ciscocucmapi

Release 0.0.2

Dec 31, 2020

Contents

CHAPTER 1

Cisco CUCM API

docs	
tests	
package	

Python Wrappers for Cisco CUCM SOAP APIs

- Free software: MIT license

1.1 Features

The ciscocucmapi package is inspired by the most excellent webexteamssdk Python API wrapper for Cisco Spark. The library wraps a python-zeep client to manage CUCM SOAP connections (specifically for AXL) and CRUD operations for common API endpoints.

1.2 Overview

- Simplified Pythonic wrappings of Cisco UC SOAP APIs
- `python-zeep`-based client under the hood - much faster than `suds`. WSDL caching is enabled by default.
- Complete abstraction of AXL SOAP API - no xml!
- **Native Python tooling includes:**

  - Native returned AXL data objects modelled with a `dict`-like interface and characteristics
  - xml order is honoured due to `OrderedDict` implementation
  - AXL crud operations supported using both Python objects and native AXL calling requirements
- Transparent sourcing of AXL credentials from local environment variables
- Easy, template-able reading and writing to JSON objects, making Cisco UC DevOps implementations a reality

1.3 Installation

```
pip install ciscocucmapi
```

You can also install the in-development version with:

```
pip install https://github.com/jonathanelscpt/ciscocucmapi/archive/master.zip
```

1.4 Documentation

https://ciscocucmapi.readthedocs.io/

1.5 Quick Start

```
from ciscocucmapi import UCMAXLConnector
import json


axl = UCMAXLConnector()  # env vars for connection params

# adding phones
ipphone_attributes = {
    "name": "SEPDEADDEADDEAD",
    "product": "Cisco 8821",
    "devicePoolName": "US_NYC_DP",
}
axl.phone.add(**ipphone_attributes)

# api endpoints can be created prior to invoking axl method-calling for pre-processing
new_bot_device = axl.phone.create()
# very useful API template development!
with open("/path/to/templates/phone.json", "w") as _:
    json.dump(axl.phone.model(), _, indent=4)

# getting existing phones with null-string dicts or lists of `returnedTags`
dead_device = axl.phone.get(name="SEPDEADDEADDEAD",
                            returnedTags={"name": "", "devicePoolName": "",
                                          "callingSearchSpaceName": ""})
beefy_device = axl.phone.get(name="SEPBEEFBEEFBEEF",
                             returnedTags=["name", "devicePoolName",
↪"callingSearchSpaceName"])
```

(continues on next page)

(continued from previous page)

```
# listing phones by name
nyc_bot_attrs = {
    "name": "BOT%",
    "devicePoolName": "US_NYC%",
    "callingSearchSpaceName": "US_%"
}
nyc_bot_devices = axl.phone.list(searchCriteria=nyc_bot_attrs,
                                 returnedTags=["name", "description", "lines"])
# implicit "return all" available for `searchCriteria` and `returnedTags`
# use sparingly for large data sets!
all_devices = axl.phone.list()

# property-like getters and setters
botuser15 = next(filter(lambda person: person.name == 'BOTUSER015', nyc_bot_devices))
botuser15.callingSearchSpaceName = "US_NYC_NATIONAL_CSS"

# updating a phone
botuser15.callingSearchSpaceName = "US_NYC_INTERNATIONAL_CSS"
botuser15.newName = "BOTJONELS"
botuser15.locationName = "Hub_None"
axl.phone.update(name=botuser15.name,
                 newName=botuser15.newName,
                 callingSearchSpaceName=botuser15.callingSearchSpaceName,
                 locationName=botuser15.locationName)

# deleting a phone
axl.phone.remove(uuid=botuser15.uuid)

# Thin AXL sql querying and execution also available
numplan = axl.sql.query("SELECT * FROM numplan")
directory_numbers = [row['dnorpattern'] for row in numplan]
numplan.csv(destination_path="/path/to/datadump/numplan.csv")  # pathlib also␣
↪supported
```

1.6 Connector Environment Variables

The following env vars are supported for ease of use:

- `AXL_USERNAME`
- `AXL_PASSWORD`
- `AXL_WSDL_URL`
- `AXL_FQDN`

1.7 AXL WSDL

The package includes the AXL wsdl for ease of use. The schema will be updated regularly to match the latest CUCM releases. By default, unless an AXL version is specified, the `current` WSDL will be used.

Due to the strictness of `python-zeep`'s WSDL and .xsd parsing, numerous AXL defects have been encountered during development and testing. As a result, the packaged WSDL and .xsd files *may* include patches to mitigate defects where applicable. Known AXL defects which have resulted in patches are catalogued in AXL_DEFECTS.rst.

If you require a more up-to-date WSDL, or are uncomfortable with using a patched schema, all `UCSOAPConnector` accept a direct path to a local WSDL file as input.

1.8 API Endpoint Support

Not all API Endpoints are supported, as API and data models are required to mitigate inconsistencies in the AXL API. If you'd like to extend API support, please create a pull request, or raise a GitHub issue and I'll add an enhancement.

I am not currently back-testing all version support, and do not intend to test against pre-9 UCM versions. The package has been developed primarily against UCM 11.5. If any API definitions interfere with the backwards compatibility of AXL for prior versions, please raise a GitHub issue and I will address this.

1.9 Supported Languages and AXL Versions

- Currently only Python 3.6+ is supported. There are no plans to support Python 2.7.
- All AXL versions *should* be supported, however only 11.5 has been currently tested. All AXL data models include static metadata on mandatory params for `add` calls. It is not expected that these should change across AXL schema versions. Please raise a defect if you encounter any issues.
- Other API methods should contain reliable schema-driven metadata and attributes.

1.10 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

1.11 Donate

If this library has helped you, or if you would like to support future development, donations are most welcome:

Cryptocurrency	Address
BTC	38c7QWggrB2HLUJZFmhAC2zh4t8C57c1ec
ETH	0x01eD3b58a07c6d005281Db76e6c1AE2bfF2226AD

1.12 Support

I'm open to discussing ad-hoc commercial support or custom DevOps implementations. Please contact me at jonathanelscpt@gmail.com for more information. Note that asking questions or reporting bugs via this e-mail address may not receive responses. Please rather create GitHub issues for this.

CHAPTER 2

Installation

At the command line:

```
pip install ciscocucmapi
```

CHAPTER 3

Usage

To use ciscocucmapi in a project:

```
import ciscocucmapi
```

CHAPTER 4

Reference

4.1 ciscocucmapi

CHAPTER 5

Contributing

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

5.1 Bug reports

When 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.

5.2 Documentation improvements

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

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/jonathanelscpt/ciscocucmapi/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 code contributions are welcome :)

5.4 Development

To set up *ciscocucmapi* for local development:

1. Fork ciscocucmapi (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:jonathanelscpt/ciscocucmapi.git
```

3. Create a branch for local development:

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

   Now you can make your changes locally.

4. When you're done making changes run all the checks and docs builder with tox one command:

```
tox
```

5. 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
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Jonathan Els - https://afterthenumber.com/

CHAPTER 7

Changelog

7.1 0.0.0 (2019-12-30)

- First alpha release.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search