pykb Documentation

Release 1.5

Séverin Lemaignan et al.

November 04, 2014

Contents

1 Package documentation **1**
1.1 kb module 1

2 Examples **5**

Python Module Index **7**

Package documentation

1.1 kb module

class kb.**EmbeddedKBClient**(*defaultontology=None*)

__init__(*defaultontology=None*)

__module__ = 'kb'

call_server(*method*, **args*, ***kwargs*)

close()

kb = None

kb_thread = None

kb_users = 0

process(*defaultontology*)

sendmsg(*msg*)

class kb.**EventCallbackExecutor**(*in_event_queue*, *out_polled_event_queue*, *callback_queue*)
Bases: threading.Thread

__init__(*in_event_queue*, *out_polled_event_queue*, *callback_queue*)

__module__ = 'kb'

close()

run()

class kb.**KB**(*host='localhost'*, *port=6969*, *embedded=False*, *defaultontology=None*, *sock=None*)

__contains__(*pattern*)
This will return 'True' is either a concept - described by its ID or label- or a statement or a set of statement is present (or can be infered) in the ontology.

This allows syntax like:

```
if 'Toto' in kb:
    #...
if 'toto sees tata' in kb:
    #...
```

__del__()

__enter__()

__exit__(*exc_type*, *exc_val*, *exc_tb*)

__getitem__(**args*)

This method introduces a different way of querying the ontology server. It uses the args (be it a string or a set of strings) to find concepts that match the pattern. An optional 'models' parameter can be given to specify the list of models the query is executed on.

Depending on the argument, 4 differents behaviours are possible:

- with a string that can not be lexically split into 3 tokens (ie, a string that do not look like a `s p o` tuple), a lookup is performed, and matching resource are returned
- **with a single `s p o` pattern:**
  - if only one of s, p, o is an unbound variable, returns the list of resources matching this pattern.
  - if 2 or 3 of the tokens are unbound variables (like `kb["* * *"]` or `kb["* rdf:type *"]`), a list of statements matching the pattern is returned.
- with a list of patterns, a list of dictionaries is returned with possible combination of values for the different variables. For instance, `kb[["?agent desires ?action", "?action rdf:type Jump"]]` would return something like: `[{"agent":"james", "action": "jumpHigh"}, {"agent": "laurel", "action":"jumpHigher"}]`

Attention: if more than one argument is passed, and if the last argument is a list, this list is used as the set of models to execute the query on. If not such list is provided, the query is executed on all models.

Use example:

```
import kb

kb = KB()

for agent in kb["* rdf:type Agent"]:
    #...

if kb["* livesIn ?house", "?house isIn toulouse", ['GERALD']]:
    #...

#Assuming 'toulouse' has label "ville rose":
city_id = kb["ville rose"]
```

__iadd__(*stmts*)

This method allows to easily add new statements to the ontology with the += operator. It can only add statement to the default robot's model (other agents' model are not accessible).

```
kb = KB(<host>, <port>)
kb += "toto likes icecream"
kb += ["toto loves tata", "tata rdf:type Robot"]
```

__init__(*host='localhost'*, *port=6969*, *embedded=False*, *defaultontology=None*, *sock=None*)

__isub__(*stmts*)

This method allows to easily retract statements from the ontology with the −= operator. It can only add statement to the robot's model (other agents' model are not accessible). If a statement doesn't exist, it is silently skipped.

```
kb = KB(<host>, <port>)
kb -= "toto likes icecream"
kb -= ["toto loves tata", "tata rdf:type Robot"]
```

`__module__` = 'kb'

`add_method`(*m*)

`close`()

`subscribe`(*pattern*, *callback=None*, *var=None*, *type='NEW_INSTANCE'*, *trigger='ON_TRUE'*, *models=None*)

Allows to subscribe to an event, and get notified when the event is triggered.

Example with callbacks:

```
>>> def onevent(evt):
>>>     print("In callback. Got evt %s" % evt)
>>>
>>> self.kb.subscribe(["?o isIn room"], onevent)
>>> self.kb += ["alfred isIn room"]
>>> # 'onevent' get called
In callback. Got evt [u'alfred']
```

Example with 'polled' events:

```
>>> evt_id = self.kb.subscribe(["?o isIn room"])
>>> self.kb += ["alfred isIn room"]
>>> print(str(self.kb.events.get()))
('evt_7694742461071211105', [u'alfred'])
```

If 'callback' is provided, the callback will be invoked with the result of the event (content depend on event type) *in a separate thread*. If not callback is provided, the incoming events are stored in the KB.events queue, and you can poll them yourself (which allow for better control of the execution flow).

The 'var' parameter can be used with the 'NEW_INSTANCE' type of event to tell which variable must be returned.

The 'models' parameter allows for registering an event in a specific list of models. By default, the pattern is monitored on every models.

Returns the event id of the newly created event.

exception `kb.`**`KbError`**(*value*)

Bases: `exceptions.Exception`

`__init__`(*value*)

`__module__` = 'kb'

`__str__`()

`__weakref__`

list of weak references to the object (if defined)

class `kb.`**`NullHandler`**(*level=0*)

Bases: `logging.Handler`

Defines a NullHandler for logging, in case kb is used in an application that doesn't use logging.

Initializes the instance - basically setting the formatter to None and the filter list to empty.

`__module__` = 'kb'

`emit`(*record*)

class `kb.`**`RemoteKBClient`**(*event_queue*, *map*, *host='localhost'*, *port=6969*, *sock=None*)

Bases: `asynchat.async_chat`

`__init__`(*event_queue*, *map*, *host='localhost'*, *port=6969*, *sock=None*)

`__module__` **= 'kb'**

`call_server`(*method*, **args*, ***kwargs*)

`collect_incoming_data`(*data*)

`decode`(*raw*)

`encode`(*method*, **args*, ***kwargs*)

`found_terminator`()

`handle_error`()

`initiate_send`()

`use_encoding` **= 0**

Examples

```
import kb
import time

REASONING_DELAY = 0.2

def onevent(evt):
    print("Something happened! %s" % evt)

with kb.KB() as kb:

    kb += ["alfred rdf:type Human", "alfred likes icecream"]

    if 'alfred' in kb:
        print("Hello Alfred!")

    if 'alfred likes icecream' in kb:
        print("Oh, you like icrecreams?")

    kb -= ["alfred likes icecream"]

    if 'alfred likes *' not in kb:
        print("You don't like anything? what a pity...")

    kb += ["Human rdfs:subClassOf Animal"]
    time.sleep(REASONING_DELAY) # give some time to the reasoner

    if 'alfred rdf:type Animal' in kb:
        print("I knew it!")

    for facts in kb.about("Human"):
        print(facts)

    for known_human in kb["?human rdt:type Human"]:
        print(known_human)


    kb += ["alfred desires jump", "alfred desires oil"]
    kb += ["jump rdf:type Action"]

    for action_lover in kb["?agent desires ?obj", "?obj rdf:type Action"]:
        print(action_lover)

    kb.subscribe(["?agent isIn ?loc", "?loc rdf:type Room"], onevent)
```

```
kb += ["alfred isIn sleepingroom", "sleepingroom rdf:type Room"]

time.sleep(1) # event should have been triggered!
```

Python Module Index

k

`kb`, 1

Symbols

__contains__() (kb.KB method), 1
__del__() (kb.KB method), 1
__enter__() (kb.KB method), 2
__exit__() (kb.KB method), 2
__getitem__() (kb.KB method), 2
__iadd__() (kb.KB method), 2
__init__() (kb.EmbeddedKBClient method), 1
__init__() (kb.EventCallbackExecutor method), 1
__init__() (kb.KB method), 2
__init__() (kb.KbError method), 3
__init__() (kb.RemoteKBClient method), 4
__isub__() (kb.KB method), 2
__module__ (kb.EmbeddedKBClient attribute), 1
__module__ (kb.EventCallbackExecutor attribute), 1
__module__ (kb.KB attribute), 3
__module__ (kb.KbError attribute), 3
__module__ (kb.NullHandler attribute), 3
__module__ (kb.RemoteKBClient attribute), 4
__str__() (kb.KbError method), 3
__weakref__ (kb.KbError attribute), 3

A

add_method() (kb.KB method), 3

C

call_server() (kb.EmbeddedKBClient method), 1
call_server() (kb.RemoteKBClient method), 4
close() (kb.EmbeddedKBClient method), 1
close() (kb.EventCallbackExecutor method), 1
close() (kb.KB method), 3
collect_incoming_data() (kb.RemoteKBClient method), 4

D

decode() (kb.RemoteKBClient method), 4

E

EmbeddedKBClient (class in kb), 1
emit() (kb.NullHandler method), 3
encode() (kb.RemoteKBClient method), 4
EventCallbackExecutor (class in kb), 1

F

found_terminator() (kb.RemoteKBClient method), 4

H

handle_error() (kb.RemoteKBClient method), 4

I

initiate_send() (kb.RemoteKBClient method), 4

K

KB (class in kb), 1
kb (kb.EmbeddedKBClient attribute), 1
kb (module), 1
kb_thread (kb.EmbeddedKBClient attribute), 1
kb_users (kb.EmbeddedKBClient attribute), 1
KbError, 3

N

NullHandler (class in kb), 3

P

process() (kb.EmbeddedKBClient method), 1

R

RemoteKBClient (class in kb), 3
run() (kb.EventCallbackExecutor method), 1

S

sendmsg() (kb.EmbeddedKBClient method), 1
subscribe() (kb.KB method), 3

U

use_encoding (kb.RemoteKBClient attribute), 4