ProvIt

--- IN DEVELOPMENT ---

This package aims at supporting the tracking of provenance of Python-scripted workflows as an RDF-Graph (semantic graph), according to the PROV-Ontology.

A future tool will allow for a meaningful visualization of provenance stored in a fashion this package provides.

While the ProvIt-Package tries to enable the tracking of provenance without further knowledge of semantic graph databases or the PROV-Ontology the end of this document provides some foundational concepts.

Getting started

The package assumes that a certain workflow is constructed out of a set of processes with well defined inputs and outputs. Such a process is typically written down as a method:

```
def add(x,y):
    z = x + y
    return z

a = 4
b = 5
c = add(a,b)
```

In this simple case the process is the addition of two numbers. If we use the ProvIt package in this example the code looks as follows:

```
from ProvIt import ProvenanceEngine

# define your methods
def add(x, y):
    # add two numbers
    z = x + y
    return z

# setup the provenance engine object and provide it with a namespace
# and it's abbreviation. It defaults to:
# namespace = 'https://your.project.com/example#', abbreviation = 'ex'
# every module should have it's own namespace
provEngine = ProvenanceEngine(namespace =
'https://usefulmodulecollection.org/modul1#', abbreviation = 'm1')


a = 4
# the crucial part of the sematic graph are the unique IDs of every node.
# These IDs are returned as you call the add*()-methods:
aID = provEngine.addEntity()
```

```
b = 5
# you can and should provide Entities with a human-readable label and a
# detailed description. If no label is provided it defaults to the global
# identifier, which isn't meaningful (it's just unique).
bID = provEngine.addEntity(
    label = 'Variable B',
    description = "it's value is: " + str(b)
)


c = add(a,b)
# add the result as entity.
cID = provEngine.addEntity('Variable C', 'A + B = ' + str(c))


# add the process to the graph
procID = provEngine.addProcess(
    label = 'add',
    description = 'adds two numbers and gives out a sum'
)

# finally connect the nodes
provEngine.relateProcessAndEntities(
    inputIDs = [aID,bID],
    outputIDs = cID,
    processID = procID
)

# save the graph in the Turtle format
provEngine.serialize('module1.ttl')
```

It seems quite a lot to type. But while the amount of lines per method can scale indefinitely, the amount of line to track the provenance of this method stays pretty constant. The above code would generate a semantic graph which (in a simplified version) looks like this: