Pyramid Cubicweb Documentation

Release 0.2.0

Christophe de Vienne

May 22, 2015

Contents

CHAPTER 1

Summary

Easily build a webservice API on top of a cubic web database.

Once activated, the cube provides new controllers (their regid is *webservices*) which respond on the rest path of entities, if the requests matches one of the following condition:

- Content-Type == 'application/json'
- Accept == 'application/json'

The following API is automatically provided for all the entities:

HTTP	action
GET /etype?filter=xxx	return a list of entities matching the filter
POST /etype	create a new entity
GET /etype/1234	return a particular entity
PUT /etype/1234	update an entity
DELETE /etype/1234	delete an entity

1.1 Getting Started

- Install the cube:

```
pip install cubicweb-wsme
```

- Activate the cube on your instance:

```
add_cube("wsme")
```

  Your instance now provides webservices

- Call your APIs, for example from javascript (using jQuery):

```
$.ajax({
    url: 'http://localhost:8080/cwuser',
    dataType: 'json',
    data: {
        // get users which name starts with "dupon"
        filter: JSON.stringify({
            'surname': {
                '$ilike': 'dupon%'
            }
        },
        // retrieve the user groups references
        fetch: [
            'in_group'
        ],
        // order by login DESC
```

```
            orderby: '-login'
        },
        traditional: true,
        success: function (data) {
            // data is a list looking like this:
            // [{eid: 000, login: "xx", ...., in_group: [{eid: 0, modification_date: 'xxx'}, {ei
            console.log("Got", data, "!");
        }
    });
```

1.2 Content

1.2.1 Types

The default mapping of an entity can be overriden. Below is an example with CWUser and CWGroup.

Note: The webservice types are not autoregistered by the registry, hence an explicit registering is needed. The *cubes.wsme.types.scan()* function makes things easier if many types are defined in the same module.

```
from cubes.wsme.types import Base, wsattr, scan

class CWUser(Base):
    login = wsattr('login', datatype=wsme.types.text)
    password = wsattr('upassword', datatype=wsme.types.text)

    in_group = wsattr('in_group', datatype=['CWGroup'])

class CWGroup(Base):
    name = wsattr('name', datatype=wsme.types.text)

    users = wsattr('in_group', role='subject', datatype=[CWUser])


def register_callback(vreg):
    scan(vreg, __name__)
```

1.2.2 Query

Filter format

The filter format is partially inspired by https://www.parse.com/docs/rest#queries

operators

Key	Operation
`$lt`	Less Than
`$lte`	Less Than Or Equal To
`$gt`	Greater Than
`$gte`	Greater Than Or Equal To
`$ne`	Not Equal To
`$in`	Contained In
`$nin`	Not Contained in
`$or`	Or
`$and`	And

Filter attribute

Exact match:

```
{'attrname': value}
```

Other comparisons:

```
{'attrname': {'$op': value, '$op2': othervalue}}
```

Use and/or:

```
{'$or': {'attrname': value, 'attr2name': value}}
{'$or': [
    {'attrname': value},
    {'attrname': {
        '$in': [1, 2, 3]}}}]}
```

Filter relations

If comparing by eid, same as attribute

Exact match:

```
{'relname': eid}
```

Other:

```
{"relname": {"$op": eid}}
```

Filter on relation target attributes/relations:

```
{"relname": <entity filter>}

{"relname": {"attrname": value}}

{"relname": {"$or": {"attrname": value, "attr2name": ovalue}}}
```

1.2.3 API

`cubes.wsme.types`

Types

class `cubes.wsme.types.`**`PassThroughType`**

Special webservice type that transmit a value without doing anything

It is used in wsme signature for the ‘entity’ argument which is passed by the controller itself and should not be transtyped in any way.

`cubes.wsme.types.`**`JsonData`** **= <cubes.wsme.types.JsonDataType object>**

User type that carry json encoded arbitrary data.

`cubes.wsme.types.`**`binary`** **= <cubes.wsme.types.BinaryType object>**

webservice type that map the `cubicweb.Binary` values.

class `cubes.wsme.types.`**`wsattr`**(*rtype=None*, *role=’subject’*, *etype=None*, *datatype=None*, *writeonly=False*, ***kw*)

Cubicweb-specific version of `wsme.types.wsattr`

To be used on a *`Base`* class. All the attributes are optionnal and can be deduced from the *`rtype`*.

rtype
The corresponding relation in the model

role
The role of the parent class in the relation

etype
The entity type on the other side of the relation

datatype
The webservice type

`cubes.wsme.types.`**`iswsattr`**(*obj*)
returns True if an object is a *`wsattr`*

class `cubes.wsme.types.`**`Base`**(*entity=None*, *keyonly=False*, *fetch=()*)
Bases: `wsme.types.Base`

Base class for a complex type that map an entity type

eid
Entity eid

alias of `long`

final_values()
Returns a dict with all the attribute values.

This dict can be used to feed `cubicweb.entity.Entity.cw_set()`.

classmethod **finalize_init**()
Finalize the class initialization.

This last step resolve types in the underlying attributes.

from_entity(*entity*, *keyonly=False*, *fetch=()*)
Load values from an entity

Parameters

- **entity** – the entity
- **keyonly** – if *True*, only the .eid and .modfication_date will be loaded. The result can be used as “timestamped reference”.
- **fetch** – a list of relations to eager load. Unless specified, all the ‘1’ or ‘?’ relation targets will be loaded as ‘keyonly’, and the ‘*’ or ‘+’ relations will not be loaded at all.

classmethod **reginit**(*vreg*)
Register the class

Use the informations in the registry, and most notably the schema, to initialize the attributes.

to_entity(*entity*)
Update the entity attributes (not the non-final relations).

class `cubes.wsme.types.`**`Any`**(*entity=None*, *keyonly=False*, *fetch=()*)
Complex type to carry any type of entity.

Automatically used for polymorphic relations targets

`cubes.wsme.types.`**`scan`**(*vreg*, *modname*)
Scan a module for any class inheriting *`Base`* and register them.

cubes.wsme.controller

‘webservice’ controller implementation

class `cubes.wsme.controller.`**`WSController`**(**args*, ***kwargs*)
Bases: `cubicweb.web.controller.Controller`

A controller that rely on WSME to provide webservice API for an entity.

`publish`(*rset*)
Main entry-point of the controller.

Will dispatch the request to the adequate function depending on the http method and the form/rset content.

It also takes care of converting the inputs (form & body) to call arguments using the WSME api, based on the function signatures.

The following *form* values are used, which are normaly set by `cubes.wsme.views.RestPathEvaluator`:

- *_ws_method*: the HTTP method
- *_ws_etype*: the etype (ignored, only used by the selector)
- *_ws_rtype*: the relation type if provided
- *_ws_rtype_target*: An option relation target id

If the **:arg:‘rset‘** contains an entity, it will be considered as the target of the API call.

classmethod **`resolve_types`**(*registry*)
Late-resolve the types of the function signatures.

This function is called at regitering time (by `__registered__()`).

class `cubes.wsme.controller.`**`WSCRUDController`**(**args*, ***kwargs*)
Bases: *`cubes.wsme.controller.WSController`*

An entity type CRUD controller

The displatch is summarized in this table, where ‘entity’ means that an entity exists in the rset:

form-rset / verb	GET	POST	PUT	DELETE
	`_get()`	*`_post()`*		
entity	*`_entity_get()`*		*`_entity_put()`*	*`_entity_delete()`*
entity, _ws_rtype	*`_entity_rtype_get()`*	*`_entity_rtype_post()`*		
entity, _ws_rtype, _ws_rtype_target				*`_entity_rtype_target_delete()`*

`_create`(*data*)
Create an entity from ws data

`_entity_delete`(*entity*)
Default implementation of *DELETE /etype/eid*.

`_entity_get`(*entity*, *fetch=[]*)
Default implementation of *GET /etype/eid*.

`_entity_put`(*entity*, *fetch=[]*, *data=None*)
Default implementation of *PUT /etype/eid*.

`_entity_rtype_get`(*entity*, *rtype*, *orderby=None*, *limit=None*, *offset=None*, *keyonly=False*)
Default implementation of *GET /etype/eid/rtype*.

`_entity_rtype_post`(*entity*, *rtype*, *eid*)
Default implementation of *POST /etype/eid/rtype*.

`_entity_rtype_target_delete`(*entity*, *rtype*, *eid*)
Default implementation of *DELETE /etype/eid/rtype/eid*.

`_get`(*orderby=None*, *filter=None*, *limit=0*, *offset=0*, *fetch=[]*, *keyonly=False*)
List entities with an optional filter.

Default implementation of *GET /etype*.

Parameters

- **`filter`** –
- **`fetch`** – A list of relations and subrelations of which the target entities will be returned.

`_get_entities` (*datalist*)
Get a list of entities from a list a webservice data

`_get_entity` (*data*)
Get an entity and update/create it and its related entities all along.

Parameters **`data`** – A webservice type instance

`_handle_data` (*data*)
Handle webservice data.

It returns a tuple *(eid, values, relation_values)*, where eid can be None if the data had none, *values* contains the final and inlined values, and *relation_values* the relation values. These variables are dictionnaries that can be fed cw_set().

While handling the entity data, the related entities present in the data will be updated/create (via *`_get_entity()`*).

`_post` (*fetch=[]*, *keyonly=False*, *data=None*)
Default implementation of *POST /etype*.

`_update` (*data*)
Update an existing entity from ws data

1.3 Change History

1.4 Indices and tables

- genindex
- modindex
- search

Python Module Index