metadataStore Documentation

Release .1

Arman Arkilic

September 22, 2014

Contents

CHAPTER 1

Contact Information

author: *Arman Arkilic*

contact: arkilic@bnl.gov

Brookhaven National Laboratory

National Synchrotron Light Source-II

CHAPTER 2

Contents

2.1 Overview

2.1.1 Purpose

Advanced techniques for a new generation of hardware will drive higher data rates for experiments at NSLS2 beamlines. The large data volumes and data acquisition rates require high-performance logging and storage systems. A typical experiment involves not only the raw data from detectors, but also requires additional data from the beamline such as experiment owner, beamline_id, energy, motor positions, wavelength, etc... To date, this information is largely held separately and manipulated individually. metadataStore is a service that is used in order to record this sort of metadata in beamline experiments. It is designed and implemented with the needs of various experiments in mind, therefore it is flexible enough to satisfy the needs of different experimental setups.

metadataStore can be used independently or embedded with the dataBroker, which utilizes an integrated approach that blends different data resources together and makes them available for data analysis and visualization clients.

2.1.2 Technology

metadataStore backend database is mongoDb, a No-SQL database that was chosen due to its performance and flexibility. Python(v 2.7) is chosen as language of implementation. mongoDb Python driver(pymongo) is used to perform database related operations.

2.1.3 Downloading/Installing

Requirements

Python(version 2.7.X), pymongo (version 2.6+), six, Distutils, Git

Installation

MongoDb Installation

Step 1:

```
% sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
```

Step 2:

```
% sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 7F0CEB10
```

Step 3:

```
% sudo apt-get update
```

Step 4:

```
% sudo apt-get install -y mongodb-org
```

Step 5:

```
% sudo apt-get install -y mongodb-org=2.6.1 mongodb-org-server=2.6.1 mongodb-org-shell=2.6.1 mongodb-
```

metadataStore Installation

Step 1:

metadataStore is available via git repository: https://github.com/arkilic/metadataStore

Clone this repository:

```
%git clone https://github.com/arkilic/metadataStore
```

Step 2:

metadataStore includes a setup.py script. Distutils, building and installing a module distribution using the Distutils is one simple command to run from a terminal:

```
% python setup.py install
```

If user does not have sudo access to the machine and/or does not want to install this package for all users:

```
% python setup.py install --user
```

2.1.4 Getting Help

metadataStore can be embedded within other applications or used interactively. If used within IPython, 'help' keyword provides information regarding routines:

```
% from metadataStore.userapi.commands import search
% help(search)
% search(scan_id=None, owner=None, start_time=None, beamline_id=None, end_time=None, data=False, hea
%
% Provides an easy way to search Header entries inserted in metadataStore
% Usage:
%   search(scan_id=s_id)
%   search(scan_id=s_id, owner='ark*')
%   search(scan_id=s_id, start_time=datetime.datetime(2014, 4, 5))
%   search(scan_id=s_id, start_time=datetime.datetime(2014, 4, 5), owner='arkilic')
%   search(scan_id=s_id, start_time=datetime.datetime(2014, 4, 5), owner='ark*')
%   search(scan_id=s_id, start_time=datetime.datetime(2014, 4, 5), owner='arkili.')
```

2.1.5 Schema

metadataStore service consists of four collections: header, beamline_config, event_descriptor and event

A Sample Header:

```
%{'status': 'In Progress',
% 'beamline_id': None,
% 'tags': ['CSX_Experiment1', 'CSX_Experiment2'],
% 'start_time': datetime.datetime(2014, 9, 16, 13, 7, 58, 299000),
% 'scan_id': 3315, 'custom': {},
% 'owner': u'arkilic',
% 'end_time': datetime.datetime(2014, 9, 16, 13, 7, 58, 299000),
% 'header_versions': [],
% 'event_descriptors': {'event_descriptor_0': {'data_keys': ['motor5', 'motor4', 'motor3', 'list_of_1
%                                              'tag': 'experimental', 'descriptor_name': 'scan',
%                                              'header_id': ObjectId('5418362efa44833ca9b08d08'),
%                                              'event_type_id': 12, '_id': ObjectId('5418362efa44833c
%                                              'events': {
%                                                        'event_0': {'descriptor_id': ObjectId('5418
%                                                                    'description': None,
%                                                                    'header_id': ObjectId('5418362e
%                                                                    'seq_no': 3,
%                                                                    'owner': 'arkilic',
%                                                                    '_id': ObjectId('5418362efa4483
%                                                                    'data': {u'motor5': 36, u'motor
%                                                        'event_1': {'descriptor_id': ObjectId('5418
%                                                                    'description': None,
%                                                                    'header_id': ObjectId('5418362e
%                                                                    'seq_no': 1,
%                                                                    'owner': u'arkilic',
%                                                                    '_id': ObjectId('5418362efa4483
%                                                                    'data': {}}}, u'type_descriptor
% 'configs': {'config_0': {'header_id': ObjectId('5418362efa44833ca9b08d08'),
%                          '_id': ObjectId('5418362efa44833ca9b08d09'),
%                          'config_params': {}}},
%                          '_id': ObjectId('5418362efa44833ca9b08d08')}}
% }
```

Header

Each run stands for a specific set of operations performed by data collection routines. An experiment is a collection of these runs. metadataStore service does not group set of specific runs as experiments. This is done within dataBroker service(currently in development). Users/developers that use metadataStore can either use dataBroker API or develop simple applications on top of metadataStore that keep track of set of scan_id(s) to define specific experiments.

Each header consists of the following keys:

_id: (bson.ObjectId) primary key for header entry

scan_id: user/data collection defined unique identifier describing a given run

start_time: (datetime) run header initialization timestamp

end_time: (datetime) run header close timestamp

tag: (list) list of strings that assigns a user/data collection defined tag to each header

owner: (str) data collection or system defined user info

header_versions: (list) Keeps track of header version

beamline_id: (str) descriptor for beamline

custom: (dict) Field for custom information

configs: (dict) Field that holds information regarding beamline configurations

event_descriptors: (dict) Field that holds event definitions and events

Beamline Config

The 'configs' key inside each header holds name-value pairs and/or nested dictionaries with name-value pairs for a given run.

Beamline Config Keys:

_id: (bson.ObjectId) Unique identifier for entry in beamline_config collection.

header_id: (bson.ObjectId) foreign key pointing back to header.

config_params: (dict) data collection/user scripts can populate this dictionary with information of their choice.

Event Descriptor

Data related to a given run is captured inside 'events'. Event descriptors serve as event containers and define the contents of each event. In other words, event descriptor holds the actual run metadata under 'events' key.

Event Descriptor Keys:

_id: (bson.ObjectId) Unique identifier for entry in event_descriptor collection.

header_id : (bson.ObjectId) foreign key pointing back to header.

event_type_id: (int) event type integer descriptor generated by data collection routines.

tag: (list) list of strings that assigns a user/data collection defined tag to each event descriptor

descriptor_name: (str) event type string descriptor

type_descriptor: (dict) defines fields and field data types for a given event type

Event

Metadata for each run is captured inside events. A set of events that belong to a run header can be accessed within event descriptors as described above. Data analysis/visualization tools that want to access metadata can do this by parsing run header. For the example above:

```
> from metadataStore.userapi.commands import search
> headers = search(owner='arkilic', data=True)
>
> my_event_0 = headers['header_0']['event_descriptors']['event_descriptor_0']['event_0']
>
>'event_0': {'descriptor_id': ObjectId('5418362efa44833ca9b08d0a'),
>                                                                    'description': None,
>                                                                    'header_id': ObjectId('5418362e
>                                                                    'seq_no': 3,
>                                                                    'owner': 'arkilic',
>                                                                    '_id': ObjectId('5418362efa4483
>                                                                    'data': {u'motor5': 36, u'motor
```

There are also series of utility libraries that allow manipulation of query results. See utilities section for more information.

Event Keys:

header_id: (bson.ObjectId) foreign key pointing back to header

descriptor_id: (bson.ObjectId) foreign key pointing to event descriptor

seq_no: (int) defines an event's order in an event descriptor.(order in a set of events within descriptor)

owner: (str) event owner

description: (str) string field that describes an event(optional)

data: (dict) stores the metadata as a python dictionary

2.1.6 Tutorial

Start here for a quick overview

2.1.7 Examples

Examples of how to perform specific tasks

2.2 Library Reference

2.2.1 Collection Declarations

Provides information regarding MongoDB Collection template creation. This is analogous to table definitions in SQL databases

Header

EventDescriptor

Event

BeamlineConfig

2.2.2 Collection API

Provides routines for data collection libraries to populate metadataStore

2.2.3 User API

Provides routines for data collection libraries to populate metadataStore

2.2.4 Data API

Includes the raw commands. Developers/expert users can create a set of new behaviors using this module.

2.2.5 Utiliy Library

metadataStore.utilities.utility.**get_calib_dict**(*run_header*)
Return the calibration dictionary that is saved in the run_header

run_header [dict] Run header to convert events to lists. Can only be one header.

dict Dictionary that contains all information inside the run_header's beamline_config key. If there are multiple 'configs' then the return value is a nested dictionary keyed on config_# for the number of config dicts

bool True: Multiple 'config' sections were present, dict is a nested dict False: One 'config' section was present, dict is not a nested dict

metadataStore.utilities.utility.**get_data_keys**(*run_header*)
Return the names of the data keys. This function assumes that there is only one event descriptor in a run header

run_header [dict] Run header to convert events to lists. Can only be one header.

list List of the data keys in the run header keyed by the event_descriptor name

metadataStore.utilities.utility.**get_data_keys_futureproof**(*run_header*)
Return the names of the data keys. This function assumes that there is only one event descriptor in a run header

Note: This function only works for one event_descriptor per run_header. As soon as we start getting multiple event_descriptors per run header this function will need to be modified

run_header [dict] Run header to convert events to lists. Can only be one header.

dict List of the data keys in the run header keyed by the event_descriptor descriptor_name. If descriptor_name is not unique for all event_descriptors, see Notes 1

1. the descriptor_name field in event_descriptors is assumed to be unique. If it is not, then append the last four characters of _id to it. Ideally this would be something more like a PV name and less hostile than the hashed _id field

metadataStore.utilities.utility.**listify**(*run_header*, *data_keys=None*, *bash_to_lower=True*)
Transpose the events into lists

from this: run_header : {

"event_0_data" : {"key1": "val1", "key2": "val2", ...}, "event_1_data" : {"key1": "val1", "key2": "val2", ...}, ... }

to this: {"key1": [val1, val2, ...],

"key2" = [val1, val2, ...], "keyN" = [val1, val2, ...], "time" = [time1, time2, ...],

}

run_header [dict] Run header to convert events to lists

event_descriptors_key [str] Name of the event_descriptor

data_keys [hashable or list, optional]

- If data_key is a valid key for an event_data entry, turn that event_data into a list
- If data_key is a list of valid keys for event_data entries, turn those event_data keys into lists
- If data_key is None, turn all event data keys into lists

bash_to_lower [boolean] True: Compare strings after casting to lowercase False: Compare strings without casting to lowercase

dict data is keyed on run header data keys with an extra field 'time'

2.3 Tutorials

2.3.1 User API Tutorial

Does not have bulk insert on header or event create uses parameter fields rather than dictionaries No update/delete allowed

2.3.2 Collection API Tutorial

Has bulk insert on header and event create Uses dictionaries to in order to create entries No update/delete allowed

2.3.3 Utility Tools Tutorial

Provides ways to manipulate the header data structure

2.4 VisTrails Integration

Quickstart for VisTrails intallation and overview of metadataStore related tools

2.4.1 Download/Installation

Installation instructions for vistrails

2.4.2 metadataStore Tools in Vistrails

Data parsing and visualization tools

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*

m