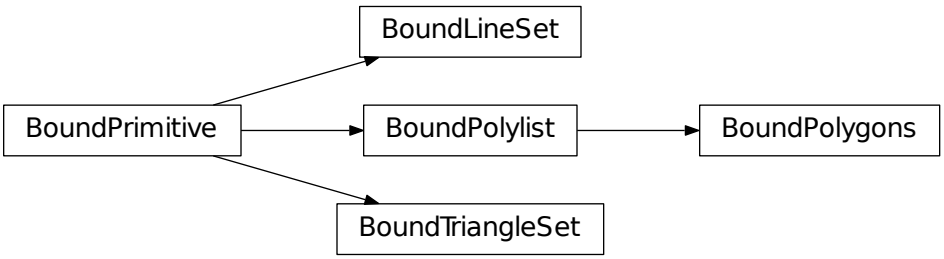
Module containing classes and functions for the <polylist> primitive.

Members

collada.polylist.BoundPolylist	A polylist bound to a transform matrix and materials mapping.
collada.polylist.Polylist	Class containing the data COLLADA puts in a <polylist> tag, a collection of polygons.
collada.polylist.Polygon	Single polygon representation.

collada.primitive





Module containing the base class for primitives

Members

collada.primitive.BoundPrimitive	A *collada.primitive.Primitive* bound to a transform matrix and material mapping.
collada.primitive.Primitive	Base class for all primitive sets like TriangleSet, LineSet, Polylist, etc.

collada.scene

This module contains several classes related to the scene graph.

Supported scene nodes are:

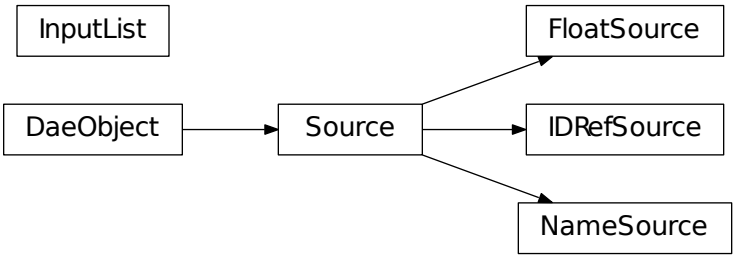
- <node> which is loaded as a Node
- <instance_camera> which is loaded as a CameraNode
- <instance_light> which is loaded as a LightNode
- <instance_material> which is loaded as a MaterialNode

- <instance_geometry> which is loaded as a GeometryNode
- <instance_controller> which is loaded as a ControllerNode
- <scene> which is loaded as a Scene

Members

collada.scene.CameraNode	Represents a camera being instantiated in a scene, as defined in the collada <instance_camera> tag.
collada.scene.ControllerNode	Represents a controller instance in a scene, as defined in the collada <instance_controller> tag.
collada.scene.ExtraNode	Represents extra information in a scene, as defined in a collada <extra> tag.
collada.scene.GeometryNode	Represents a geometry instance in a scene, as defined in the collada <instance_geometry> tag.
collada.scene.LightNode	Represents a light being instantiated in a scene, as defined in the collada <instance_light> tag.
collada.scene.LookAtTransform	Contains a transformation for aiming a camera as defined in the collada <lookat> tag.
collada.scene.MaterialNode	Represents a material being instantiated in a scene, as defined in the collada <instance_material> tag.
collada.scene.MatrixTransform	Contains a matrix transformation as defined in the collada <matrix> tag.
collada.scene.Node	Represents a node object, which is a point on the scene graph, as defined in the collada <node> tag.
collada.scene.RotateTransform	Contains a rotation transformation as defined in the collada <rotate> tag.
collada.scene.ScaleTransform	Contains a scale transformation as defined in the collada <scale> tag.
collada.scene.Scene	The root object for a scene, as defined in a collada <scene> tag
collada.scene.SceneNode	Abstract base class for all nodes within a scene.
collada.scene.Transform	Base class for all transformation types
collada.scene.TranslateTransform	Contains a translation transformation as defined in the collada <translate> tag.

collada.source

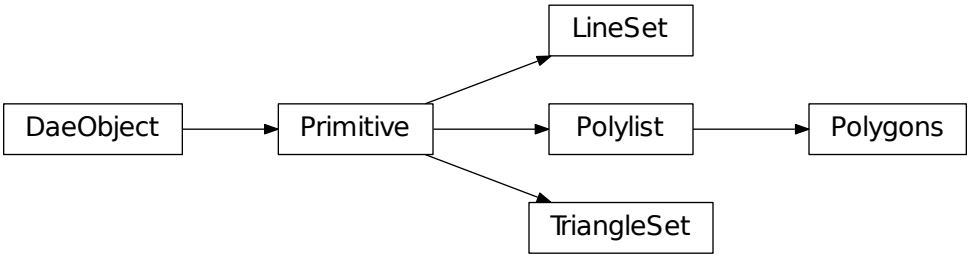


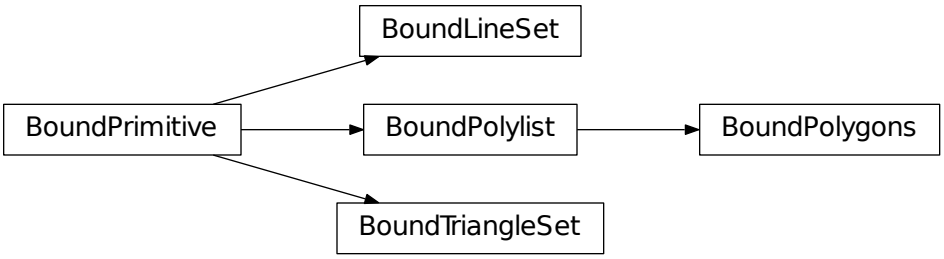
Module for managing data sources defined in geometry tags.

Members

collada.source.FloatSource	Contains a source array of floats, as defined in the collada <float_array> inside a <source>.
collada.source.IDRefSource	Contains a source array of ID references, as defined in the collada <IDREF_array> inside a <source>.
collada.source.InputList	Used for defining input sources to a geometry.
collada.source.NameSource	Contains a source array of strings, as defined in the collada <Name_array> inside a <source>.
collada.source.Source	Abstract class for loading source arrays

collada.triangleset





Module containing classes and functions for the <triangles> primitive.

Members

collada.triangleset.BoundTriangleSet	A triangle set bound to a transform matrix and materials mapping.
collada.triangleset.Triangle	Single triangle representation.
collada.triangleset.TriangleSet	Class containing the data COLLADA puts in a <triangles> tag, a collection of triangles.

collada.util

This module contains utility functions

Members

collada.util.checkSource	Check if a source objects complies with the needed *components* and has the needed length
collada.util.normalize_v3	Normalize a numpy array of 3 component vectors with shape (N,3)
collada.util.toUnitVec	Converts the given vector to a unit vector

3.2 API Reference

3.2.1 collada.asset

Contains COLLADA asset information.

Members

collada.asset.Asset	Defines asset-management information

Continued on next page

Table 3.109 – continued from previous page

collada.asset.Contributor	Defines authoring information for asset management
collada.asset.UP_AXIS	The up-axis of the collada document.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

collada, 100
collada.asset, 109
collada.camera, 100
collada.common, 101
collada.controller, 101
collada.geometry, 101
collada.light, 102
collada.lineset, 103
collada.material, 103
collada.polygons, 104
collada.polylist, 105
collada.primitive, 106
collada.scene, 106
collada.source, 108
collada.triangleset, 109
collada.util, 109

Index

Symbols

__init__() (collada.Collada method), 15
__init__() (collada.asset.Asset method), 19
__init__() (collada.asset.Contributor method), 18
__init__() (collada.camera.BoundCamera method), 87
__init__() (collada.camera.BoundOrthographicCamera method), 89
__init__() (collada.camera.BoundPerspectiveCamera method), 88
__init__() (collada.camera.Camera method), 39
__init__() (collada.camera.OrthographicCamera method), 42
__init__() (collada.camera.PerspectiveCamera method), 40
__init__() (collada.common.DaeObject method), 17
__init__() (collada.controller.BoundController method), 90
__init__() (collada.controller.BoundMorph method), 90
__init__() (collada.controller.BoundSkin method), 90
__init__() (collada.controller.BoundSkinPrimitive method), 91
__init__() (collada.controller.Controller method), 37
__init__() (collada.controller.Morph method), 38
__init__() (collada.controller.Skin method), 37
__init__() (collada.geometry.BoundGeometry method), 78
__init__() (collada.geometry.Geometry method), 21
__init__() (collada.light.AmbientLight method), 47
__init__() (collada.light.BoundAmbientLight method), 92
__init__() (collada.light.BoundDirectionalLight method), 93
__init__() (collada.light.BoundLight method), 91
__init__() (collada.light.BoundPointLight method), 94
__init__() (collada.light.BoundSpotLight method), 95
__init__() (collada.light.DirectionalLight method), 46
__init__() (collada.light.Light method), 45
__init__() (collada.light.PointLight method), 49
__init__() (collada.light.SpotLight method), 51
__init__() (collada.lineset.BoundLineSet method), 82
__init__() (collada.lineset.Line method), 36
__init__() (collada.lineset.LineSet method), 28
__init__() (collada.material.CImage method), 55
__init__() (collada.material.Effect method), 53
__init__() (collada.material.Map method), 58
__init__() (collada.material.Material method), 52
__init__() (collada.material.Sampler2D method), 57
__init__() (collada.material.Surface method), 57
__init__() (collada.polygons.BoundPolygons method), 86
__init__() (collada.polygons.Polygons method), 32
__init__() (collada.polylist.BoundPolylist method), 84
__init__() (collada.polylist.Polygon method), 35
__init__() (collada.polylist.Polylist method), 30
__init__() (collada.primitive.BoundPrimitive method), 79
__init__() (collada.primitive.Primitive method), 24
__init__() (collada.scene.CameraNode method), 72
__init__() (collada.scene.ControllerNode method), 70
__init__() (collada.scene.ExtraNode method), 73
__init__() (collada.scene.GeometryNode method), 69
__init__() (collada.scene.LightNode method), 71
__init__() (collada.scene.LookAtTransform method), 76
__init__() (collada.scene.MaterialNode method), 71
__init__() (collada.scene.MatrixTransform method), 76
__init__() (collada.scene.Node method), 67
__init__() (collada.scene.NodeNode method), 68
__init__() (collada.scene.RotateTransform method), 74
__init__() (collada.scene.ScaleTransform method), 75
__init__() (collada.scene.Scene method), 66
__init__() (collada.scene.SceneNode method), 66
__init__() (collada.scene.Transform method), 73
__init__() (collada.scene.TranslateTransform method), 74
__init__() (collada.source.FloatSource method), 62
__init__() (collada.source.IDRefSource method), 64
__init__() (collada.source.InputList method), 60
__init__() (collada.source.NameSource method), 63
__init__() (collada.source.Source method), 61
__init__() (collada.triangleset.BoundTriangleSet method), 80
__init__() (collada.triangleset.Triangle method), 34

__init__() (collada.triangleset.TriangleSet method), 26
__init__() (collada.util.IndexedList method), 98

A

A_ONE (collada.material.OPAQUE_MODE attribute), 59
addInput() (collada.source.InputList method), 60
almostEqual() (collada.material.Effect method), 55
ambient (collada.material.Effect attribute), 55
AmbientLight (class in collada.light), 47
angle (collada.scene.RotateTransform attribute), 75
animations (collada.Collada attribute), 16
aspect_ratio (collada.camera.BoundOrthographicCamera attribute), 89
aspect_ratio (collada.camera.BoundPerspectiveCamera attribute), 88
aspect_ratio (collada.camera.OrthographicCamera attribute), 43
aspect_ratio (collada.camera.PerspectiveCamera attribute), 41
Asset (class in collada.asset), 19
assetInfo (collada.Collada attribute), 16
author (collada.asset.Contributor attribute), 18
authoring_tool (collada.asset.Contributor attribute), 18

B

bind() (collada.camera.OrthographicCamera method), 43
bind() (collada.camera.PerspectiveCamera method), 41
bind() (collada.controller.Morph method), 38
bind() (collada.controller.Skin method), 38
bind() (collada.geometry.Geometry method), 23
bind() (collada.light.AmbientLight method), 48
bind() (collada.light.DirectionalLight method), 47
bind() (collada.light.PointLight method), 50
bind() (collada.light.SpotLight method), 52
bind() (collada.lineset.LineSet method), 29
bind() (collada.polygons.Polygons method), 33
bind() (collada.polylist.Polylist method), 31
bind() (collada.primitive.Primitive method), 25
bind() (collada.triangleset.TriangleSet method), 27
BoundAmbientLight (class in collada.light), 91
BoundCamera (class in collada.camera), 87
BoundController (class in collada.controller), 90
BoundDirectionalLight (class in collada.light), 92
BoundGeometry (class in collada.geometry), 78
BoundLight (class in collada.light), 91
BoundLineSet (class in collada.lineset), 82
BoundMorph (class in collada.controller), 90
BoundOrthographicCamera (class in collada.camera), 89
BoundPerspectiveCamera (class in collada.camera), 88
BoundPointLight (class in collada.light), 93
BoundPolygons (class in collada.polygons), 85
BoundPolylist (class in collada.polylist), 84
BoundPrimitive (class in collada.primitive), 79
BoundSkin (class in collada.controller), 90
BoundSkinPrimitive (class in collada.controller), 90
BoundSpotLight (class in collada.light), 94
BoundTriangleSet (class in collada.triangleset), 80
bumpmap (collada.material.Effect attribute), 54

C

Camera (class in collada.camera), 39
camera (collada.scene.CameraNode attribute), 72
CameraNode (class in collada.scene), 72
cameras (collada.Collada attribute), 16
checkSource() (collada.util method), 98
children (collada.scene.Node attribute), 68
CImage (class in collada.material), 55
Collada (class in collada), 15
collada (collada.geometry.Geometry attribute), 22
collada (collada.scene.Scene attribute), 66
collada (module), 100
collada.asset (module), 109
collada.camera (module), 100
collada.common (module), 101
collada.controller (module), 101
collada.geometry (module), 101
collada.light (module), 102
collada.lineset (module), 103
collada.material (module), 103
collada.polygons (module), 104
collada.polylist (module), 105
collada.primitive (module), 106
collada.scene (module), 106
collada.source (module), 108
collada.triangleset (module), 109
collada.util (module), 109
color (collada.light.AmbientLight attribute), 48
color (collada.light.BoundAmbientLight attribute), 92
color (collada.light.BoundDirectionalLight attribute), 93
color (collada.light.BoundPointLight attribute), 94
color (collada.light.BoundSpotLight attribute), 95
color (collada.light.DirectionalLight attribute), 46
color (collada.light.PointLight attribute), 49
color (collada.light.SpotLight attribute), 52
comments (collada.asset.Contributor attribute), 18
components (collada.source.FloatSource attribute), 62
components (collada.source.IDRefSource attribute), 65
components (collada.source.NameSource attribute), 64
constant_att (collada.light.PointLight attribute), 50
constant_att (collada.light.SpotLight attribute), 52
Contributor (class in collada.asset), 18
contributors (collada.asset.Asset attribute), 20
Controller (class in collada.controller), 36
controller (collada.scene.ControllerNode attribute), 70
ControllerNode (class in collada.scene), 70
controllers (collada.Collada attribute), 16
copyright (collada.asset.Contributor attribute), 18

count() (collada.util.IndexedList method), 99
created (collada.asset.Asset attribute), 19
createLineSet() (collada.geometry.Geometry method), 22
createPolygons() (collada.geometry.Geometry method), 23
createPolylist() (collada.geometry.Geometry method), 23
createTriangleSet() (collada.geometry.Geometry method), 23

D

DaeBrokenRefError, 96
DaeError, 96
DaeIncompleteError, 96
DaeMalformedError, 96
DaeObject (class in collada.common), 17
DaeSaveValidationError, 96
DaeUnsupportedError, 96
data (collada.material.CImage attribute), 56
data (collada.source.FloatSource attribute), 62
data (collada.source.IDRefSource attribute), 65
data (collada.source.NameSource attribute), 63
diffuse (collada.material.Effect attribute), 55
direction (collada.camera.BoundOrthographicCamera attribute), 89
direction (collada.camera.BoundPerspectiveCamera attribute), 88
direction (collada.light.BoundDirectionalLight attribute), 93
direction (collada.light.BoundSpotLight attribute), 95
DirectionalLight (class in collada.light), 45
double_sided (collada.geometry.Geometry attribute), 22
double_sided (collada.material.Effect attribute), 55

E

Effect (class in collada.material), 53
effect (collada.material.Material attribute), 53
effects (collada.Collada attribute), 16
emission (collada.material.Effect attribute), 55
errors (collada.Collada attribute), 16
ExtraNode (class in collada.scene), 73
eye (collada.scene.LookAtTransform attribute), 77

F

falloff_ang (collada.light.SpotLight attribute), 52
falloff_exp (collada.light.SpotLight attribute), 52
floatarray (collada.material.CImage attribute), 56
FloatSource (class in collada.source), 62
format (collada.material.Surface attribute), 57

G

generateNormals() (collada.triangleset.BoundTriangleSet method), 82
generateNormals() (collada.triangleset.TriangleSet method), 27
generateTexTangentsAndBinormals() (collada.triangleset.TriangleSet method), 27
geometries (collada.Collada attribute), 16
Geometry (class in collada.geometry), 21
geometry (collada.scene.GeometryNode attribute), 69
GeometryNode (class in collada.scene), 69
getInputList() (collada.lineset.LineSet method), 29
getInputList() (collada.polygons.Polygons method), 33
getInputList() (collada.polylist.Polylist method), 31
getInputList() (collada.primitive.Primitive method), 25
getInputList() (collada.triangleset.TriangleSet method), 27
getList() (collada.source.InputList method), 61

I

id (collada.camera.OrthographicCamera attribute), 43
id (collada.camera.PerspectiveCamera attribute), 41
id (collada.geometry.Geometry attribute), 22
id (collada.light.AmbientLight attribute), 48
id (collada.light.DirectionalLight attribute), 46
id (collada.light.PointLight attribute), 49
id (collada.light.SpotLight attribute), 52
id (collada.material.CImage attribute), 56
id (collada.material.Effect attribute), 54
id (collada.material.Material attribute), 53
id (collada.material.Sampler2D attribute), 58
id (collada.material.Surface attribute), 57
id (collada.scene.Node attribute), 68
id (collada.scene.Scene attribute), 66
id (collada.source.FloatSource attribute), 62
id (collada.source.IDRefSource attribute), 65
id (collada.source.NameSource attribute), 63
IDRefSource (class in collada.source), 64
ignoreErrors() (collada.Collada method), 17
image (collada.material.Surface attribute), 57
images (collada.Collada attribute), 16
index() (collada.util.IndexedList method), 99
index_of_refraction (collada.material.Effect attribute), 55
IndexedList (class in collada.util), 98
indices (collada.polylist.Polygon attribute), 35
indices (collada.triangleset.Triangle attribute), 35
InputList (class in collada.source), 60
inputs (collada.scene.MaterialNode attribute), 71
interest (collada.scene.LookAtTransform attribute), 77

K

keywords (collada.asset.Asset attribute), 20

L

Light (class in collada.light), 44
light (collada.scene.LightNode attribute), 72
LightNode (class in collada.scene), 71
lights (collada.Collada attribute), 16
Line (class in collada.lineset), 36

linear_att (collada.light.PointLight attribute), 50
linear_att (collada.light.SpotLight attribute), 52
lines() (collada.lineset.BoundLineSet method), 83
LineSet (class in collada.lineset), 28
load() (collada.common.DaeObject static method), 17
load() (collada.primitive.Primitive method), 25
load() (collada.scene.SceneNode method), 67
load() (collada.scene.Transform method), 73
LookAtTransform (class in collada.scene), 76

M

magfilter (collada.material.Sampler2D attribute), 58
Map (class in collada.material), 58
Material (class in collada.material), 52
material (collada.lineset.Line attribute), 36
material (collada.polylist.Polygon attribute), 35
material (collada.triangleset.Triangle attribute), 35
MaterialNode (class in collada.scene), 70
materialnodebysymbol (collada.geometry.BoundGeometry attribute), 78
materials (collada.Collada attribute), 16
materials (collada.scene.ControllerNode attribute), 70
materials (collada.scene.GeometryNode attribute), 69
matrix (collada.camera.BoundOrthographicCamera attribute), 89
matrix (collada.camera.BoundPerspectiveCamera attribute), 88
matrix (collada.geometry.BoundGeometry attribute), 78
matrix (collada.light.BoundSpotLight attribute), 95
matrix (collada.scene.LookAtTransform attribute), 77
matrix (collada.scene.MatrixTransform attribute), 76
matrix (collada.scene.Node attribute), 68
matrix (collada.scene.RotateTransform attribute), 75
matrix (collada.scene.ScaleTransform attribute), 76
matrix (collada.scene.TranslateTransform attribute), 74
MatrixTransform (class in collada.scene), 76
minfilter (collada.material.Sampler2D attribute), 58
modified (collada.asset.Asset attribute), 19
Morph (class in collada.controller), 38

N

name (collada.geometry.Geometry attribute), 22
name (collada.material.Material attribute), 53
NameSource (class in collada.source), 63
Node (class in collada.scene), 67
node (collada.scene.NodeNode attribute), 69
NodeNode (class in collada.scene), 68
nodes (collada.Collada attribute), 16
nodes (collada.scene.Scene attribute), 66
normal (collada.lineset.BoundLineSet attribute), 83
normal (collada.lineset.LineSet attribute), 29
normal (collada.polygons.BoundPolygons attribute), 86
normal (collada.polygons.Polygons attribute), 33
normal (collada.polylist.BoundPolylist attribute), 85
normal (collada.polylist.Polylist attribute), 31
normal (collada.primitive.BoundPrimitive attribute), 79
normal (collada.primitive.Primitive attribute), 25
normal (collada.triangleset.BoundTriangleSet attribute), 81
normal (collada.triangleset.TriangleSet attribute), 27
normal_index (collada.lineset.BoundLineSet attribute), 83
normal_index (collada.lineset.LineSet attribute), 29
normal_index (collada.polygons.BoundPolygons attribute), 86
normal_index (collada.polygons.Polygons attribute), 33
normal_index (collada.polylist.BoundPolylist attribute), 85
normal_index (collada.polylist.Polylist attribute), 31
normal_index (collada.primitive.BoundPrimitive attribute), 80
normal_index (collada.primitive.Primitive attribute), 25
normal_index (collada.triangleset.BoundTriangleSet attribute), 81
normal_index (collada.triangleset.TriangleSet attribute), 27
normal_indices (collada.polylist.Polygon attribute), 36
normal_indices (collada.triangleset.Triangle attribute), 35
normalize_v3() (collada.util method), 98
normals (collada.lineset.Line attribute), 36
normals (collada.polylist.Polygon attribute), 35
normals (collada.triangleset.Triangle attribute), 35

O

objects() (collada.scene.CameraNode method), 72
objects() (collada.scene.ControllerNode method), 70
objects() (collada.scene.GeometryNode method), 69
objects() (collada.scene.LightNode method), 72
objects() (collada.scene.Node method), 68
objects() (collada.scene.Scene method), 66
objects() (collada.scene.SceneNode method), 67
OPAQUE_MODE (class in collada.material), 59
opaque_mode (collada.material.Effect attribute), 55
original (collada.camera.BoundOrthographicCamera attribute), 90
original (collada.camera.BoundPerspectiveCamera attribute), 89
original (collada.geometry.BoundGeometry attribute), 78
original (collada.light.BoundAmbientLight attribute), 92
original (collada.light.BoundDirectionalLight attribute), 93
original (collada.light.BoundPointLight attribute), 94
original (collada.light.BoundSpotLight attribute), 95
OrthographicCamera (class in collada.camera), 41

P

params (collada.material.Effect attribute), 54

path (collada.material.CImage attribute), 56
PerspectiveCamera (class in collada.camera), 40
pilimage (collada.material.CImage attribute), 56
PointLight (class in collada.light), 48
Polygon (class in collada.polylist), 35
Polygons (class in collada.polygons), 32
polygons() (collada.polygons.BoundPolygons method), 86
polygons() (collada.polylist.BoundPolylist method), 85
Polylist (class in collada.polylist), 30
position (collada.camera.BoundOrthographicCamera attribute), 89
position (collada.camera.BoundPerspectiveCamera attribute), 88
position (collada.light.BoundPointLight attribute), 94
position (collada.light.BoundSpotLight attribute), 95
Primitive (class in collada.primitive), 24
primitives (collada.geometry.Geometry attribute), 22
primitives() (collada.geometry.BoundGeometry method), 78

Q

quad_att (collada.light.PointLight attribute), 50
quad_att (collada.light.SpotLight attribute), 52

R

reflective (collada.material.Effect attribute), 55
reflectivity (collada.material.Effect attribute), 55
reverse() (collada.util.IndexedList method), 99
revision (collada.asset.Asset attribute), 20
RGB_ZERO (collada.material.OPAQUE_MODE attribute), 59
RotateTransform (class in collada.scene), 74

S

sampler (collada.material.Map attribute), 59
Sampler2D (class in collada.material), 57
save() (collada.asset.Asset method), 20
save() (collada.asset.Contributor method), 19
save() (collada.camera.Camera method), 40
save() (collada.camera.OrthographicCamera method), 43
save() (collada.camera.PerspectiveCamera method), 41
save() (collada.Collada method), 17
save() (collada.common.DaeObject method), 17
save() (collada.controller.Controller method), 37
save() (collada.controller.Skin method), 38
save() (collada.geometry.Geometry method), 23
save() (collada.light.AmbientLight method), 48
save() (collada.light.DirectionalLight method), 46
save() (collada.light.Light method), 45
save() (collada.light.PointLight method), 50
save() (collada.light.SpotLight method), 52
save() (collada.material.CImage method), 56
save() (collada.material.Effect method), 55
save() (collada.material.Map method), 59
save() (collada.material.Material method), 53
save() (collada.material.Sampler2D method), 58
save() (collada.material.Surface method), 57
save() (collada.scene.CameraNode method), 72
save() (collada.scene.ControllerNode method), 70
save() (collada.scene.GeometryNode method), 70
save() (collada.scene.LightNode method), 72
save() (collada.scene.MaterialNode method), 71
save() (collada.scene.Node method), 68
save() (collada.scene.NodeNode method), 69
save() (collada.scene.Scene method), 66
save() (collada.scene.SceneNode method), 67
save() (collada.source.FloatSource method), 63
save() (collada.source.IDRefSource method), 65
save() (collada.source.NameSource method), 64
save() (collada.source.Source method), 61
ScaleTransform (class in collada.scene), 75
Scene (class in collada.scene), 65
scene (collada.Collada attribute), 16
SceneNode (class in collada.scene), 66
scenes (collada.Collada attribute), 16
shaders (collada.material.Effect attribute), 54
shadingtype (collada.material.Effect attribute), 54
shapes() (collada.lineset.BoundLineSet method), 83
shapes() (collada.polygons.BoundPolygons method), 87
shapes() (collada.polylist.BoundPolylist method), 85
shapes() (collada.primitive.BoundPrimitive method), 79
shapes() (collada.triangleset.BoundTriangleSet method), 82
shininess (collada.material.Effect attribute), 55
Skin (class in collada.controller), 37
sort() (collada.util.IndexedList method), 99
Source (class in collada.source), 61
source_data (collada.asset.Contributor attribute), 18
source_geometry (collada.controller.Morph attribute), 38
sourceById (collada.geometry.Geometry attribute), 22
specular (collada.material.Effect attribute), 55
SpotLight (class in collada.light), 50
subject (collada.asset.Asset attribute), 20
supported (collada.material.Effect attribute), 54
Surface (class in collada.material), 57
surface (collada.material.Sampler2D attribute), 58
symbol (collada.scene.MaterialNode attribute), 71

T

target (collada.scene.MaterialNode attribute), 71
target_list (collada.controller.Morph attribute), 38
texbinormal_indexset (collada.lineset.LineSet attribute), 29
texbinormal_indexset (collada.polygons.Polygons attribute), 33
texbinormal_indexset (collada.polylist.Polylist attribute), 31

texbinormal_indexset (collada.primitive.Primitive attribute), 25
texbinormal_indexset (collada.triangleset.TriangleSet attribute), 27
texbinormalset (collada.lineset.LineSet attribute), 29
texbinormalset (collada.polygons.Polygons attribute), 33
texbinormalset (collada.polylist.Polylist attribute), 31
texbinormalset (collada.primitive.Primitive attribute), 25
texbinormalset (collada.triangleset.TriangleSet attribute), 27
texcoord (collada.material.Map attribute), 59
texcoord_indexset (collada.lineset.BoundLineSet attribute), 83
texcoord_indexset (collada.lineset.LineSet attribute), 29
texcoord_indexset (collada.polygons.BoundPolygons attribute), 87
texcoord_indexset (collada.polygons.Polygons attribute), 33
texcoord_indexset (collada.polylist.BoundPolylist attribute), 85
texcoord_indexset (collada.polylist.Polylist attribute), 31
texcoord_indexset (collada.primitive.BoundPrimitive attribute), 80
texcoord_indexset (collada.primitive.Primitive attribute), 25
texcoord_indexset (collada.triangleset.BoundTriangleSet attribute), 81
texcoord_indexset (collada.triangleset.TriangleSet attribute), 27
texcoord_indices (collada.polylist.Polygon attribute), 36
texcoord_indices (collada.triangleset.Triangle attribute), 35
texcoords (collada.lineset.Line attribute), 36
texcoords (collada.polylist.Polygon attribute), 35
texcoords (collada.triangleset.Triangle attribute), 35
texcoordset (collada.lineset.BoundLineSet attribute), 83
texcoordset (collada.lineset.LineSet attribute), 29
texcoordset (collada.polygons.BoundPolygons attribute), 87
texcoordset (collada.polygons.Polygons attribute), 34
texcoordset (collada.polylist.BoundPolylist attribute), 85
texcoordset (collada.polylist.Polylist attribute), 32
texcoordset (collada.primitive.BoundPrimitive attribute), 80
texcoordset (collada.primitive.Primitive attribute), 25
texcoordset (collada.triangleset.BoundTriangleSet attribute), 81
texcoordset (collada.triangleset.TriangleSet attribute), 27
textangent_indexset (collada.lineset.LineSet attribute), 29
textangent_indexset (collada.polygons.Polygons attribute), 34
textangent_indexset (collada.polylist.Polylist attribute), 32
textangent_indexset (collada.primitive.Primitive attribute), 25
textangent_indexset (collada.triangleset.TriangleSet attribute), 27
textangentset (collada.lineset.LineSet attribute), 30
textangentset (collada.polygons.Polygons attribute), 34
textangentset (collada.polylist.Polylist attribute), 32
textangentset (collada.primitive.Primitive attribute), 25
textangentset (collada.triangleset.TriangleSet attribute), 28
title (collada.asset.Asset attribute), 19
toUnitVec() (collada.util method), 98
Transform (class in collada.scene), 73
TranslateTransform (class in collada.scene), 74
transparency (collada.material.Effect attribute), 55
transparent (collada.material.Effect attribute), 55
Triangle (class in collada.triangleset), 34
triangles() (collada.polylist.Polygon method), 36
triangles() (collada.triangleset.BoundTriangleSet method), 81
TriangleSet (class in collada.triangleset), 26
triangleset() (collada.polygons.BoundPolygons method), 87
triangleset() (collada.polygons.Polygons method), 34
triangleset() (collada.polylist.BoundPolylist method), 85
triangleset() (collada.polylist.Polylist method), 31

U

uintarray (collada.material.CImage attribute), 56
unitmeter (collada.asset.Asset attribute), 20
unitname (collada.asset.Asset attribute), 20
up (collada.camera.BoundOrthographicCamera attribute), 90
up (collada.camera.BoundPerspectiveCamera attribute), 88
up (collada.light.BoundSpotLight attribute), 95
UP_AXIS (class in collada.asset), 20
upaxis (collada.asset.Asset attribute), 20
upvector (collada.scene.LookAtTransform attribute), 77

V

vertex (collada.lineset.BoundLineSet attribute), 83
vertex (collada.lineset.LineSet attribute), 30
vertex (collada.polygons.BoundPolygons attribute), 87
vertex (collada.polygons.Polygons attribute), 34
vertex (collada.polylist.BoundPolylist attribute), 85
vertex (collada.polylist.Polylist attribute), 32
vertex (collada.primitive.BoundPrimitive attribute), 79
vertex (collada.primitive.Primitive attribute), 25
vertex (collada.triangleset.BoundTriangleSet attribute), 81
vertex (collada.triangleset.TriangleSet attribute), 28
vertex_index (collada.lineset.BoundLineSet attribute), 83
vertex_index (collada.lineset.LineSet attribute), 30

vertex_index (collada.polygons.BoundPolygons attribute), 87
vertex_index (collada.polygons.Polygons attribute), 34
vertex_index (collada.polylist.BoundPolylist attribute), 85
vertex_index (collada.polylist.Polylist attribute), 32
vertex_index (collada.primitive.BoundPrimitive attribute), 80
vertex_index (collada.primitive.Primitive attribute), 25
vertex_index (collada.triangleset.BoundTriangleSet attribute), 81
vertex_index (collada.triangleset.TriangleSet attribute), 28
vertices (collada.lineset.Line attribute), 36
vertices (collada.polylist.Polygon attribute), 35
vertices (collada.triangleset.Triangle attribute), 35

W

write() (collada.Collada method), 17

X

x (collada.scene.RotateTransform attribute), 75
x (collada.scene.ScaleTransform attribute), 75
x (collada.scene.TranslateTransform attribute), 74
X_UP (collada.asset.UP_AXIS attribute), 20
xfov (collada.camera.BoundPerspectiveCamera attribute), 88
xfov (collada.camera.PerspectiveCamera attribute), 41
xmag (collada.camera.BoundOrthographicCamera attribute), 89
xmag (collada.camera.OrthographicCamera attribute), 43
xmlnode (collada.asset.Asset attribute), 20
xmlnode (collada.asset.Contributor attribute), 18
xmlnode (collada.camera.OrthographicCamera attribute), 43
xmlnode (collada.camera.PerspectiveCamera attribute), 41
xmlnode (collada.Collada attribute), 16
xmlnode (collada.common.DaeObject attribute), 17
xmlnode (collada.geometry.Geometry attribute), 22
xmlnode (collada.light.AmbientLight attribute), 48
xmlnode (collada.light.DirectionalLight attribute), 46
xmlnode (collada.light.PointLight attribute), 50
xmlnode (collada.light.SpotLight attribute), 52
xmlnode (collada.lineset.LineSet attribute), 29
xmlnode (collada.material.CImage attribute), 56
xmlnode (collada.material.Effect attribute), 55
xmlnode (collada.material.Map attribute), 59
xmlnode (collada.material.Material attribute), 53
xmlnode (collada.material.Sampler2D attribute), 58
xmlnode (collada.material.Surface attribute), 57
xmlnode (collada.polylist.Polylist attribute), 31
xmlnode (collada.scene.CameraNode attribute), 72
xmlnode (collada.scene.ControllerNode attribute), 70
xmlnode (collada.scene.ExtraNode attribute), 73
xmlnode (collada.scene.GeometryNode attribute), 69
xmlnode (collada.scene.LightNode attribute), 72
xmlnode (collada.scene.LookAtTransform attribute), 77
xmlnode (collada.scene.MaterialNode attribute), 71
xmlnode (collada.scene.MatrixTransform attribute), 76
xmlnode (collada.scene.Node attribute), 68
xmlnode (collada.scene.NodeNode attribute), 69
xmlnode (collada.scene.RotateTransform attribute), 75
xmlnode (collada.scene.ScaleTransform attribute), 76
xmlnode (collada.scene.Scene attribute), 66
xmlnode (collada.scene.TranslateTransform attribute), 74
xmlnode (collada.source.FloatSource attribute), 62
xmlnode (collada.source.IDRefSource attribute), 65
xmlnode (collada.source.NameSource attribute), 64

Y

y (collada.scene.RotateTransform attribute), 75
y (collada.scene.ScaleTransform attribute), 75
y (collada.scene.TranslateTransform attribute), 74
Y_UP (collada.asset.UP_AXIS attribute), 20
yfov (collada.camera.BoundPerspectiveCamera attribute), 88
yfov (collada.camera.PerspectiveCamera attribute), 41
ymag (collada.camera.BoundOrthographicCamera attribute), 89
ymag (collada.camera.OrthographicCamera attribute), 43

Z

z (collada.scene.RotateTransform attribute), 75
z (collada.scene.ScaleTransform attribute), 76
z (collada.scene.TranslateTransform attribute), 74
Z_UP (collada.asset.UP_AXIS attribute), 20
zfar (collada.camera.BoundOrthographicCamera attribute), 89
zfar (collada.camera.BoundPerspectiveCamera attribute), 88
zfar (collada.camera.OrthographicCamera attribute), 43
zfar (collada.camera.PerspectiveCamera attribute), 41
zfar (collada.light.BoundPointLight attribute), 94
zfar (collada.light.PointLight attribute), 50
znear (collada.camera.BoundOrthographicCamera attribute), 89
znear (collada.camera.BoundPerspectiveCamera attribute), 88
znear (collada.camera.OrthographicCamera attribute), 43
znear (collada.camera.PerspectiveCamera attribute), 41