pyacad

Release 0.0.5

Kevin Axel Tagliaferri

Aug 20, 2024

CONTENTS:

A package aimed to simplfy coding of Activex Automation Module of AutoCAD, based on pywin32 library.

CHAPTER

ONE

GETTING STARTED

1.1 Installation

If you have pip:

```
pip install pyacad
```

1.2 Requirements

- pywin32

1.3 Retrieving AutoCAD ActiveX documentation

Visit the official Autodesk site:

- AutoCAD 2024
- AutoCAD 2023

CHAPTER

TWO

USAGE

For the following examples, we will use Autocad and APoint.

```
from pyacad import Autocad
```

2.1 Initializing the AutoCAD API:

```
acad = Autocad()
```

2.2 Retrieving active document name

```
# Only the document name.
acad.doc.Name

# For the document path.
acad.doc.FullName
```

2.3 Adding objects to the active document

2.3.1 Points

```
p0 = APoint(2, 1)
point = acad.model.AddPoint(p0())
```

2.3.2 Lines

```
p0, p1 = APoint(1, 1), APoint(2, 1)
line = acad.model.AddLine(p0(), p1())
```

2.3.3 Polylines

For drawing polylines we need to use aDouble.

```
from pyacad import aDouble
points = aDouble([0, 0, 1, 0, 1, 1, 0, 1, 0, 0])
polyline = acad.model.AddLightweightPolyline(points)
```

2.3.4 Circles

```
p0 = APoint(3, 1)
radius = .5
circle = acad.model.AddCircle(p0(), radius)
```

2.3.5 Text

```
p0 = APoint(0, 3)
height = 1
textstring = "Hello World!"
text = acad.model.AddText(textstring, p0(), height)
```

2.3.6 MultiLineText

```
p0 = APoint(0, 4)
width = 1
textstring = "This is a MText."
mtext = acad.model.AddMText(p0(), width, textstring)
```

2.3.7 Hatch

For drawing hatchs we need to use aDispatch.

```
# Defining boundary.
outer_boundary = []
outer_boundary.append(acad.model.AddCircle(APoint(0, 0)(), 1))
outer_boundary_dispatch = aDispatch(outer_boundary)

# Creating hatch and adding boundary.
hatch = acad.model.AddHatch(0, "ANSI31", True)
hatch.AppendOuterLoop(outer_boundary_dispatch)
hatch.Evaluate()
```

2.3.8 Aligned Dimension

```
p0, p1, p2 = APoint(0, 4), APoint(4, 4), APoint(2, 4.5)
acad.model.AddDimAligned(p0(), p1(), p3())
```

2.3.9 Further information about objects

For more information on creating objects, I recommend visiting the official Autodesk site on ActiveX Automation for your corresponding version of AutoCAD.

- AutoCAD 2024
- AutoCAD 2023

2.4 Retrieving over documents, layouts, layer, objects and more.

2.4.1 Retrieving documents from the AutoCAD API

```
acad.iter_documents()
```

2.4.2 Retrieving blocks from the document

```
blocks = acad.iter_blocks()
```

Alternatively, you can pass a specific document using the document parameter, which should be of the type returned by the app.iter_documents() function.

```
docs = [*acad.iter_documents()]
doc_selected = docs[0]  # 0 if you want select first document of list.
blocks = acad.iter_blocks(document=doc_selected)
```

2.4.3 Retrieving dimension styles from the document

```
dim_styles = acad.iter_dim_styles()
```

You can also do it in the same way as shown in iter_blocks().

2.4.4 Retrieving layers from the document

```
layers = acad.iter_layers()
```

You can also do it in the same way as shown in iter_blocks().

2.4.5 Retrieving layouts from the document

```
layouts = acad.iter_layouts()
```

You can also do it in the same way as shown in iter_blocks().

2.4.6 Retrieving objects from the document

You can iterate over the objects in a drawing.

```
objects = acad.iter_objects()
```

Also you can filter for a concrete obejct type.

```
text_obejects = acad.iter_objects("Text")
```

2.4.7 Retrieving text styles from the document

```
text_styles = acad.iter_text_styles()
```

You can also do it in the same way as shown in iter_blocks().

CHAPTER

THREE

REFERENCES

Since Read the Docs compiles documentation in a Linux environment, the automatic generation of documentation using Sphinx is hindered because pywin32 only supports Win32 environments. Given this, we have compiled the documentation locally and uploaded it in PDF format to the following link at GitHub.

API Documentation

`class pyacad.Autocad.`**`Autocad`**`(`*create_if_not_exist: bool = True*, *visible: bool = True*`)`

Main class of AutoCAD app.

`property app`

Creates/gets and returns AutoCAD Application.

`property doc`

Returns active document.

`iter_blocks``(`*document=None*`)`

Iterate over the existing block definitions in the specified document.

Parameters

`document` (*`win32com.client.CDispatch, None`*) – COM object returned from the AutoCAD application. If *None* is specified, the active document is used.

Yield

Generator of block definitions in the specified document.

Return type

generator

`iter_dim_styles``(`*document=None*`)`

Iterate over the existing dimension styles in the specified document.

Parameters

`document` (*`win32com.client.CDispatch, None`*) – COM object returned from the AutoCAD application. If *None* is specified, the active document is used.

Yield

Generator of dimension styles in the specified document.

Return type

generator

`iter_layers``(`*document=None*`)`

Iterate over the layers in the specified document.

Parameters

`document` (*`win32com.client.CDispatch, None`*) – COM object returned from the AutoCAD application. If *None* is specified, the active document is used.

Yield

Generator of layers in the specified document.

Return type
generator

`iter_layouts`(*document=None*, *skip_model=False*)
Iterate over the layouts in the specified document.

Parameters

- `document` (*`win32com.client.CDispatch, None`*) – COM object returned from the AutoCAD application. If *None* is specified, the active document is used.
- `skip_model` (*`bool`*) – Whether to skip the model layout. Defaults to *False*.

Yield
Generator of layouts in the specified document.

Return type
generator

`iter_objects`(*block=None*, *filter_for=None*, *limit=None*)
Iterate over the objects in a specified 'block'.

Parameters

- `block` (*`win32com.client.CDispatch, None`*) – COM object returned from the AutoCAD application. If *None* is specified, the active layout is used.
- `filter_for` (*`list of str, tuple of str, None`*) – A filter for specific object types. Can be a list or tuple of strings. If *None*, no filtering is applied.
- `limit` (*`int, None`*) – The maximum number of objects to iterate over. If *None*, no limit is applied.

Yield
Generator of objects in the specified layout or active layout if none is specified.

Return type
generator

`iter_text_styles`(*document=None*)
Iterate over the existing text styles in the specified document.

Parameters
`document` (*`win32com.client.CDispatch, None`*) – COM object returned from the AutoCAD application. If *None* is specified, the active document is used.

Yield
Generator of text styles in the specified document.

Return type
generator

`property model`
Returns active model space of curring document.

`class pyacad.APoint.APoint`(*x: float*, *y: float = 0*, *z: float = 0*)
3D point with basic geometric operations and support for passing as a parameter for *AutoCAD* Automation functions.

Variables

- `x` (*`int, float, list of int, list of float, tuple of int, tuple of float`*) – The X coordinate of the point.
- `y` (*`int, float, None`*) – The Y coordinate of the point.
- `z` (*`int, float, None`*) – The Z coordinate of the point.

`distance_to`(*other*)

Calculate the distance to another 3D point.

Parameters
`other` (*Point3D*) – The other Point3D object to calculate the distance to.

Returns
The distance between the two points.

Return type
float

`pyacad.APoint.distance`(*p1*, *p2*)

Calculate the distance to another 3D point.

Parameters

- `p1` (*Point3D*) – The other Point3D object to calculate the distance to.
- `p2` (*Point3D*) – The other Point3D object to calculate the distance to.

Returns
The distance between the two points.

Return type
float

`pyacad.Types.aDispatch`(*object*)

Packs a win32 object into Variant Array.

Parameters
`object` – Win32 object.

Returns
An array variant suitable for AutoCAD.

Return type
VARIANT

`pyacad.Types.aDouble`(*xyz*)

Packs a list or tuple of floats into an array for passing to AutoCAD.

Parameters
`xyz` (*list of float, tuple of float*) – A tuple or list of floats to be packed into an array.

Returns
An array variant suitable for AutoCAD.

Return type
VARIANT

`pyacad.Types.aInt`(*xyz*)

Packs list of floats into an array for passing to AutoCAD (same as aDouble).

Parameters
`xyz` (*list of float, tuple of float*) – List of floats or integers.

Returns
An array variant suitable for AutoCAD.

Return type
VARIANT

(˘˘) If you want to pay me for a beer, coffee, or something else: C|_|

PYTHON MODULE INDEX