Ontology Glossary Verification

Jared Stroud

Approved for Public Release; Distribution Unlimited: 18-0398.

MITRE

The Problem

- Ontologies encompass a broad spectrum of data.
- You're focused on a subset of data.
- As a developer, you should not have to learn an entire ontology to implement your focus area.

Glossary Verifier

- Verify.py ingests your tool's output which can come in the form of ttl,n3,xml,JSON-LD, and checks that your tool is compliant with the glossary used in within an ontology's RDF schema.

- **What it does:**
  - Compare's an ontology's RDF Schema to a tool's CASE/UCO output for glossary correctness.

- **What it doesn't do:**
  - Dynamically prove ontology correctness.

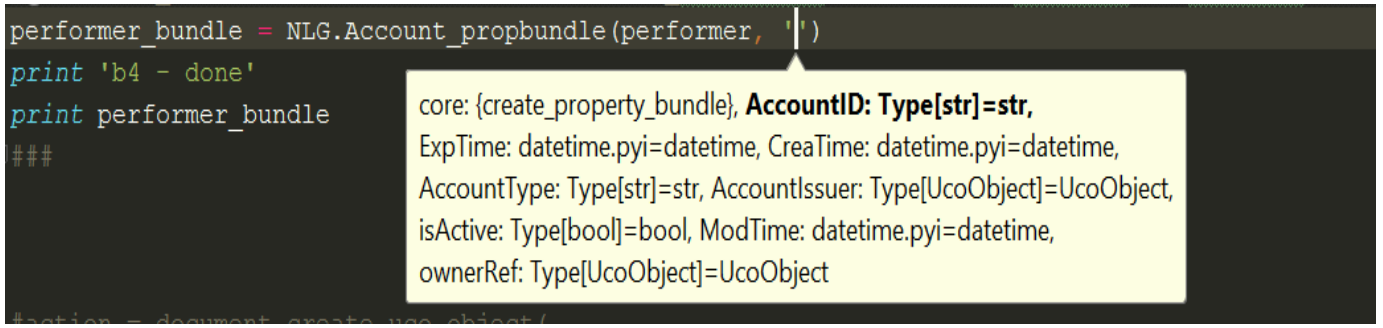
```
                          \Ontology-Verifier\src>python verify.py -g case.ttl -gf turtle -if json-ld -i output.json-ld
[+] Match!
        http://case.example.org/core#toolType in case.ttl and output.json-ld

[+] Match!
        http://case.example.org/core#endTime in case.ttl and output.json-ld

[+] Match!
        http://case.example.org/core#createdTime in case.ttl and output.json-ld

[+] Match!
        http://case.example.org/core#startTime in case.ttl and output.json-ld

[+] Match!
        http://case.example.org/core#name in case.ttl and output.json-ld

[!] http://case.example.org/core#TimeCreated not in case.ttl, but it is in output.json-ld

[+] Match!
        http://case.example.org/core#createdTime in case.ttl and output.json-ld

[+] Match!
        http://case.example.org/core#propertyBundle in case.ttl and output.json-ld

[+] Match!
        http://case.example.org/core#propertyBundle in case.ttl and output.json-ld
```


MITRE

An API to Help – NLG.py

- By performing runtime type checking of CASE/UCO objects, you can be sure that you're aware if a correct object/property bundle/etc. … being used in the CASE/UCO ontology.

- **Currently Python only.**
  Easily adoptable to other languages.
  Easy to extend upon.



IntelliSense Auto Completion

```
Traceback (most recent call last):
  File "sandbox.py", line 11, in <module>
    nlgObj = propbundle_HTTPConnection(uco, http_message_body_data_ref=cObj)
  File "C:\Users\jestroud\PycharmProjects\CASE-API\parameter_approach\NLG.py", line 1807, in propbundle_HTTPConnection
    "[propbundle_HTTPConnection] request_method is required."
AssertionError: [propbundle_HTTPConnection] request_method is required.
```

Example Scenario of Tool Usage

- A developer wants to adopt CASE/UCO, but has to implement/extend their own schema for custom {network, host, memory, etc…} artifacts. Lacking knowledge of RDF and linked-data as a whole, verify.py will ensure that what their tool has implemented conforms to their RDF schema outlined in a ttl,n3,xml,json-ld document.

- **It will not ensure their ontology is implemented correctly.** This is where NLG.py bindings come into place. With the specific asserts and type checking we ensure that the ontology format is followed.

Next Steps

- CI/CD development of CASE API through templating languages.
  - Ex: TravisCI w/ RDF ingestion engine to produce API bindings.

- SPARQL queries for specific relationship correctness.

Questions?

- **jestroud@mitre.org**

NOTICE

This software (or technical data) was produced for the U. S.
Government under contract SB-1341-14-CQ-0010, and is subject to the Rights in Data-General Clause 52.227-14, Alt. IV (DEC 2007)

© 2018 The MITRE Corporation. All rights reserved.

MITRE