dxfgrabber Documentation

Release 1.0.0

Manfred Moitzi

Jan 01, 2018
## Contents

1 Development Status 3

2 Read DXF files 5
  2.1 Options dict for reading DXF files 5

3 Helper Functions 7

4 Drawing Content 9
  4.1 Layer Table 10
  4.2 Layer 10
  4.3 Style Table 11
  4.4 Style 11
  4.5 Linetype Table 11
  4.6 Linetype 12
  4.7 Blocks Section 12
  4.8 Entity Section 12

5 Entity Types 13
  5.1 Base Class Shape 13
  5.2 Block 14
  5.3 Line 15
  5.4 Point 15
  5.5 Circle 15
  5.6 Arc 16
  5.7 Solid 16
  5.8 Trace 16
  5.9 Face 16
  5.10 Text 16
  5.11 Attrib 18
  5.12 Atdef 18
  5.13 Insert 18
  5.14 Polyline 18
  5.15 Vertex 19
  5.16 Polymesh 20
  5.17 Polyface 20
  5.18 LWPolyline 21
  5.19 Ellipse 22
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.20</td>
<td>Ray</td>
</tr>
<tr>
<td>5.21</td>
<td>XLine</td>
</tr>
<tr>
<td>5.22</td>
<td>Spline</td>
</tr>
<tr>
<td>5.23</td>
<td>Helix</td>
</tr>
<tr>
<td>5.24</td>
<td>MText</td>
</tr>
<tr>
<td>5.25</td>
<td>Sun</td>
</tr>
<tr>
<td>5.26</td>
<td>Light</td>
</tr>
<tr>
<td>5.27</td>
<td>Mesh</td>
</tr>
<tr>
<td>5.28</td>
<td>Body</td>
</tr>
<tr>
<td>5.29</td>
<td>Region</td>
</tr>
<tr>
<td>5.30</td>
<td>3DSolid</td>
</tr>
<tr>
<td>5.31</td>
<td>Surface</td>
</tr>
<tr>
<td>5.32</td>
<td>PlaneSurface</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Howtos</td>
</tr>
<tr>
<td>6.1</td>
<td>Open a DXF file</td>
</tr>
<tr>
<td>6.2</td>
<td>Query Header Variables</td>
</tr>
<tr>
<td>6.3</td>
<td>Query Entities</td>
</tr>
<tr>
<td>6.4</td>
<td>Query Blocks</td>
</tr>
<tr>
<td>6.5</td>
<td>Layers</td>
</tr>
<tr>
<td>6.6</td>
<td>Layouts (Modelspace or Paperspace)</td>
</tr>
</tbody>
</table>
last updated Jan 01, 2018.

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

Python compatibility: dxfgrabber is tested with CPython 3.6 and PyPy.

License: dxfgrabber is licensed under the MIT license.

simple usage:

```python
dxf = dxfgrabber.readfile("drawing.dxf")
print("DXF version: {}"
      .format(dxf.dxfversion))
header_var_count = len(dxf.header)  # dict of dxf header vars
layer_count = len(dxf.layers)       # collection of layer definitions
block_definition_count = len(dxf.blocks)  # dict like collection of block definitions
entity_count = len(dxf.entities)    # list like collection of entities
```
CHAPTER 1

Development Status

With version 1.0.0 dxfgrabber enters the just maintenance and bugfix mode, no further development (features) planned.
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().

2.1 Options dict for reading DXF files

default options:

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

<table>
<thead>
<tr>
<th>key</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>grab_blocks</td>
<td>if True read block definitions from DXF file, else the dict Drawing.blocks is empty.</td>
</tr>
<tr>
<td>assure_3d_coords</td>
<td>guarantees (x, y, z) tuples for ALL coordinates</td>
</tr>
<tr>
<td>resolve_text_styles</td>
<td>if True Text, Attrib, Attdf and MText attributes will be set by the associated text style</td>
</tr>
<tr>
<td>if necessary</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 3

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.
class Drawing
Contains all collected data from the DXF file.

Drawing.dxfversion
DXF version as string.

<table>
<thead>
<tr>
<th>DXF</th>
<th>AutoCAD Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC1009</td>
<td>AutoCAD R12</td>
</tr>
<tr>
<td>AC1015</td>
<td>AutoCAD R2000</td>
</tr>
<tr>
<td>AC1018</td>
<td>AutoCAD R2004</td>
</tr>
<tr>
<td>AC1021</td>
<td>AutoCAD R2007</td>
</tr>
<tr>
<td>AC1024</td>
<td>AutoCAD R2010</td>
</tr>
<tr>
<td>AC1027</td>
<td>AutoCAD R2013</td>
</tr>
<tr>
<td>AC1032</td>
<td>AutoCAD R2018</td>
</tr>
</tbody>
</table>

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.

4.1 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.

4.2 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
4.3 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.

4.4 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.big_font
Bigfont file name

4.5 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.

4.6 Linetype

class Linetype
TODO

4.7 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.

4.8 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 = dxfgrabber.readfile('test.dxf')
all_layer_0_entities = [entity for entity in dwg.entities if entity.layer == '0']
5.1 Base Class Shape

class Shape
    Base class for all drawing entities.
Shape.paperspace
    True for paperspace and False for modelspace.
Shape.dxftype
    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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Casts and receives shadows</td>
</tr>
<tr>
<td>1</td>
<td>Casts shadows</td>
</tr>
<tr>
<td>2</td>
<td>Receives shadows</td>
</tr>
<tr>
<td>3</td>
<td>Ignores shadows</td>
</tr>
<tr>
<td>None</td>
<td>if not set</td>
</tr>
</tbody>
</table>

#### class TrueColor (int)

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

```python
import dxfgrabber

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

Unpack TrueColor:

```python
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`.

## 5.2 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.<code>flags</code>
   Block flags as int, for explanation see the DXF specifications from Autodesk and see also Block.<code>is_...</code> properties.

Block.<code>xrefpath</code>
   Path to external reference as string

Block.<code>is_xref</code>
   True if block is an external reference.

Block.<code>is_xref_overlay</code>
   True if block is an external overlay reference.

Block.<code>is_anonymous</code>
   True if block is an anonymous block, created by hatch or dimension.

Block.<code>__iter__</code>:
   Support for iterator protocol, iterates over all block entities.

Block.<code>__getitem__(index)</code>:
   Returns block entity at location index, slicing is supported.

Block.<code>__len__()</code>:
   Returns count of block entities, support for standard len() function.

5.3 Line

class Line (<code>Shape</code>)

Line.<code>start</code>
   Start point of line (x, y[, z]) as tuple

Line.<code>end</code>
   End point of line (x, y[, z]) as tuple

5.4 Point

class Point (<code>Shape</code>)

Point.<code>point</code>
   Location of point (x, y[, z]) as tuple

5.5 Circle

class Circle (<code>Shape</code>)

Circle.<code>center</code>
   Location of circle center point (x, y[, z]) as tuple

Circle.<code>radius</code>
   Circle radius as float

5.3. Line
5.6 Arc

class Arc (Shape)

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

arc.radius
    Arc radius as float

arc.start_angle
    Arc start angle in degrees as float. (full circle = 360 degrees)

arc.end_angle
    Arc end angle in degrees as float. (full circle = 360 degrees)

5.7 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.

5.8 Trace

class Trace (Solid)

Same as Solid.

5.9 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].

5.10 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 big_font 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Left</td>
</tr>
<tr>
<td>1</td>
<td>Center</td>
</tr>
<tr>
<td>2</td>
<td>Right</td>
</tr>
<tr>
<td>3</td>
<td>Aligned (if vertical alignment = 0)</td>
</tr>
<tr>
<td>4</td>
<td>Middle (if vertical alignment = 0)</td>
</tr>
<tr>
<td>5</td>
<td>Fit (if vertical alignment = 0)</td>
</tr>
</tbody>
</table>

Text.valign
Vertical alignment as int.

<table>
<thead>
<tr>
<th>Value</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Baseline</td>
</tr>
<tr>
<td>1</td>
<td>Bottom</td>
</tr>
<tr>
<td>2</td>
<td>Middle</td>
</tr>
<tr>
<td>3</td>
<td>Top</td>
</tr>
</tbody>
</table>

Text.is_backwards
True if text is mirrored in X.

Text.is_upside_down
True if text is mirrored in Y.

Text.align_point
Second alignment point as tuple or None.

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

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

Text.plain_text()
Get text content without formatting codes like %%u.

5.10. Text
5.11 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.

5.12 Attdef

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

5.13 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.attribs
   List of Attrib entities attached to the Insert entity.

Insert.find_attrib(tag):
   Get Attrib entity by tag, returns None if not found.

5.14 Polyline

class Polyline (Shape)
   Multiple 2D- or 3D vertices connected by lines. The DXF entity POLYLINE is also used to define Polylines and Polymeshes, dxfgrabber defines separated classes for these entities see: Polyface and Polymesh.

Polyline.is_closed
   True if polyline is closed.
Polyline.\texttt{mode}
\begin{itemize}
\item Returns the polyline mode: \texttt{polyline2d}, \texttt{polyline3d} or \texttt{spline2d}.
\end{itemize}

Polyline.\texttt{spline\_type}
\begin{itemize}
\item If polyline is a 2D spline: \texttt{quadratic\_bspline}, \texttt{cubic\_bspline}, \texttt{bezier\_curve} else \texttt{None}.
\end{itemize}

Polyline.\texttt{default\_start\_width}
\begin{itemize}
\item Default line segment start width, if not set in vertex entity.
\end{itemize}

Polyline.\texttt{default\_end\_width}
\begin{itemize}
\item Default line segment end width, if not set in vertex entity.
\end{itemize}

Polyline.\texttt{points}
\begin{itemize}
\item List of all vertex locations as (x, y[, z]) \texttt{tuple}. If this polyline is a 2d spline these points are just the fit points.
\end{itemize}

Polyline.\texttt{control\_points}
\begin{itemize}
\item List of all control points as (x, y[, z]) \texttt{tuple}, if this polyline is a 2d spline.
\end{itemize}

Polyline.\texttt{tangents}
\begin{itemize}
\item List of all vertex tangent angles as \texttt{float} in degrees or \texttt{None} if not defined. (Just for fit points)
\end{itemize}

Polyline.\texttt{width}
\begin{itemize}
\item List of all vertex width values as (start\_width, end\_width) \texttt{tuple}. Just for fit points if this polyline is a 2D spline.
\end{itemize}

Polyline.\texttt{bulge}
\begin{itemize}
\item List of all vertex bulge values as \texttt{floats}.
\end{itemize}

Polyline.\texttt{\_getitem\_\_}(\texttt{index})
\begin{itemize}
\item Returns vertex \texttt{index} as \texttt{Vertex} entity. support for standard operator \texttt{vertex = polyline[index]}. Raises \texttt{IndexError}
\end{itemize}

Polyline.\texttt{\_len\_\_}()
\begin{itemize}
\item Returns count of vertices.
\end{itemize}

Polyline.\texttt{\_iter\_\_}()
\begin{itemize}
\item Iterate of all vertices, as \texttt{Vertex} entity.
\end{itemize}

### 5.15 Vertex

class \texttt{Vertex}(\texttt{Shape})

Vertex.\texttt{location}
\begin{itemize}
\item Location as (x, y, z)-tuple.
\end{itemize}

Vertex.\texttt{start\_width}

Vertex.\texttt{end\_width}

Vertex.\texttt{bulge}
\begin{itemize}
\item 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 \texttt{Autodesk}.
\end{itemize}

Vertex.\texttt{tangent}
\begin{itemize}
\item Curve fitting tangent in degrees as \texttt{float} or \texttt{None}. (full circle = 360 degrees)
\end{itemize}
5.16 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No smooth surface fitted</td>
</tr>
<tr>
<td>5</td>
<td>Quadratic B-spline surface</td>
</tr>
<tr>
<td>6</td>
<td>Cubic B-spline surface</td>
</tr>
<tr>
<td>8</td>
<td>Bezier surface</td>
</tr>
</tbody>
</table>

5.16.1 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.

5.17 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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No smooth surface fitted</td>
</tr>
<tr>
<td>5</td>
<td>Quadratic B-spline surface</td>
</tr>
<tr>
<td>6</td>
<td>Cubic B-spline surface</td>
</tr>
<tr>
<td>8</td>
<td>Bezier surface</td>
</tr>
</tbody>
</table>

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).

5.18 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

LWPolyline.__iter__()
   Iterate over all polyline points.

5.19 Ellipse

class Ellipse(Shape)

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

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

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

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

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

5.20 Ray

class Ray(Shape)

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

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

5.21 XLine

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

5.22 Spline

class Spline(Shape)

Spline.flags
   Binary coded flags, constants stored in dxfgrabber.const.
<table>
<thead>
<tr>
<th>Spline.flags</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLINE_CLOSED</td>
<td>1</td>
</tr>
<tr>
<td>SPLINE_PERIODIC</td>
<td>2</td>
</tr>
<tr>
<td>SPLINE_RATIONAL</td>
<td>4</td>
</tr>
<tr>
<td>SPLINE_PLANAR</td>
<td>8</td>
</tr>
<tr>
<td>SPLINE_LINEAR</td>
<td>16</td>
</tr>
</tbody>
</table>

Spline.

- **.degree**
  Degree of the spline curve as `int`

- **.start_tangent**
  Start tangent as `(x, y, z)` as `tuple` or `None`

- **.end_tangent**
  End tangent as `(x, y, z)` as `tuple` or `None`

- **.control_points**
  List of control points `(x, y, z)` as `tuple`

- **.fit_points**
  List of fit points `(x, y, z)` as `tuple`

- **.knots**
  List of knot values as `float`

- **.weights**
  List of weight values as `float`

- **.normal_vector**
  Normal vector if spline is planar else `None`.

- **.is_closed**

- **.is_periodic**

- **.is_rational**

- **.is_planar**

- **.is_linear**

### 5.23 Helix

3D spiral; Helix is also a Spline.

**class Helix (Spline)**

- **.helix_version**
  Tuple (main version, maintainance version)

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

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

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

- **.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

## 5.24 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.

```python
class MText(Shape)
    Multi line text entity.

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

    .raw_text
        Whole text content as one string.

    .height
        Text height as float

    .rect_width
        Reference rectangle width as float in drawing units.

    .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.

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

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

    .attachment_point
        Text attachment point as int.
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Top left</td>
</tr>
<tr>
<td>2</td>
<td>Top center</td>
</tr>
<tr>
<td>3</td>
<td>Top right</td>
</tr>
<tr>
<td>4</td>
<td>Middle left</td>
</tr>
<tr>
<td>5</td>
<td>Middle center</td>
</tr>
<tr>
<td>6</td>
<td>Middle right</td>
</tr>
<tr>
<td>7</td>
<td>Bottom left</td>
</tr>
<tr>
<td>8</td>
<td>Bottom center</td>
</tr>
<tr>
<td>9</td>
<td>Bottom right</td>
</tr>
</tbody>
</table>
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.big_font
   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 

5.25 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

5.26 Light

class Light (Shape)
   Defines a light source.

Light.version

Light.name
**Light**

- **light_type**
  - distant = 1; point = 2; spot = 3

- **status**
  - Boolean value: on/off?

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

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

- **plot_glyph**
  - Boolean value

- **intensity**

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

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

- **attenuation_type**
  - 0 = None; 1 = Inverse Linear; 2 = Inverse Square

- **use_attenuation_limits**
  - Boolean value

- **attenuation_start_limit**

- **attenuation_end_limit**

- **hotspot_angle**

- **fall_off_angle**

- **cast_shadows**
  - Boolean value

- **shadow_type**
  - 0 = Ray traced shadows; 1 = Shadow maps

- **shadow_map_size**

- **shadow_softness**

---

**5.27 Mesh**

- **class Mesh (Shape)**
  - 3D mesh entity similar to the *Polyface* entity.

- **version**

- **blend_crez**
  - Boolean value (on/off)

- **subdivision_levels**

- **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:

```python
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:

```python
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.

## 5.28 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.

## 5.29 Region

class **Region**(Body)
ACIS based 2D enclosed areas.

## 5.30 3DSolid

class **3DSolid**(Body)
ACIS based 3D solid geometry.
5.31 Surface

class Surface (Body)
   ACIS based 3D freeform surfaces.

5.32 PlaneSurface

class PlaneSurface (Surface)
   ACIS based 3D plane surfaces.
6.1 Open a DXF file

Open files from file system:

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

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

6.2 Query Header Variables

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

Example:

```python
dxfversion = dwg.header['$ACADVER']
```

For available HEADER variables and their meaning see: DXF Reference

6.3 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:

```python
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']
```
6.4 Query Blocks

Block references are just DXF entities called INSERT.

Get all block references for block TestBlock:

```python
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:

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

and use the in operator (Iterator protocol):

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

6.5 Layers

Layers are nothing special, they are just another attribute of the DXF entity, dxfgrabber 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:

```python
layer = dwg.layers[dfx_entity.layer]
color = layer.color if dfx_entity.color == dxfgrabber.BYLAYER else dfx_entity.color
linetype = layer.linetype if dfx_entity.linetype is None else dfx_entity.linetype
```

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

6.6 Layouts (Modelspace or Paperspace)

dxfgrabber 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:

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

shortcuts since 0.5.1:

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

__contains__() (BlocksSection method), 12
__getitem__() (BlocksSection method), 12
__getitem__() (EntitySection method), 12
__getitem__() (LWPolyline method), 21
__getitem__() (LayerTable method), 10
__getitem__() (LinetypeTable method), 11
__getitem__() (Polyface method), 20
__getitem__() (Polyline method), 19
__getitem__() (StyleTable method), 11
__iter__() (BlocksSection method), 12
__iter__() (EntitySection method), 12
__iter__() (LWPolyline method), 22
__iter__() (LayerTable method), 10
__iter__() (LinetypeTable method), 12
__iter__() (Polyface method), 20
__iter__() (Polyline method), 19
__iter__() (StyleTable method), 11
__len__() (BlocksSection method), 12
__len__() (EntitySection method), 12
__len__() (LWPolyline method), 21
__len__() (LayerTable method), 10
__len__() (LinetypeTable method), 12
__len__() (Polyface method), 20
__len__() (Polyline method), 19
__len__() (StyleTable method), 11
__len__() (SubFace method), 20
3DSolid (built-in class), 27

attrs (Insert attribute), 18
axis_base_point (Helix attribute), 23
axis_vector (Helix attribute), 23

B

b (TrueColor attribute), 14
basepoint (Block attribute), 14
big_font (MText attribute), 25
big_font (Style attribute), 11
big_font (Text attribute), 17
blend_crease (Mesh attribute), 26
Block (built-in class), 14
blocks (Drawing attribute), 9
BlocksSection (built-in class), 12
Body (built-in class), 27
bulge (LWPolyline attribute), 21
bulge (Polyline attribute), 19
bulge (Vertex attribute), 19

C

cast_shadows (Light attribute), 26
center (Arc attribute), 16
center (Circle attribute), 15
center (Ellipse attribute), 22
Circle (built-in class), 15
col_count (Insert attribute), 18
col_spacing (Insert attribute), 18
color (Layer attribute), 10
color (Shape attribute), 13
const_width (LWPolyline attribute), 21
constrain (Helix attribute), 24
control_points (Polyline attribute), 19
control_points (Spline attribute), 23

date (Sun attribute), 25
daylight_savings_time (Sun attribute), 25
default_end_width (Polyline attribute), 19
default_start_width (Polyline attribute), 19

A

aci_to_true_color(), 7
acis (Body attribute), 27
align_point (Text attribute), 17
Arc (built-in class), 16
attachment_point (MText attribute), 24
attenuation_end_limit (Light attribute), 26
attenuation_start_limit (Light attribute), 26
attenuation_type (Light attribute), 26
Attrib (built-in class), 18
degree (Spline attribute), 23
Drawing (built-in class), 9
dxftype (Shape attribute), 13
dxfversion (Drawing attribute), 9

ddx_rectangle (Mesh attribute), 27
edges (Mesh attribute), 27
elevation (LWPolyline attribute), 21
Ellipse (built-in class), 22
encoding (Drawing attribute), 9
end (Line attribute), 15
dend_angle (arc attribute), 16
ed_end_param (Ellipse attribute), 22
dend_tangent (Spline attribute), 23
dend_width (Vertex attribute), 19
entities (Drawing attribute), 10
EntitySection (built-in class), 12
extrusion (Shape attribute), 13

Face (built-in class), 16
face_record (SubFace attribute), 20
faces (Mesh attribute), 26
fall_off_angle (Light attribute), 26
filename (Drawing attribute), 9
fit_points (Spline attribute), 23
flags (Block attribute), 15
flags (Spline attribute), 22
font (MText attribute), 25
font (Style attribute), 11
font (Text attribute), 17
from_aci() (TrueColor method), 14
from_rgb() (TrueColor method), 14
frozen (Layer attribute), 10

G

g (TrueColor attribute), 14
get() (BlocksSection method), 12
get() (LayerTable method), 10
get() (LinetypeTable method), 11
get() (StyleTable method), 11
get_edge() (Mesh method), 27
get_face() (Mesh method), 27
get_location() (Polymesh method), 21
get_vertex() (Polymesh method), 21

H

halign (Text attribute), 17
handedness (Helix attribute), 24
header (Drawing attribute), 9
height (MText attribute), 24
height (Style attribute), 11

height (Text attribute), 17
Helix (built-in class), 23
helix_version (Helix attribute), 23
horizontal_width (MText attribute), 24
hotspot_angle (Light attribute), 26

I

Insert (built-in class), 18
insert (Insert attribute), 18
insert (MText attribute), 24
insert (Text attribute), 16
intensity (Light attribute), 26
intensity (Sun attribute), 25
invisible (Shape attribute), 13
is_anonymous (Block attribute), 15
is_backwards (Style attribute), 11
is_backwards (Text attribute), 17
is_closed (LWPolyline attribute), 21
is_closed (Polyline attribute), 18
is_closed (Spline attribute), 23
is_edge_invisible() (Face method), 16
is_linear (Spline attribute), 23
is_mclosed (Polymesh attribute), 21
is_nclosed (Polymesh attribute), 21
is_periodic (Spline attribute), 23
is_planar (Spline attribute), 23
is_rational (Spline attribute), 23
is_sab (Body attribute), 27
is_sat (Body attribute), 27
is_upside_down (Style attribute), 11
is_upside_down (Text attribute), 17
is_xref (Block attribute), 15
is_xref_overlay (Block attribute), 15

K

knots (Spline attribute), 23

L

Layer (built-in class), 10
layer (Shape attribute), 13
layers (Drawing attribute), 9
LayerTable (built-in class), 10
Light (built-in class), 25
light_color (Light attribute), 26
light_type (Light attribute), 25
Line (built-in class), 15
line_spacing (MText attribute), 24
lines() (MText method), 25
Linetype (built-in class), 12
linetype (Layer attribute), 10
linetype (Shape attribute), 13
linetypes (Drawing attribute), 9
LinetypeTable (built-in class), 11
location (Vertex attribute), 19

Index
locked (Layer attribute), 10
ltscale (Shape attribute), 13
LWPolyline (built-in class), 21

M
m_smooth_density (Polymesh attribute), 21
major_axis (Ellipse attribute), 22
mcount (Polymesh attribute), 21
Mesh (built-in class), 26
mode (Polyline attribute), 18
modelspace() (Drawing method), 10
MText (built-in class), 24

N
n_smooth_density (Polymesh attribute), 21
name (Block attribute), 14
name (Insert attribute), 18
name (Layer attribute), 10
name (Light attribute), 25
name (Style attribute), 11
names() (LayerTable method), 10
names() (LinetypeTable method), 12
names() (StyleTable method), 11
ncount (Polymesh attribute), 21
normal_vector (Spline attribute), 23

O
objects (Drawing attribute), 10
oblique (Style attribute), 11
oblique (Text attribute), 17
on (Layer attribute), 10

P
paperspace (Shape attribute), 13
paperspace() (Drawing method), 10
plain_text() (MText method), 25
plain_text() (Text method), 17
PlaneSurface (built-in class), 28
plot_glyph (Light attribute), 26
Point (built-in class), 15
point (Point attribute), 15
points (Face attribute), 16
points (LWPolyline attribute), 21
points (Polyline attribute), 19
points (Solid attribute), 16
Polyface (built-in class), 20
Polyline (built-in class), 18
Polymesh (built-in class), 20
position (Light attribute), 26

R
r (TrueColor attribute), 14
radius (Circle attribute), 15
radius (Helix attribute), 23
ratio (Ellipse attribute), 22
raw_text (MText attribute), 24
Ray (built-in class), 22
read() (built-in function), 5
readfile() (built-in function), 5
rect_width (MText attribute), 24
Region (built-in class), 27
rgb() (TrueColor method), 14
rotation (Insert attribute), 18
rotation (Text attribute), 17
row_count (Insert attribute), 18
tangent (Vertex attribute), 19

S
data (Insert attribute), 18
shadow_map_size (Light attribute), 26
shadow_map_size (Sun attribute), 25
shadow_mode (Shape attribute), 13
shadow_softness (Light attribute), 26
shadow_softness (Sun attribute), 25
shadow_type (Light attribute), 26
shadow_type (Sun attribute), 25
Shadows (Sun attribute), 25
Shape (built-in class), 13
smooth_type (Polyface attribute), 20
smooth_type (Polymesh attribute), 21
Solid (built-in class), 16
Spline (built-in class), 22
spline_type (Polyline attribute), 19
start (Line attribute), 15
start (Ray attribute), 22
start_angle (arc attribute), 16
start_param (Ellipse attribute), 22
start_point (Helix attribute), 23
start_tangent (Spline attribute), 23
start_width (Vertex attribute), 19
status (Light attribute), 26
status (Sun attribute), 25
Style (built-in class), 11
style (MText attribute), 24
style (Text attribute), 17
styles (Drawing attribute), 9
StyleTable (built-in class), 11
subdivision_levels (Mesh attribute), 26
SubFace (built-in class), 20
Sun (built-in class), 25
sun_color (Sun attribute), 25
Surface (built-in class), 28

t (TrueColor attribute), 14
radius (arc attribute), 16

Index 33
tangents (Polyline attribute), 19
target (Light attribute), 26
Text (built-in class), 16
text (Text attribute), 17
thickness (Shape attribute), 13
Trace (built-in class), 16
transparency (Shape attribute), 13
true_color (Light attribute), 26
true_color (Shape attribute), 13
TrueColor (built-in class), 14
turn_height (Helix attribute), 24
turns (Helix attribute), 23

U
unit_vector (Ray attribute), 22
use_attenuation_limits (Light attribute), 26

V
valign (Text attribute), 17
version (Light attribute), 25
version (Mesh attribute), 26
version (Sun attribute), 25
Vertex (built-in class), 19
vertical_height (MText attribute), 24
vertices (Mesh attribute), 26
vertices (Polyface attribute), 20

W
weights (Spline attribute), 23
width (LWPolyline attribute), 21
width (Polyline attribute), 19
width (Style attribute), 11
width (Text attribute), 17

X
xdirection (MText attribute), 25
XLine (built-in class), 22
xrefpath (Block attribute), 15