pyautocad Documentation

Release 0.2.0

Roman Haritonov

December 08, 2016
## Contents

1 Contents:  
1.1 Getting started ................................................. 3  
1.2 Usage .......................................................... 4  
1.3 API ............................................................ 6  

2 Indices and tables  

Python Module Index  

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>3</td>
</tr>
<tr>
<td>Getting started</td>
<td>3</td>
</tr>
<tr>
<td>Usage</td>
<td>4</td>
</tr>
<tr>
<td>API</td>
<td>6</td>
</tr>
<tr>
<td>Indices and tables</td>
<td>13</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>15</td>
</tr>
</tbody>
</table>
pyautocad - library aimed to simplify writing ActiveX Automation scripts for AutoCAD with Python
1.1 Getting started

1.1.1 Installation

If you have pip or easy_install, you can just:

```bash
pip install --upgrade pyautocad
```

or:

```bash
easy_install -U pyautocad
```

Also, you can download Windows installer from PyPI pyautocad page.

1.1.2 Requirements

- comtypes
  
  **Note:** If you are using pip or easy_install, then it will be installed automatically. Otherwise you should install comtypes package manually.

- Optional: xld and tablib for working with tables

1.1.3 Retrieving AutoCAD ActiveX documentation

A copy of the AutoCAD ActiveX guide and reference can be found in the help directory of your AutoCAD install.

- acad_aag.chm - ActiveX and VBA Developer’s Guide
- acadauto.chm - ActiveX and VBA Reference

Reference can also be found in C:\Program Files\Common Files\Autodesk Shared\acadauto.chm
1.1.4 What’s next?

Read the Usage section, or look for real applications in examples folder of source distribution.

Note: Applications in examples are Russian engineering specific, but anyway I hope you’ll find something interesting in that code.

For more info on features see API documentation and sources.

1.2 Usage

1.2.1 Main interface and types

For our first example, we will use Autocad (main Automation object) and pyautocad.types.APoint for operations with coordinates

```python
from pyautocad import Autocad, APoint
```

Let’s create AutoCAD application or connect to already running application:

```python
acad = Autocad(create_if_not_exists=True)
acad.prompt("Hello, Autocad from Python\n")
print acad.doc.Name
```

To work with AutoCAD documents and objects we can use ActiveX interface, Autocad (from pyautocad) contains some methods to simplify common Automation tasks, such as object iteration and searching, getting objects from user’s selection, printing messages.

There are shortcuts for current ActiveDocument - Autocad.doc and ActiveDocument.ModelSpace - Autocad.model

Let’s add some objects to document:

```python
p1 = APoint(0, 0)
p2 = APoint(50, 25)
for i in range(5):
    text = acad.model.AddText(u'Hi %s!' % i, p1, 2.5)
    acad.model.AddLine(p1, p2)
    acad.model.AddCircle(p1, 10)
p1.y += 10
```

Now our document contains some Texts, Lines and Circles, let’s iterate them all:

```python
for obj in acad.iter_objects():
    print obj.ObjectName
```

We can also iterate objects of concrete type:

```python
for text in acad.iter_objects('Text'):
    print text.TextString, text.InsertionPoint
```

Note: Object name can be partial and case insensitive, e.g. acad.iter_objects('tex') will return AcDbText and AcDbMText objects
Or multiple types:

```python
for obj in acad.iter_objects(['Text', 'Line']):
    print obj.ObjectName
```

Also we can find first object with some conditions. For example, let’s find first text item which contains 3:

```python
def text_contains_3(text_obj):
    return '3' in text_obj.TextString

text = acad.find_one('Text', predicate=text_contains_3)
print text.TextString
```

To modify objects in document, we need to find interesting objects and change its properties. Some properties are described with constants, e.g. text alignment. These constants can be accessed through `ACAD`. Let’s change all text objects text alignment:

```python
from pyautocad import ACAD

for text in acad.iter_objects('Text'):
    old_insertion_point = APoint(text.InsertionPoint)
    text.Alignment = ACAD.acAlignmentRight
    text.TextAlignmentPoint = old_insertion_point
```

In previous code we have converted `text.InsertionPoint` to `APoint` because we can’t just use default tuple when setting another properties such as `text.TextAlignmentPoint`.

If we need to change position of some object, we should use `APoint`, for example let’s change lines end position:

```python
for line in acad.iter_objects('Line'):
    p1 = APoint(line.StartPoint)
    line.EndPoint = p1 - APoint(20, 0)
```

### 1.2.2 Working with tables

**Note:** To work with tables, `xlrde` and `tablib` should be installed.

To simplify importing and exporting data there is `Table` class exist. It allows you to read and write tabular data in popular formats:

- csv
- xls
- xlsx (write only)
- json

Let’s try to solve some basic task. We need to save text and position from all text objects to Excel file, and then load it back.

First we need to add some objects to AutoCAD:

```python
from pyautocad import Autocad, APoint
from pyautocad.contrib.tables import Table

acad = Autocad()
p1 = APoint(0, 0)
for i in range(5):
```

### 1.2. Usage
Now we can iterate this objects and save them to Excel table:

```python
obj = acad.model.AddText(u'Hi %s!' % i, p1, 2.5)
p1.y += 10
```

Table objects can be saved to Excel table:

```python
table = Table()
for obj in acad.iter_objects('Text'):
    x, y, z = obj.InsertionPoint
    table.writerow([obj.TextString, x, y, z])
table.save('data.xls', 'xls')
```

After saving this data to `data.xls` and probably changing it with some table processor software (e.g. Microsoft Office Excel) we can retrieve our data from file:

```python
data = Table.data_from_file('data.xls')
data
```

will contain:

```python
[
    [u'Hi 0!', 0.0, 0.0, 0.0],
    [u'Hi 1!', 0.0, 10.0, 0.0],
    [u'Hi 2!', 0.0, 20.0, 0.0],
    [u'Hi 3!', 0.0, 30.0, 0.0],
    [u'Hi 4!', 0.0, 40.0, 0.0]]
```

See also:

Example of working with AutoCAD table objects at examples/dev_get_table_info.py

### 1.2.3 Improve speed

- ActiveX technology is quite slow. When you are accessing object attributes like position, text, etc, every time call is passed to AutoCAD. It can slowdown execution time. For example if you have program, which combines single line text based on its relative positions, you probably need to get each text position several times. To speed this up, you can cache objects attributes using the `pyautocad.cache.Cached` proxy (see example in class documentation).

- To improve speed of AutoCAD table manipulations, you can use `Table.RegenerateTableSuppressed = True` or handy context manager `suppressed_regeneration_of(table):

```python
table = acad.model.AddTable(pos, rows, columns, row_height, col_width)
with suppressed_regeneration_of(table):
    table.SetAlignment(ACAD.acDataRow, ACAD.acMiddleCenter)
    for row in range(rows):
        for col in range(columns):
            table.SetText(row, col, 's %s %s' % (row, col))
```

### 1.2.4 Utility functions

There is also some utility functions for work with AutoCAD text objects and more. See `pyautocad.utils` documentation.

### 1.3 API

This part of the documentation covers all the interfaces of `pyautocad`
1.3.1 api - Main Autocad interface

```python
class pyautocad.api.AutoCAD(create_if_not_exists=False, visible=True)
    Main AutoCAD Automation object

    Parameters
    • create_if_not_exists – if AutoCAD doesn’t run, then new instance will be crated
    • visible – new AutoCAD instance will be visible if True (default)

app
    Returns active AutoCAD.Application
    if AutoCAD was created with create_if_not_exists=True, it will create
    AutoCAD.Application if there is no active one

doc
    Returns ActiveDocument of current Application

ActiveDocument
    Same as doc

Application
    Same as app

model
    ModelSpace from active document

iter_layouts(doc=None, skip_model=True)
    Iterate layouts from doc

    Parameters
    • doc – document to iterate layouts from if doc=None (default), ActiveDocument is used
    • skip_model – don’t include ModelSpace if True

iter_objects(object_name_or_list=None, block=None, limit=None, dont_cast=False)
    Iterate objects from block

    Parameters
    • object_name_or_list – part of object type name, or list of it
    • block – Autocad block, default - ActiveDocument.ActiveLayout.Block
    • limit – max number of objects to return, default infinite
    • dont_cast – don’t retrieve best interface for object, may speedup iteration. Returned objects should be casted by caller

iter_objects_fast(object_name_or_list=None, container=None, limit=None)
    Shortcut for iter_objects(dont_cast=True)
    Shouldn’t be used in normal situations

find_one(object_name_or_list, container=None, predicate=None)
    Returns first occurrence of object which match predicate

    Parameters
    • object_name_or_list – like in iter_objects()
    • container – like in iter_objects()
• **predicate** – callable, which accepts object as argument and returns *True* or *False*

**Returns**  Object if found, else *None*

```python
best_interface (obj)
```
Retrieve best interface for object

```python
prompt (text)
```
Prints text in console and in *AutoCAD* prompt

```python
get_selection (text='Select objects')
```
Asks user to select objects

**Parameters**  **text** – prompt for selection

```python
static aDouble (*seq)
```
shortcut for *pyautocad.types.aDouble()*

```python
static aInt (*seq)
```
shortcut for *pyautocad.types.aInt()*

```python
static aShort (*seq)
```
shortcut for *pyautocad.types.aShort()*

```python
pyautocad.api.ACAD
```
Constants from *AutoCAD* type library, for example:

| text.Alignment = ACAD.acAlignmentRight |

---

### 1.3.2 *types* - 3D Point and other *AutoCAD* data types

**class** *pyautocad.types.APoint*

3D point with basic geometric operations and support for passing as a parameter for *AutoCAD* Automation functions

**Usage:**

```python
>>> p1 = APoint(10, 10)
>>> p2 = APoint(20, 20)
>>> p1 + p2
APoint(30.00, 30.00, 0.00)
```

Also it supports iterable as parameter:

```python
>>> APoint([10, 20, 30])
APoint(10.00, 20.00, 30.00)
>>> APoint(range(3))
APoint(0.00, 1.00, 2.00)
```

Supported math operations: `+`, `*`, `/`, `+=`, `-=`, `*=` ,`/=`

```python
>>> p = APoint(10, 10)
>>> p + p
APoint(20.00, 20.00, 0.00)
>>> p + 10
APoint(20.00, 20.00, 10.00)
>>> p * 2
APoint(20.00, 20.00, 0.00)
>>> p -= 1
```
>>> p
APoint(9.00, 9.00, -1.00)

It can be converted to tuple or list:

>>> tuple(APoint(1, 1, 1))
(1.0, 1.0, 1.0)

**x**

x coordinate of 3D point

**y**

y coordinate of 3D point

**z**

z coordinate of 3D point

**distance_to** *(other)*

Returns distance to other point

**Parameters**

other – APoint instance or any sequence of 3 coordinates

```python
pyautocad.types.distance(p1, p2)
```

Returns distance between two points p1 and p2

```python
pyautocad.types.aDouble(*seq)
```

Returns array.array of doubles (’d’ code) for passing to AutoCAD

For 3D points use APoint instead.

```python
pyautocad.types.aInt(*seq)
```

Returns array.array of ints (’l’ code) for passing to AutoCAD

```python
pyautocad.types.aShort(*seq)
```

Returns array.array of shorts (’h’ code) for passing to AutoCAD

### 1.3.3 utils - Utility functions

```python
pyautocad.utils.timing(message)
```

Context manager for timing execution

**Parameters** message – message to print

Usage:

```python
with timing('some operation'):
    do_some_actions()
```

Will print:

```python
some operation: 1.000 s # where 1.000 is actual execution time
```

```python
pyautocad.utils.suppressed_regeneration_of(table)
```

New in version 0.1.2.

Context manager. Suppresses table regeneration to dramatically speedup table operations

**Parameters** table – table object
with suppressed_regeneration_of(table):
    populate(table)  # or change its properties

pyautocad.utils.unformat_mtext(s, exclude_list=('P', 'S'))
Returns string with removed format information

    • s – string with multitext
    • exclude_list – don’t touch tags from this list. Default (‘P’, ‘S’) for newline and frac-

>>> text = ur'\fGOST type A|b0|i0|c204|p34;TEST\fGOST type A|b0|i0|c0|p34;123'
>>> unformat_mtext(text)
u'TEST123'

pyautocad.utils.mtext_to_string(s)
Returns string with removed format information as unformat_mtext() and P (paragraphs) replaced with

newlines

>>> text = ur'\fGOST type A|b0|i0|c204|p34;TEST\fGOST type A|b0|i0|c0|p34;123\Ptest321'
>>> mtext_to_string(text)
u'TEST123\nmttest321'

pyautocad.utils.string_to_mtext(s)
Returns string in Autocad multitext format

    Replaces newlines \n with \P, etc.

pyautocad.utils.text_width(text_item)
Returns width of Autocad Text or MultiText object

pyautocad.utils.dynamic_print(text)
Prints text dynamically in one line

    Used for printing something like animations, or progress

1.3.4 contrib.tables - Import and export tabular data from popular formats

class pyautocad.contrib.tables.Table
    Represents table with ability to import and export data to following formats:
        • csv
        • xls
        •xlsx (write only)
        • json

When you need to store some data, it can be done as follows:

table = Table()
for i in range(5):
    table.writerow([i, i, i])
table.save('data.xls', 'xls')
To import data from file, use `data_from_file()`:

```python
data = Table.data_from_file('data.xls')
```

**writerow(row)**

Add row to table

**Parameters**

- `row` (list or tuple) – row to add

**append(row)**

Synonym for `writerow()`

**clear()**

Clear current table

**save(filename, fmt, encoding='cp1251')**

Save data to file

**Parameters**

- `filename` – path to file
- `fmt` – data format (one of supported, e.g. ‘xls’, ‘csv’)
- `encoding` – encoding for ‘csv’ format

**convert(fmt)**

Return data, converted to format

**Parameters**

- `fmt` – desirable format of data

**Note:** to convert to `csv` format, use `to_csv()`

See also `available_write_formats()`

**to_csv(stream, encoding='cp1251', delimiter=';', **kwargs)**

Writes data in `csv` format to stream

**Parameters**

- `stream` – stream to write data to
- `encoding` – output encoding
- `delimiter` – `csv` delimiter
- `kwargs` – additional parameters for `csv.writer`

**static data_from_file(filename, fmt=None, csv_encoding='cp1251', csv_delimiter=';')**

Returns data in desired format from file

**Parameters**

- `filename` – path to file with data
- `fmt` – format of file, if it’s `None`, then it tries to guess format from `filename` extension
- `csv_encoding` – encoding for `csv` data
- `csv_delimiter` – delimiter for `csv` data

Format should be in `available_read_formats()`

1.3. API
1.3.5 cache - Cache all object’s attributes

New in version 0.1.2.

class pyautocad.cache.Cached(instance)
    Proxy for caching object attributes.

    Consider external class Foo with expensive property (we can’t change its code):

class Foo(object):
    @property
    def x(self):
        print 'consuming time'
        time.sleep(1)
        return 42

Cache all attributes and test access:

    foo = Foo()
    cached_foo = Cached(foo)
    for i in range(10):
        print cached_foo.x

Output:

    consuming time
    42
    42
    42
    42
    42

    It’s possible to switch caching off with switch_caching() and retrieve original instance with get_original()

get_original()  
    Returns original instance

switch_caching(is_enabled)  
    Switch caching on or off

    Parameters is_enabled(bool) – caching status True or False
Indices and tables

- genindex
- modindex
- search
p
pyautocad.api, 7
pyautocad.cache, 12
pyautocad.contrib.tables, 10
pyautocad.types, 8
pyautocad.utils, 9
Index

A
ACAD (in module pyautocad.api), 8
ActiveDocument (pyautocad.api.Autocad attribute), 7
aDouble() (in module pyautocad.types), 9
aDouble() (pyautocad.api.Autocad static method), 8
aInt() (in module pyautocad.types), 9
aInt() (pyautocad.api.Autocad static method), 8
APoint (class in pyautocad.types), 8
app (pyautocad.api.Autocad attribute), 7
append() (pyautocad.contrib.tables.Table method), 11
Application (pyautocad.api.Autocad attribute), 7
aShort() (in module pyautocad.types), 9
aShort() (pyautocad.api.Autocad static method), 8
Autocad (class in pyautocad.api), 7

B
best_interface() (pyautocad.api.Autocad method), 8

C
Cached (class in pyautocad.cache), 12
clear() (pyautocad.contrib.tables.Table method), 11
convert() (pyautocad.contrib.tables.Table method), 11

D
data_from_file() (pyautocad.contrib.tables.Table static method), 11
distance() (in module pyautocad.types), 9
distance_to() (pyautocad.types.APoint method), 9
doc (pyautocad.api.Autocad attribute), 7
dynamic_print() (in module pyautocad.utils), 10

F
find_one() (pyautocad.api.Autocad method), 7

G
get_original() (pyautocad.cache.Cached method), 12
get_selection() (pyautocad.api.Autocad method), 8

I
iter_layouts() (pyautocad.api.Autocad method), 7
iter_objects() (pyautocad.api.Autocad method), 7
iter_objects_fast() (pyautocad.api.Autocad method), 7

M
model (pyautocad.api.Autocad attribute), 7
mtext_to_string() (in module pyautocad.utils), 10

P
prompt() (pyautocad.api.Autocad method), 8
pyautocad.api (module), 7
pyautocad.cache (module), 12
pyautocad.contrib.tables (module), 10
pyautocad.types (module), 8
pyautocad.utils (module), 9

S
save() (pyautocad.contrib.tables.Table method), 11
string_to_mtext() (in module pyautocad.utils), 10
suppressed_regeneration_of() (in module pyautocad.utils), 9
switch_caching() (pyautocad.cache.Cached method), 12
Table (class in pyautocad.contrib.tables), 10
text_width() (in module pyautocad.utils), 10
timing() (in module pyautocad.utils), 9
to_csv() (pyautocad.contrib.tables.Table method), 11

U
unformat_mtext() (in module pyautocad.utils), 10

W
writerow() (pyautocad.contrib.tables.Table method), 11

X
x (pyautocad.types.APoint attribute), 9

Y
y (pyautocad.types.APoint attribute), 9
Z

z (pyautocad.types.APoint attribute), 9