
dxfgripper Documentation

Release 0.7.5

Manfred Moitzi

September 23, 2015

1	Read DXF files	3
1.1	Options dict for reading DXF files	3
2	Helper Functions	5
3	Drawing Content	7
3.1	Layer Table	8
3.2	Layer	8
3.3	Style Table	8
3.4	Style	9
3.5	Linetype Table	9
3.6	Linetype	10
3.7	Blocks Section	10
3.8	Entity Section	10
4	Entity Types	11
4.1	Base Class Shape	11
4.2	Block	12
4.3	Line	13
4.4	Point	13
4.5	Circle	13
4.6	Arc	13
4.7	Solid	14
4.8	Trace	14
4.9	Face	14
4.10	Text	14
4.11	Attrib	16
4.12	Attdef	16
4.13	Insert	16
4.14	Polyline	16
4.15	Vertex	17
4.16	Polyface	18
4.17	Polymesh	18
4.18	LWPolyline	19
4.19	Ellipse	20
4.20	Ray	20
4.21	XLine	20
4.22	Spline	20

4.23	Helix	21
4.24	MText	22
4.25	Sun	23
4.26	Light	23
4.27	Mesh	24
4.28	Body	25
4.29	Region	25
4.30	3DSolid	25
4.31	Surface	25
4.32	PlaneSurface	25
5	Howtos	27
5.1	Open a DXF file	27
5.2	Query Header Variables	27
5.3	Query Entities	27
5.4	Query Blocks	27
5.5	Layers	28
5.6	Layouts (Modelspace or Paperspace)	28

last updated September 23, 2015.

dxfgripper is a Python library to grab information from DXF drawings - all DXF versions supported.

Python compatibility: *dxfgripper* is tested with CPython 2.7, CPython3.4 and PyPy.

License: *dxfgripper* is licensed under the MIT license.

simple usage:

```
dxfgripper = dxfgripper.readfile("drawing.dxf")
print("DXF version: {}".format(dxfgripper.dxfversion))
header_var_count = len(dxfgripper.header) # dict of dxf header vars
layer_count = len(dxfgripper.layers) # collection of layer definitions
block_definition_count = len(dxfgripper.blocks) # dict like collection of block definitions
entity_count = len(dxfgripper.entities) # list like collection of entities
```

Read DXF files

readfile (*filename* [, *options*=None])

Read DXF file *filename* from the file system, and returns an object *Drawing*. *options* is a dict with options for reading DXF files.

read (*stream* [, *options*=None])

Like *readfile* (), but reads the DXF data from a *stream*. *stream* only requires a method *readline* ()

1.1 Options dict for reading DXF files

default options:

```
DEFAULT_OPTIONS = {
    "grab_blocks": True,
    "assure_3d_coords": False,
    "resolve_text_styles": True,
}
```

key	description
grab_blocks	if True read block definitions from DXF file, else the dict <i>Drawing.blocks</i> is empty.
assure_3d_coords	guarantees (x, y, z) tuples for ALL coordinates
resolve_text_styles	if True <i>Text</i> , <i>Attrib</i> , <i>Attdef</i> and <i>MText</i> attributes will be set by the associated text style if necessary

Helper Functions

`aci_to_true_color(index)`

Returns the DXF default true color value for AutoCAD Color Index *index* as *TrueColor* object. Raises *IndexError* for *index* < 0 and *index* > 255.

Drawing Content

class Drawing

Contains all collected data from the DXF file.

Drawing.dxfversion

DXF version as *string*.

DXF	AutoCAD Version
AC1009	AutoCAD R12
AC1015	AutoCAD R2000
AC1018	AutoCAD R2004
AC1021	AutoCAD R2007
AC1024	AutoCAD R2010
AC1027	AutoCAD R2013

Drawing.encoding

content encoding, default is cp1252

Drawing.filename

filename if read from a file.

Drawing.header

Contains all the DXF header vars in a *dict* like object. For explanation of DXF header vars and their content see the DXF specifications from [Autodesk](#). Header var content are basic Python types like *string*, *int*, and *float* as simple types and *tuples of float values* for 2D- and 3D points.

Drawing.layers

Contains all layer definitions in an object of type *LayerTable*.

Drawing.styles

Contains all text style definitions in an object of type *StyleTable*.

Drawing.linetypes

Contains all linetype definitions in an object of type *LinetypeTable*.

Drawing.blocks

Contains all block definitions in a *dict* like object of type *BlocksSection*.

Drawing.entities

Contains all drawing entities in a *list* like object of type *EntitySection*.

Drawing.objects

Contains DXF objects from the objects section in a *list* like object of type *EntitySection*.

Drawing.modelspace()

Iterate over all DXF entities in *modelspace*.

Drawing.**paperspace**()
 Iterate over all DXF entities in *paperspace*.

3.1 Layer Table

class LayerTable
 Contains all layer definitions as objects of type *Layer*.

LayerTable.**get**(*name*)
 Return layer *name* as object of type *Layer*. Raises *KeyError*

LayerTable.**__getitem__**(*name*)
 Support for index operator: `dwg.layers[name]`

LayerTable.**names**(*name*)
 Returns a sorted list of all layer names.

LayerTable.**__iter__**()
 Iterate over all layers, yields *Layer* objects.

LayerTable.**__len__**()
 Returns count of layers, support for standard `len()` function.

3.2 Layer

class Layer

Layer.**name**
 Layer name as *string*

Layer.**color**
 Layer color as *int* in range 1 to 255.

Layer.**linetype**
 Layer linetype as *string*.

Layer.**locked**
 type is *bool*

Layer.**frozen**
 type is *bool*

Layer.**on**
 type is *bool*

3.3 Style Table

class StyleTable
 Contains all text style definitions as objects of type *Style*.

StyleTable.**get**(*name*)
 Return text style *name* as object of type *Style*. Raises *KeyError*

StyleTable.**__getitem__**(*name*)
 Support for index operator: `dwg.styles[name]`

`StyleTable.names(name)`

Returns a sorted list of all text style names.

`StyleTable.__iter__()`

Iterate over all text styles, yields *Style* objects.

`StyleTable.__len__()`

Returns count of text styles, support for standard `len()` function.

3.4 Style

class Style

`Style.name`

Text style name.

`Style.height`

Text fixed height as *float*, is 0 for no fixed height.

`Style.width`

Text width factor.

`Style.oblique`

Text oblique angle. (0 deg = vertical)

`Style.is_backwards`

True if text is mirrored in X.

`Style.is_upside_down`

True if text is mirrored in Y.

`Style.font`

Primary font file name

`Style.bigfont`

Bigfont file name

3.5 Linetype Table

class LinetypeTable

Contains all linetype definitions as objects of type *Linetype*.

`LinetypeTable.get(name)`

Return linetype *name* as object of type *Linetype*. Raises *KeyError*

`LinetypeTable.__getitem__(name)`

Support for index operator: `dwg.linetypes[name]`

`LinetypeTable.names(name)`

Returns a sorted list of all linetype names.

`LinetypeTable.__iter__()`

Iterate over all linetypes, yields *Linetype* objects.

`LinetypeTable.__len__()`

Returns count of linetypes, support for standard `len()` function.

3.6 Linetype

class Linetype

TODO

3.7 Blocks Section

class BlocksSection

Contains all block definitions as objects of type *Block*.

`BlocksSection.__len__()`

Returns count of blocks, support for standard `len()` function.

`BlocksSection.__iter__()`

Iterates over blocks, yields *Block* objects.

`BlocksSection.__contains__(self, name)`

Returns True if a block *name* exists, support for standard `in` operator.

`BlocksSection.__getitem__(name)`

Returns block *name*, support for the index operator: `block = dwg.blocks[name]`. Raises *KeyError*

`BlocksSection.get(name[, default=None])`

Returns block *name* if exists or *default*.

3.8 Entity Section

class EntitySection

Contains all drawing entities.

`EntitySection.__len__()`

Returns count of entities, support for standard `len()` function.

`EntitySection.__iter__()`

Iterates over all entities.

`EntitySection.__getitem__(index)`

Returns entity a location *index*, *slicing* is possible, support for the index operator `dwg.entity = entities[index]`. Raises *IndexError*

example for accessing entities:

```
dwg = dxfgripper.readfile('test.dxf')
all_layer_0_entities = [entity for entity in dwg.entities if entity.layer == '0']
```

Entity Types

4.1 Base Class Shape

class Shape

Base class for all drawing entities.

Shape.paperspace

True for *paperspace* and False for *modelspace*.

Shape.dxfname

DXF entity name, like CIRCLE or LINE

Shape.layer

Layer name as *string*

Shape.linetype

Linetype as *string* or *None*, *None* means linetype by layer.

Shape.thickness

Element thickness as *float*.

Shape.extrusion

Vector as (x, y, z) *tuple*, indicate the the entity's extrusion direction. Default = (0, 0, 1)

Shape.ltscale

Linetype scale as *float*

Shape.invisible

True if entity is invisible.

Shape.color

Entity color as ACI (AutoCAD Color Index) where 256 means color by layer and 0 means color by block.

Shape.true_color

Entity color as 0x00RRGGBB 24-bit integer value, returns a *TrueColor* object. Value is *None* if not set.

Shape.transparency

Entity transparency as float from 0.0 to 1.0, 0.0 is opaque and 1.0 is 100% transparent. Value is *None* if not set.

Shape.shadow_mode

Value	Description
0	Casts and receives shadows
1	Casts shadows
2	Receives shadows
3	Ignores shadows
None	if not set

class TrueColor (*int*)

Represents a true color value as *int*. Create new *TrueColor* objects:

```
t = TrueColor(0xAABBCC)
t = TrueColor.from_rgb(0xAA, 0xBB, 0xCC)
t = TrueColor.from_aci(1) # ACI for red (AutoCAD Color Index)
```

Unpack TrueColor:

```
r, g, b = t.rgb() # fastest way
r, g, b = t # unpacking by t.__getitem__()

red = t.r
green = t.g
blue = t.b

red = t[0]
green = t[1]
blue = t[2]
```

TrueColor.r

Red value as *int*.

TrueColor.g

Green value as *int*.

TrueColor.b

Blue value as *int*.

TrueColor.rgb()

Returns a tuple (red, green, blue) each value in range 0 to 255. (255, 255, 255) = white.

TrueColor.from_rgb (*r, g, b*)

Returns a *TrueColor* object.

TrueColor.from_aci (*index*)

Returns the DXF default true color value for AutoCAD Color Index *index* as *TrueColor* object. Raises *IndexError* for *index* < 1 and *index* > 255.

4.2 Block

class Block (*Shape*)

Block.basepoint

Base point of block definition as 2D- or 3D point of type *tuple*.

Block.name

Block name as *string*

Block.flags

Block flags as *int*, for explanation see the DXF specifications from [Autodesk](#) and see also *Block.is_...* properties.

Block.xrefpath

Path to external reference as *string*

Block.is_xref

True if block is an external reference.

Block.is_xref_overlay

True if block is an external overlay reference.

Block.is_anonymous

True if block is an anonymous block, created by hatch or dimension.

Block.__iter__:

Support for iterator protocol, iterates over all block entities.

Block.__getitem__(index):

Returns block entity at location *index*, *slicing* is supported.

Block.__len__():

Returns count of block entities, support for standard `len()` function.

4.3 Line

class Line (*Shape*)

Line.start

Start point of line (x, y[, z]) as *tuple*

Line.end

End point of line (x, y[, z]) as *tuple*

4.4 Point

class Point (*Shape*)

Point.point

Location of point (x, y[, z]) as *tuple*

4.5 Circle

class Circle (*Shape*)

Circle.center

Location of circle center point (x, y[, z]) as *tuple*

Circle.radius

Circle radius as *float*

4.6 Arc

class Arc (*Shape*)

`Arc.center`

Location of arc center point (x, y[, z]) as *tuple*

`arc.radius`

Arc radius as *float*

`arc.startangle`

Arc startangle in degrees as *float*. (full circle = 360 degrees)

`arc.endangle`

Arc endangle in degrees as *float*. (full circle = 360 degrees)

4.7 Solid

class Solid (*Shape*)

A solid filled shape with 4 points. For Triangles point 3 and point 4 has the same location.

`Solid.points`

List of points (x, y[, z]) as *tuple*.

4.8 Trace

class Trace (*Solid*)

Same as *Solid*.

4.9 Face

class Face (*Trace*)

A solid filled 3D shape with 4 points. For Triangles point 3 and point 4 has the same location. *DXF entity 3DFACE*

`Face.points`

List of points (x, y, z) as *tuple*.

`Face.is_edge_invisible` (*index*)

Returns True if edge *index* is invisible, index in [0, 1, 2, 3].

4.10 Text

The attributes *height*, *width*, *oblique*, *is_backwards* and *is_upside_down* are defined in the associated *Style* object, if the value of these attributes are 0 (*height*, *width*) or None (*oblique*, *is_backwards*, *is_upside_down*).

If the import option "resolve_text_styles" is *True*, all the above mentioned attributes and *font* and *bigfont* already have the 'final' value, no need to look into the *Style* object.

class Text (*Shape*)

`Text.insert`

Location of text (x, y, z) as *tuple*.

`Text.text`

Text content as *string*.

`Text.height`

Text height as *float*, if 0 you have to look into the styles table *Drawing.styles* with *Text.style* as key.

`Text.width`

Text width factor.

`Text.oblique`

Text oblique angle. (0 deg = vertical)

`Text.rotation`

Rotation angle in degrees as *float*. (full circle = 360 degrees)

`Text.style`

Text style name as *string*

`Text.halign`

Horizontal alignment as *int*.

Value	Alignment
0	Left
1	Center
2	Right
3	Aligned (if vertical alignment = 0)
4	Middle (if vertical alignment = 0)
5	Fit (if vertical alignment = 0)

`Text.valign`

Vertical alignment as *int*.

Value	Alignment
0	Baseline
1	Bottom
2	Middle
3	Top

`Text.is_backwards`

True if text is mirrored in X.

`Text.is_upside_down`

True if text is mirrored in Y.

`Text.alignpoint`

Second alignment point as tuple or *None*.

`Text.font`

Font name as string, if import option *resolve_text_styles* is *True* else "".

`Text.bigfont`

Bigfont name as string, if import option *resolve_text_styles* is *True* else "".

`Text.plain_text()`

Get text content without formatting codes like %%.u.

4.11 Attrib

class Attrib (*Text*)

A text entity, in usual cases attached to a block reference entity *Insert*, inherits from *Text*.

Attrib.tag

The attribute tag as *string*.

4.12 Attdef

Same as *Attrib*, but located in a block definition entity *Block*.

4.13 Insert

class Insert (*Shape*)

Insert.name

Name of block definition as *string*.

Insert.insert

Location of block reference (x, y, z) as *tuple*.

Insert.rotation

Rotation angle in degrees as *float*. (full circle = 360 degrees)

Insert.scale

(x, y, z) block scaling as *tuple*, default is (1.0, 1.0, 1.0)

Insert.row_count

Row count for multiple block references.

Insert.col_count

Column count for multiple block references.

Insert.row_spacing

Row distance for multiple block references.

Insert.col_spacing

col distance for multiple block references.

Insert.attrs

List of *Attrib* entities attached to the *Insert* entity.

Insert.find_attr(tag) :

Get *Attrib* entity by *tag*, returns *None* if not found.

4.14 Polyline

class Polyline (*Shape*)

Multiple 2D- or 3D vertices connected by lines. The DXF entity *POLYLINE* is also used to define *Polyfaces* and *Polymeshes*, dxfgripper defines separated classes for this entities see: *Polyface* and *Polymesh*.

Polyline.is_closed

True if polyline is closed.

Polyline.mode

Returns the polyline mode: `polyline2d`, `polyline3d` or `spline2d`.

Polyline.spline_type

If polyline is a 2D spline: `quadratic_bspline`, `cubic_bspline`, `bezier_curve` else `None`.

Polyline.default_start_width

Default line segment start width, if not set in vertex entity.

Polyline.default_end_width

Default line segment end width, if not set in vertex entity.

Polyline.points

List of all vertex locations as (x, y[, z]) *tuple*. If this polyline is a 2d spline these points are just the fit points.

Polyline.controlpoints

List of all control points as (x, y[, z]) *tuple*, if this polyline is a 2d spline.

Polyline.tangents

List of all vertex tangent angles as *float* in degrees or `None` if not defined. (Just for fit points)

Polyline.width

List of all vertex width values as (start_width, end_width) *tuple*. Just for fit points if this polyline is a 2D spline.

Polyline.bulge

List of all vertex bulge values as *floats*.

Polyline.__getitem__ (index)

Returns vertex *index* as *Vertex* entity. support for standard operator `vertex = polyline[index]`.
Raises *IndexError*

Polyline.__len__ ()

Returns count of vertices.

Polyline.__iter__ ()

Iterate of all vertices, as *Vertex* entity.

4.15 Vertex

class Vertex (Shape)

Vertex.location

Location as (x, y, z)-tuple.

Vertex.start_width

Vertex.end_width

Vertex.bulge

The bulge is the tangent of one fourth the included angle for an arc segment, made negative if the arc goes clockwise from the start point to the endpoint. A bulge of 0 indicates a straight segment, and a bulge of 1 is a semicircle. If you have questions ask *Autodesk*.

Vertex.tangent

Curve fitting tangent in degrees as *float* or `None`. (full circle = 360 degrees)

4.16 Polyface

class Polyface (*Shape*)

Dxftype is *POLYFACE*, which is a *POLYLINE* DXF entity.

Polyface.vertices

List of all *Polyface* vertices a *Vertex* object.

Polyface.__getitem__ (*index*)

Returns face *index* as *SubFace* object. support for standard operator `face = polyface[index]`. Raises *IndexError*

Polyface.__len__ ()

Returns count of faces.

Polyface.__iter__ ()

Iterate of all faces, as *SubFace* objects.

Polyface.smooth_type

Smooth surface type; integer codes, not bit-coded:

Value	Description
0	No smooth surface fitted
5	Quadratic B-spline surface
6	Cubic B-spline surface
8	Bezier surface

4.16.1 SubFace

class SubFace

A SubFace describes a single face of a *Polyface*.

SubFace.face_record

Face record vertex, the basic DXF structure of faces, where you can get the DXF attributes of the face like color or linetype: `subface.face_record.color`

SubFace.__len__ ()

Returns count of vertices 3 or 4.

SubFace.__getitem__ (*pos*) :

Returns vertex at index *pos* as *Vertex* object

SubFace.__iter__ () :

Returns a list of the face vertices as (x, y, z)-tuples.

SubFace.indices () :

Returns a list of vertex indices, get vertex by index from `Polyface.vertices[index]`.

SubFace.is_edge_visible (*pos*) :

Returns *True* if face edge *pos* is visible else *False*.

4.17 Polymesh

class Polymesh (*Shape*)

Dxftype is *POLYMESH*, which is a *POLYLINE* DXF entity.

A *Polymesh* is a grid of m x n vertices, where every vertex has its own 3D location.

`Polymesh.mcount`

Count of vertices in m direction as *int*.

`Polymesh.ncount`

Count of vertices in n direction as *int*.

`Polymesh.is_mclosed`

True if *Polymesh* is closed in m direction.

`Polymesh.is_nclosed`

True if *Polymesh* is closed in n direction.

`Polymesh.m_smooth_density`

Smooth surface M density.

`Polymesh.n_smooth_density`

Smooth surface N density.

`Polymesh.smooth_type`

Smooth surface type; integer codes, not bit-coded:

Value	Description
0	No smooth surface fitted
5	Quadratic B-spline surface
6	Cubic B-spline surface
8	Bezier surface

`Polymesh.get_vertex(pos)`

Returns the *Vertex* at *pos*, where *pos* is a *tuple* (m, n). First vertex is (0, 0).

`Polymesh.get_location(pos)`

Returns the location (x, y, z) as *tuple* at *pos*, where *pos* is a *tuple* (m, n). First vertex is (0, 0).

4.18 LWPolyline

class LWPolyline (*Shape*)

LWPolyline is a lightweight only 2D Polyline.

`LWPolyline.points`

List of 2D polyline points as (x, y) *tuple*, or (x, y, z=0) *tuple* if option *assure_3d_points* is *True*.

`LWPolyline.width`

List of (start_width, end_width) values. To be ignored if *const_width* is not 0.

`LWPolyline.bulge`

List of bulge values as *float*

`LWPolyline.const_width`

Polyline has this constant width, if this value is not 0.

`LWPolyline.is_closed`

True if the polyline is closed.

`LWPolyline.elevation`

`LWPolyline.__len__()`

Returns the count of polyline points.

`LWPolyline.__getitem__(index)`

Returns polyline point at position *index*, *slicing* is supported. Raises *IndexError*

`LWPPolyline.__iter__()`
Iterate over all polyline points.

4.19 Ellipse

class Ellipse (*Shape*)

Ellipse.center
Location of ellipse center point (x, y[, z]) as *tuple*

Ellipse.majoraxis
End point of major axis (x, y[, z]) as *tuple*

Ellipse.ratio
Ratio of minor axis to major axis as *float*.

Ellipse.startparam
Start parameter (this value is 0.0 for a full ellipse).

Ellipse.endparam
End parameter (this value is 2pi for a full ellipse)

4.20 Ray

class Ray (*Shape*)

Ray.start
Location of the ray start point (x, y, z) as *tuple*

Ray.unitvector
Ray direction as unit vector (x, y, z) as *tuple*

4.21 XLine

class XLine (*Ray*)

Same as [Ray](#), except a XLine (construction line) has no beginning and no end.

4.22 Spline

class Spline (*Shape*)

Spline.flags
Binary coded flags, constants stored in `dxfgripper.const`.

Spline.flags	value
SPLINE_CLOSED	1
SPLINE_PERIODIC	2
SPLINE_RATIONAL	4
SPLINE_PLANAR	8
SPLINE_LINEAR	16 (a linear spline is also a planar spline)

Spline.degree
Degree of the spline curve as *int*

Spline.starttangent
Start tangent as (x, y, z) as *tuple* or *None*

Spline.endtangent
End tangent as (x, y, z) as *tuple* or *None*

Spline.controlpoints
List of control points (x, y, z) as *tuple*

Spline.fitpoints
List of fit points (x, y, z) as *tuple*

Spline.knots
List of knot values as *float*

Spline.weights
List of weight values as *float*

Spline.normalvector
Normal vector if spline is planar else *None*.

Spline.is_closed

Spline.is_periodic

Spline.is_rational

Spline.is_planar

Spline.is_linear

4.23 Helix

3D spiral; Helix is also a *Spline*.

class Helix (*Spline*)

Helix.helix_version
Tuple (main version, maintainance version)

Helix.axis_base_point
Helix axis base point as (x, y, z) as *tuple*.

Helix.start_point
Helix start point as (x, y, z) as *tuple*.

Helix.axis_vector
Helix axis vector as (x, y, z) as *tuple*.

Helix.radius

Helix.turns
Count of turns.

Helix.turn_height
Height of one turn.

Helix.handedness
0 = left; 1 = right;

`Helix.constrain`

0 = Constrain turn height; 1 = Constrain turns; 2 = Constrain height

4.24 MText

The *height* attribute is defined in the associated *Style* object, if the value of *height* is 0.

If the import option "resolve_text_styles" is *True*, *height*, *font* and *bigfont* already have the 'final' value, no need to look into the *Style* object.

class MText (*Shape*)

Multi line text entity.

`MText.insert`

Location of text (x, y, z) as *tuple*.

`MText.rawtext`

Whole text content as one *string*.

`MText.height`

Text height as *float*

`MText.rect_width`

Reference rectangle width as *float* in drawing units.

`MText.horizontal_width`

Horizontal width of the characters that make up the *MText* entity. This value will always be equal to or less than the *MText.rect_width* value. In drawing units as *float*.

`MText.vertical_height`

Vertical height of the *MText* entity in drawing units as *float*.

`MText.linespacing`

Text line spacing as *float*, valid from 0.25 to 4.00.

`MText.attachmentpoint`

Text attachment point as *int*.

Value	Description
1	Top left
2	Top center
3	Top right
4	Middle left
5	Middle center
6	Middle right
7	Bottom left
8	Bottom center
9	Bottom right

`MText.style`

Text style name as *string*.

`MText.xdirection`

X-Axis direction vector as (x, y, z) as *tuple*. (unit vector)

`MText.font`

Font name as string, if import option "resolve_text_styles" is *True* else "".

`MText.bigfont`

Bigfont name as string, if import option "resolve_text_styles" is *True* else "".

`MText.lines()`

Returns a *list* of lines. It is the `MText.rawtext` splitted into lines by the `\P` character.

`MText.plain_text(split=False)`

Tries to remove format codes, returns a single string if *split* is *False* else multiple lines as list of strings without `\n`.

4.25 Sun

class Sun (*Entity*)

Sun representation. SUN is not a graphical object and resides in the objects section *Drawing.objects*.

`Sun.version`

`Sun.status`

Boolean value: on/off

`Sun.sun_color`

Light color as ACI color index 1 - 255; 256 = BYLAYER; *None* if unset

`Sun.intensity`

`Sun.shadows`

Boolean value

`Sun.date`

A Python standard `datetime.datetime` object.

`Sun.daylight_savings_time`

Boolean value

`Sun.shadow_type`

0 = Ray traced shadows; 1 = Shadow maps

`Sun.shadow_map_size`

`Sun.shadow_softness`

4.26 Light

class Light (*Shape*)

Defines a light source.

`Light.version`

`Light.name`

`Light.light_type`

distant = 1; point = 2; spot = 3

`Light.status`

Boolean value: on/off?

`Light.light_color`

Light color as ACI color index 1 - 255; 256 = BYLAYER; *None* if unset

`Light.true_color`

Light color as 24-bit RGB color 0x00RRGGBB, *None* if unset

`Light.plot_glyph`

Boolean value

`Light.intensity`

`Light.position`

3D position of the light source as (x, y, z) tuple.

`Light.target`

3D target location of the light, determines the light direction as (x, y, z) tuple.

`Light.attenuation_type`

0 = None; 1 = Inverse Linear; 2 = Inverse Square

`Light.use_attenuation_limits`

Boolean value

`Light.attenuation_start_limit`

`Light.attenuation_end_limit`

`Light.hotspot_angle`

`Light.fall_off_angle`

`Light.cast_shadows`

Boolean value

`Light.shadow_type`

0 = Ray traced shadows; 1 = Shadow maps

`Light.shadow_map_size`

`Light.shadow_softness`

4.27 Mesh

class Mesh (*Shape*)

3D mesh entity similar to the *Polyface* entity.

`Mesh.version`

`Mesh.blend_crease`

Boolean value (on/off)

`Mesh.subdivision_levels`

`Mesh.vertices`

List of 3D vertices (x, y, z).

`Mesh.faces`

List of mesh faces as tuples of vertex indices (v1, v2, v3, ...). Indices are 0-based and can be used with the mesh.vertex list:

```
first_face = mesh.faces[0]
first_vertex = mesh.vertices[first_face[0]]
```

`Mesh.edges`

List of mesh edges as 2-tuple of vertex indices (v1, v2). Indices are 0-based and can be used with the mesh.vertex list:

```
first_edge = mesh.edges[0]
first_vertex = mesh.vertices[first_edge[0]]
```

Mesh.edge_crease_list

List of float values, one for each edge.

Mesh.get_face (*index*)

Returns a tuple of 3D points ((x1, y1, z1), (x2, y2, z2), ...) for face at position *index*.

Mesh.get_edge (*index*)

Returns a 2-tuple of 3D points ((x1, y1, z1), (x2, y2, z2)) for edge at position *index*.

4.28 Body

class Body (*Shape*)

ACIS based 3D solid geometry.

Body.acis

SAT (Standard ACIS Text) data as list of strings. AutoCAD stores the ACIS data since DXF version AC1027 (R21013) as SAB (Standard ACIS Binary) data in the undocumented (2014-05-06) section ACDSDATA and *acis* is a binary string.

Body.is_sat

Is *True* if data is stored as SAT, no guarantee for presence of data, but *acis* is a list of strings for sure.

Body.is_sab

Is *True* if data is stored as SAB and *acis* is a binary string.

4.29 Region

class Region (*Body*)

ACIS based 2D enclosed areas.

4.30 3DSolid

class 3DSolid (*Body*)

ACIS based 3D solid geometry.

4.31 Surface

class Surface (*Body*)

ACIS based 3D freeform surfaces.

4.32 PlaneSurface

class PlaneSurface (*Surface*)

ACIS based 3D plane surfaces.

5.1 Open a DXF file

Open files from file system:

```
dwg = readfile("myfile.dxf")
```

To read file from a stream use: `read()`

5.2 Query Header Variables

The HEADER section of a DXF file contains the settings of variables associated with the drawing.

Example:

```
dxgversion = dwg.header['$ACADVER']
```

For available HEADER variables and their meaning see: [DXF Reference](#)

5.3 Query Entities

All entities of the DXF drawing, independent from *modelspace* or *paperspace*, resides in the `Drawing.entities` attribute and is an `EntitySection` object. Iterate over all entities with the `in` operator:

```
all_lines = [entity for entity in dwg.entities if entity.dxftype == 'LINE']  
all_entities_at_layer_0 = [entity for entity in dwg.entities if entity.layer == '0']
```

5.4 Query Blocks

Block references are just DXF entities called INSERT.

Get all block references for block `TestBlock`:

```
references = [entity for entity in dwg.entities if entity.dxftype == 'INSERT' and entity.name == 'TestBlock']
```

See available attributes for the `Insert` entity.

To examine the Block content, get the block definition from the blocks section:

```
test_block = dwg.blocks['TestBlock']
```

and use the `in` operator (Iterator protocol):

```
circles_in_block = [entity for entity in test_block if entity.dxftype == 'CIRCLE']
```

5.5 Layers

Layers are nothing special, they are just another attribute of the DXF entity, *dxfgripper* stores the layer as a simple *string*. The DXF entity can inherit some attributes from the layer: *color*, *linetype*

To get the real value of an attribute value == *BYLAYER*, get the layer definition:

```
layer = dwg.layers[dxfgripper.entity.layer]
color = layer.color if dxfgripper.entity.color == dxfgripper.BYLAYER else dxfgripper.entity.color
linetype = layer.linetype if dxfgripper.entity.linetype is None else dxfgripper.entity.linetype
```

Layers can be *locked* (if True else *unlocked*), *on* (if True else *off*) or *frozen* (if True else *thawed*).

5.6 Layouts (Modelspace or Paperspace)

dxfgripper just supports the *paperspace* attribute, it is not possible to examine in which layout a paperspace object resides (DXF12 has only one paperspace).

Get all *modelspace* entities:

```
modelspace_entities = [entity for entity in dwg.entities if not entity.paperspace]
```

shortcuts since 0.5.1:

```
modelspace_entities = list(dwg.modelspace())
paperspace_entities = list(dwg.paperspace())
```


Symbols

__contains__() (BlocksSection method), 10
 __getitem__() (BlocksSection method), 10
 __getitem__() (EntitySection method), 10
 __getitem__() (LWPolyline method), 19
 __getitem__() (LayerTable method), 8
 __getitem__() (LinetypeTable method), 9
 __getitem__() (Polyface method), 18
 __getitem__() (Polyline method), 17
 __getitem__() (StyleTable method), 8
 __iter__() (BlocksSection method), 10
 __iter__() (EntitySection method), 10
 __iter__() (LWPolyline method), 19
 __iter__() (LayerTable method), 8
 __iter__() (LinetypeTable method), 9
 __iter__() (Polyface method), 18
 __iter__() (Polyline method), 17
 __iter__() (StyleTable method), 9
 __len__() (BlocksSection method), 10
 __len__() (EntitySection method), 10
 __len__() (LWPolyline method), 19
 __len__() (LayerTable method), 8
 __len__() (LinetypeTable method), 9
 __len__() (Polyface method), 18
 __len__() (Polyline method), 17
 __len__() (StyleTable method), 9
 __len__() (SubFace method), 18
 3DSolid (built-in class), 25

A

aci_to_true_color(), 5
 acis (Body attribute), 25
 alignpoint (Text attribute), 15
 Arc (built-in class), 13
 attachmentpoint (MText attribute), 22
 attenuation_end_limit (Light attribute), 24
 attenuation_start_limit (Light attribute), 24
 attenuation_type (Light attribute), 24
 Attrib (built-in class), 16
 attribs (Insert attribute), 16

axis_base_point (Helix attribute), 21
 axis_vector (Helix attribute), 21

B

b (TrueColor attribute), 12
 basepoint (Block attribute), 12
 bigfont (MText attribute), 22
 bigfont (Style attribute), 9
 bigfont (Text attribute), 15
 blend_crease (Mesh attribute), 24
 Block (built-in class), 12
 blocks (Drawing attribute), 7
 BlocksSection (built-in class), 10
 Body (built-in class), 25
 bulge (LWPolyline attribute), 19
 bulge (Polyline attribute), 17
 bulge (Vertex attribute), 17

C

cast_shadows (Light attribute), 24
 center (Arc attribute), 13
 center (Circle attribute), 13
 center (Ellipse attribute), 20
 Circle (built-in class), 13
 col_count (Insert attribute), 16
 col_spacing (Insert attribute), 16
 color (Layer attribute), 8
 color (Shape attribute), 11
 const_width (LWPolyline attribute), 19
 constrain (Helix attribute), 21
 controlpoints (Polyline attribute), 17
 controlpoints (Spline attribute), 21

D

date (Sun attribute), 23
 daylight_savings_time (Sun attribute), 23
 default_end_width (Polyline attribute), 17
 default_start_width (Polyline attribute), 17
 degree (Spline attribute), 20
 Drawing (built-in class), 7

dxftype (Shape attribute), 11
 dxfversion (Drawing attribute), 7

E

edge_crease_list (Mesh attribute), 25
 edges (Mesh attribute), 24
 elevation (LWPolyline attribute), 19
 Ellipse (built-in class), 20
 encoding (Drawing attribute), 7
 end (Line attribute), 13
 end_width (Vertex attribute), 17
 endangle (arc attribute), 14
 endparam (Ellipse attribute), 20
 endtangent (Spline attribute), 21
 entities (Drawing attribute), 7
 EntitySection (built-in class), 10
 extrusion (Shape attribute), 11

F

Face (built-in class), 14
 face_record (SubFace attribute), 18
 faces (Mesh attribute), 24
 fall_off_angle (Light attribute), 24
 filename (Drawing attribute), 7
 fitpoints (Spline attribute), 21
 flags (Block attribute), 12
 flags (Spline attribute), 20
 font (MText attribute), 22
 font (Style attribute), 9
 font (Text attribute), 15
 from_aci() (TrueColor method), 12
 from_rgb() (TrueColor method), 12
 frozen (Layer attribute), 8

G

g (TrueColor attribute), 12
 get() (BlocksSection method), 10
 get() (LayerTable method), 8
 get() (LinetypeTable method), 9
 get() (StyleTable method), 8
 get_edge() (Mesh method), 25
 get_face() (Mesh method), 25
 get_location() (Polymesh method), 19
 get_vertex() (Polymesh method), 19

H

halign (Text attribute), 15
 handedness (Helix attribute), 21
 header (Drawing attribute), 7
 height (MText attribute), 22
 height (Style attribute), 9
 height (Text attribute), 15
 Helix (built-in class), 21

helix_version (Helix attribute), 21
 horizontal_width (MText attribute), 22
 hotspot_angle (Light attribute), 24

I

Insert (built-in class), 16
 insert (Insert attribute), 16
 insert (MText attribute), 22
 insert (Text attribute), 14
 intensity (Light attribute), 24
 intensity (Sun attribute), 23
 invisible (Shape attribute), 11
 is_anonymous (Block attribute), 13
 is_backwards (Style attribute), 9
 is_backwards (Text attribute), 15
 is_closed (LWPolyline attribute), 19
 is_closed (Polyline attribute), 16
 is_closed (Spline attribute), 21
 is_edge_invisible() (Face method), 14
 is_linear (Spline attribute), 21
 is_mclosed (Polymesh attribute), 19
 is_nclosed (Polymesh attribute), 19
 is_periodic (Spline attribute), 21
 is_planar (Spline attribute), 21
 is_rational (Spline attribute), 21
 is_sab (Body attribute), 25
 is_sat (Body attribute), 25
 is_upside_down (Style attribute), 9
 is_upside_down (Text attribute), 15
 is_xref (Block attribute), 13
 is_xref_overlay (Block attribute), 13

K

knots (Spline attribute), 21

L

Layer (built-in class), 8
 layer (Shape attribute), 11
 layers (Drawing attribute), 7
 LayerTable (built-in class), 8
 Light (built-in class), 23
 light_color (Light attribute), 23
 light_type (Light attribute), 23
 Line (built-in class), 13
 lines() (MText method), 23
 linespacing (MText attribute), 22
 Linetype (built-in class), 10
 linetype (Layer attribute), 8
 linetype (Shape attribute), 11
 linetypes (Drawing attribute), 7
 LinetypeTable (built-in class), 9
 location (Vertex attribute), 17
 locked (Layer attribute), 8
 ltscale (Shape attribute), 11

LWPPolyline (built-in class), 19

M

m_smooth_density (Polymesh attribute), 19
 majoraxis (Ellipse attribute), 20
 mcount (Polymesh attribute), 18
 Mesh (built-in class), 24
 mode (Polyline attribute), 16
 modelspace() (Drawing method), 7
 MText (built-in class), 22

N

n_smooth_density (Polymesh attribute), 19
 name (Block attribute), 12
 name (Insert attribute), 16
 name (Layer attribute), 8
 name (Light attribute), 23
 name (Style attribute), 9
 names() (LayerTable method), 8
 names() (LinetypeTable method), 9
 names() (StyleTable method), 8
 ncount (Polymesh attribute), 19
 normalvector (Spline attribute), 21

O

objects (Drawing attribute), 7
 oblique (Style attribute), 9
 oblique (Text attribute), 15
 on (Layer attribute), 8

P

paperspace (Shape attribute), 11
 paperspace() (Drawing method), 7
 plain_text() (MText method), 23
 plain_text() (Text method), 15
 PlaneSurface (built-in class), 25
 plot_glyph (Light attribute), 23
 Point (built-in class), 13
 point (Point attribute), 13
 points (Face attribute), 14
 points (LWPPolyline attribute), 19
 points (Polyline attribute), 17
 points (Solid attribute), 14
 Polyface (built-in class), 18
 Polyline (built-in class), 16
 Polymesh (built-in class), 18
 position (Light attribute), 24

R

r (TrueColor attribute), 12
 radius (arc attribute), 14
 radius (Circle attribute), 13
 radius (Helix attribute), 21

ratio (Ellipse attribute), 20
 rawtext (MText attribute), 22
 Ray (built-in class), 20
 read() (built-in function), 3
 readfile() (built-in function), 3
 rect_width (MText attribute), 22
 Region (built-in class), 25
 rgb() (TrueColor method), 12
 rotation (Insert attribute), 16
 rotation (Text attribute), 15
 row_count (Insert attribute), 16
 row_spacing (Insert attribute), 16

S

scale (Insert attribute), 16
 shadow_map_size (Light attribute), 24
 shadow_map_size (Sun attribute), 23
 shadow_mode (Shape attribute), 11
 shadow_softness (Light attribute), 24
 shadow_softness (Sun attribute), 23
 shadow_type (Light attribute), 24
 shadow_type (Sun attribute), 23
 shadows (Sun attribute), 23
 Shape (built-in class), 11
 smooth_type (Polyface attribute), 18
 smooth_type (Polymesh attribute), 19
 Solid (built-in class), 14
 Spline (built-in class), 20
 spline_type (Polyline attribute), 17
 start (Line attribute), 13
 start (Ray attribute), 20
 start_point (Helix attribute), 21
 start_width (Vertex attribute), 17
 startangle (arc attribute), 14
 startparam (Ellipse attribute), 20
 starttangent (Spline attribute), 21
 status (Light attribute), 23
 status (Sun attribute), 23
 Style (built-in class), 9
 style (MText attribute), 22
 style (Text attribute), 15
 styles (Drawing attribute), 7
 StyleTable (built-in class), 8
 subdivision_levels (Mesh attribute), 24
 SubFace (built-in class), 18
 Sun (built-in class), 23
 sun_color (Sun attribute), 23
 Surface (built-in class), 25

T

tag (Attrib attribute), 16
 tangent (Vertex attribute), 17
 tangents (Polyline attribute), 17
 target (Light attribute), 24

Text (built-in class), [14](#)
text (Text attribute), [14](#)
thickness (Shape attribute), [11](#)
Trace (built-in class), [14](#)
transparency (Shape attribute), [11](#)
true_color (Light attribute), [23](#)
true_color (Shape attribute), [11](#)
TrueColor (built-in class), [12](#)
turn_height (Helix attribute), [21](#)
turns (Helix attribute), [21](#)

U

unitvector (Ray attribute), [20](#)
use_attenuation_limits (Light attribute), [24](#)

V

valign (Text attribute), [15](#)
version (Light attribute), [23](#)
version (Mesh attribute), [24](#)
version (Sun attribute), [23](#)
Vertex (built-in class), [17](#)
vertical_height (MText attribute), [22](#)
vertices (Mesh attribute), [24](#)
vertices (Polyface attribute), [18](#)

W

weights (Spline attribute), [21](#)
width (LWPolyline attribute), [19](#)
width (Polyline attribute), [17](#)
width (Style attribute), [9](#)
width (Text attribute), [15](#)

X

xdirection (MText attribute), [22](#)
XLine (built-in class), [20](#)
xrefpath (Block attribute), [12](#)