Python AutoTask Web Services Documentation

Release 0.5.3

Matt Parr

Oct 12, 2019

Contents

Contents:

CHAPTER 1

Python AutoTask Web Services

atws is a wrapper for the AutoTask SOAP webservices API

- Free software: MIT license
- Documentation: https://atws.readthedocs.io.

1.1 Features

- Py2 and Py3 support
- Easy, programmatic query writing (no XML required)
- Query result generator retrieves all entities, not just 500
- Zone discovery (you only need a username and password)
- Picklist python module creator (IDE autofill your picklist IDs)
- Support for API v1.5 and v1.6

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Python AutoTask Web Services , run this command in your terminal:

```
$ pip install atws
```

This is the preferred method to install Python AutoTask Web Services , as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Python AutoTask Web Services can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/MattParr/python-atws
```

Or download the tarball:

```
$ curl  -OL https://github.com/MattParr/python-atws/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Python AutoTask Web Services in a project:

```
import atws
```

To enable some of the additional features that process over every entity returned by a query, the module must be imported explicitly so that it can monkeypatch the suds library. CRUD and UserdefinedFields are imported by the wrapper by default, but the others are not enabled by default.:

```
import atws.monkeypatch.attributes
```

3.1 Connecting to Autotask

- API v1.5

Only a username and password are required, but if you are initialising the library often, it may pay to also include the zone url, otherwise it needs to be discovered by performing an API lookup.:

```
at = atws.connect(username='user@usernamespace.com',password='userpassword')
```

If you have obtained an integrationcode then it must be supplied under 1.5 as well.

From api v1.6 documentation: A tracking identifier is not required to access v1.5, unless the user accessing v1.5 already has an identifier assigned. In that case, the user is required to provide their identifier.

If necessary, include the integration code in the connect parameters.:

```
at = atws.connect(username='user@usernamespace.com',
                  password='userpassword',
                  integrationcode='27-char-integration-code')
```

- API v1.6

Autotask PSA API v1.6 requires an integration code while making the connection. You must also specify the API version in the connect parameters:

```
at = atws.connect(username='user@usernamespace.com',
                  password='userpassword',
                  apiversion=1.6,
                  integrationcode='27-char-integration-code')
```

3.2 Support Files

Often, Autotask support will ask for the XML that is being sent/received in order to support a problem. Sometimes you might like to see this raw output yourself to check date conversions or entity SAX failures. There is a support file message plugin to copy XML files to a path you specify when connecting to the API.:

```
at = atws.connect(username='user@usernamespace.com',
                  password='userpassword',
                  support_file_path='/tmp')
```

3.3 Querying for entities

The Query object:

```
''' In SQL this query would be:
SELECT * FROM tickets WHERE
id > 5667
AND
(
 Status = 'Complete'
 OR
 IssueType = 'Non Work Issues'
)
'''
query = atws.Query('Ticket')
query.WHERE('id',query.GreaterThan,5667)
query.open_bracket('AND')
query.OR('Status',query.Equals,at.picklist['Ticket']['Status']['Complete'])
query.OR('IssueType',query.Equals,
         at.picklist['Ticket']['IssueType']['Non Work Issues'])
query.close_bracket()
# in ATWS XML, it would look like this
print query.pretty_print()
```

3.4 Query result cursor

The query method in the wrapper accepts the query, and returns a generator cursor which can be used to enumerate the results:

```
tickets = at.query(query)
# enumerate them
for ticket in tickets:
```

(continues on next page)

(continued from previous page)

```
    do_something(ticket)

# process them like a generator
ticket = ticket.next()

# or get a list
all_tickets = at.query(query).fetch_all()

# or if you know you are just getting one result
ticket = at.query(query).fetch_one()
```

3.5 Updating entities

Following on from the previous query result example... entities can be modified, and then returned to the API. It's best to do this using a generator function so that you can process in batches of 500 and 200. The Autotask API only gets a maximum of 500 entities per query, and can only submit 200 entities to be processed.:

```
def close_tickets(tickets):
    for ticket in tickets:
        ticket.Status = at.picklist['Ticket']['Status']['Complete']
        yield ticket


tickets = at.query(query)
# still nothing has been done
tickets_to_update = close_tickets(tickets)
# a generator cursor result again - still nothing has been done
updated_tickets = at.update(tickets_to_update)

# now the query is executed
# and then the entities are modified and resubmitted for processing
for ticket in updated_tickets:
    print ticket.id, 'was closed'

# if there were 1400 tickets in the results, then the following activity
# would take place:
# query #1 returns ticket ids 1-500
# ticket ids 1-200 are submitted for processing
# ticket ids 201-400 are submitted for processing
# query #2 returns ticket ids 501-1000
# ticket ids 401-600 are submitted for processing
##....

# if you don't need to see the results, you can just:
at.update(tickets_to_update).execute()
```

3.6 Picklists

Many entities have picklists to describe possible id values for attributes. Some common ticket entity picklist values are: Status, Priority, QueueID Looking up the picklists for an entity is an API call. There is a caching attribute on the wrapper object for accessing picklists.:

```
assert at.picklist['Ticket']['Status']['Complete'] == 5
assert at.picklist['Ticket']['Status'].reverse_lookup(5) == 'Complete'
```

Some picklists are children of parent picklists. In a ticket, Subissue type is a child of Issue type. These are handled differently due to possible naming conflicts.:

```
at.picklist['Ticket']['SubIssueType']['Hardware Failure']['Mouse']
```

In the example above, 'Hardware Failure' is an Issue Type, and 'Mouse' is a Subissue Type.

3.7 Creating entities

To create an entity, you must first create the object, and then submit it to be processed. Note that many entities have required fields.:

```
ticket = at.new('Ticket')
ticket.Title = 'test ticket'
ticket.AccountID = 0
ticket.DueDateTime = datetime.now()
ticket.Priority = at.picklist['Ticket']['Priority']['Standard']
ticket.Status = at.picklist['Ticket']['Status']['New']
ticket.QueueID = at.picklist['Ticket']['QueueID']['Your Queue Name Here']
#if you are just submitting one ticket:
ticket.create() # updates the ticket object inline using CRUD patch
# or:
new_ticket = at.create(ticket).fetch_one()

# if you are submitting many tickets, then you have the same querycursor
# options.  Process in submissions of 200 entities per API call:
tickets = at.create(new_tickets)
# or process them all at once:
tickets = at.create(new_tickets).fetch_all()
# or process them without keeping the results:
tickets = at.create(new_tickets).execute()
```

3.8 CRUD

CRUD feature to the suds objects returned in the wrapper. It supports Create, Update, Refresh, and Delete:

```
ticket = at.new('Ticket')
ticket.Title = 'Test ticket - no id yet'
assert hasattr(ticket, 'id') is False
ticket.create() # this will create the ticket in Autotask
assert ticket.id

ticket.Title = 'I changed this'
ticket.update() # this will update the ticket in Autotask
```

3.9 Userdefined Fields

Userdefined Fields are a little odd in the default suds object, so they are wrapped to provide a better interface to handle them.:

```
my_udf_value = ticket.get_udf('My Udf Name')

ticket.set_udf('My Udf Name', my_new_udf_value)
ticket.update()

# all attributes can be accessed by index
ticket_status = ticket['Status']
# if the attribute is missing, UDF will be presumed
my_udf_value = ticket['My Udf Name']
# and likewise for assignment.  if the attribute to be assigned isn't in the
SOAP specification, then a UDF will be assumed.
ticket['Status'] = at.picklist['Ticket']['Status']['Complete']
ticket['My New Userdefined Field'] = my_udf_value
ticket.update()
```

3.10 Getting Invoice Markup

Generated markup for an invoice can be fetched from ATWS by supplying invoice ID and preferred markup format (XML or HTML)

invoice_html_string = at.get_invoice_markup(3, 'html')

3.11 Additional Features

3.11.1 Attributes

3.11.2 Marshallable

3.11.3 AsDict

3.12 Advanced Example

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/MattParr/python-atws/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

Python AutoTask Web Services could always use more documentation, whether as part of the official Python AutoTask Web Services docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/MattParr/python-atws/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *atws* for local development.

1. Fork the *atws* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/python-atws.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv atws
$ cd python-atws/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 atws tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/MattParr/python-atws/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_atws
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search