decisionengine

Release 1.1.1

Fermi Research Alliance, LLC.

May 14, 2020

CONTENTS

The Decision Engine is a critical component of the HEP Cloud Facility. It provides the functionality of resource scheduling for disparate resource providers, including those which may have a cost or a restricted allocation of cycles

CHAPTER

ONE

RELEASE NOTES

1.1 Release 1.1.0

In this release:

- Fixed. https://github.com/HEPCloud/decisionengine_modules/issues/108 "Supply Postgres script to delete fields in main database before a certain date"
- significant code cleanup and pep8 compliance
- unit test work
- CI (GitHub actions and Travis) is introduced

commits

f894b1d : Skip unittest (#77)

632e64b : Add ipython

f681a79 : Make python 2.7 tests run on 1.1 branch

d6a32c0 : implementation of data reaper (#75)

2ad8614 : Use sparse checkout for first checkout to get .github/actions (#65)

812f032 : Cat output of pytest log Exit pylint entrypoint with the line count of pep8 and pylint logs Deal with (detach from . . .) Only tar up (S)RPMS dirs for rpm build.

6b05ec7 : Fix errors reported by run_pylint (#62)

d9f5b66 : Setup pep8speaks

c3b8ac2 : Run github actions as non-root uid. Install packages in virtualenv and remove system rpms.

ae01f9e : Support Python 3 for Boost Python

579761c : Support Python 3 for Boost Python

044b979 : Remove unnecessary using declarations.

00f6d00 : Add extra header dependency due to Boost Python ommission.

24e0795 : Apply clang-format

17c17f9 : Remove JSON dependency.

faa0b22 : Massive cleanup.

07b555f : Updates to Github Actions to allow building with python3.6

fef6c11 : Fix errors when running pylint.sh multiple times da6f077 : Autopep8 -i fixes

39fe5b3 : TaskManager: fix calling log_exception with correct number of arguments and minor format changes to reduce PEP8 warnings

17396da : logicengine: get rid of compuler warnings

01dc3d1 : Only track what we need

b609d73 : Configure coveralls (and some minor cleanup)

bd9ed5e : Many C++ cleanups

2a61876 : Add Badges

c864f27 : Do not call pytest fixtures directly.

307db5f : white space fix

882b58f : fix unit tests

1da687c : Replace Boost facilities with C++ STL ones.

5a6e6b1 : Run tests on push

8404245 : Add missing Boost regex library dependency.

ceb5fe7 : Apply clang-format to files that were missed earlier.

3de9940 : Apply clang-format to C++ code.

8a8f560 : Cache venv directory instead

ad017ce : Build private boost for testing

928c64a : Test pip cache

358939a : Adjust CMakeLists.txt files to use correct Python versions

9f0ddb3 : Add pylint github action.

5e6ce4a : Remove more unused C++ files.

63717fe : Setup travis to use new cmake var

74fab2a : Use cmake argumement -DPYVER=3.6 to build python3 library https://fermicloud140.fnal.gov/reviews/r/31/

843f30c : Minor cleanups per travis-lint

a538cac : Remove unused C++ files.

4c9d125 : Update repo where action is taken from

87fb2d9 : Update rpms installed in docker image. Update entrypoint.sh to use cmake3.

199ee87 : Find python3 libraries using cmake3 from epel rpm Also need to install python3-devel

4c79d2c : Remove unnused GNUmakefiles.

94342ee : Add unit test as a Github Action

1a0e102 : more advanced travis.yml

0be413f : Add helper file for pip

7794327 : Make recursive import happy

7005c78 : Add simple target

de8b0fa : python3 compliance: replace string.join() where appropriate, handle UserDict

2662e6c : note required packages

3b87119 : Add missing header includes.

3e79b84 : Remove defunct code and its tests

b1dbe1a : Ensure attribs are defined at **init**

c4ad78a : Correct logger arguments do avoid duplicate string parse

a8dcc67 : Remove unused imports (per pylint)

d3502b5 : Remove obsolete CVS directories.

d744111 : add six module to the list of required modules

0a9b1e8 : Fix class declaration

b83157e : Handle metaclasses

549f33b : Add config for Travis CI

ee71044 : Drop trailing white space

3f82af6 : Python3 forward compatible syntax

28bf291 : Add safe (for python 2.7) python3 compatible syntax

1d1d76f : prepare for python3

CHAPTER TWO

DEVELOPER DOCUMENTATION

First command `cd` is just to make sure that you end up in a directory that will contain two subdirectory `decisionengine` and `decisionengine_modules`. Of course this can be done in any directory, not necessarily home directory.

2.1 Decisionengine framework

2.1.1 Prerequisites:

```
yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/
↪pgdg-redhat-repo-latest.noarch.rpm
yum install -y python3 python3-pip cmake3 boost-devel python36-devel boost-python36-
↪devel postgresql11 postgresql11-server
pip3 install pandas DBUtils psycopg2-binary tabulate mock pytest
```

2.1.2 Build & test

```
cd
git clone https://github.com/HEPCloud/decisionengine

export PYTHONPATH=`pwd`

mkdir decisionengine/framework/logicengine/cxx/build
cd decisionengine/framework/logicengine/cxx/build
cmake3 .. -DPYVER=3.6
make -j <number> # say number of CPUs on your box
cd ../../
ln -s cxx/build/ErrorHandler/RE.so
ln -s cxx/build/ErrorHandler/libLogicEngine.so
export LD_LIBRARY_PATH=`pwd`
cd ../../
#pytest -v --tb=native
python3 -m pytest

==================================== test session starts␣
↪=====================================
platform linux -- Python 3.6.8, pytest-5.3.5, py-1.8.1, pluggy-0.13.1
rootdir: /root/junjk/decisionengine
collected 26 items
```

(continues on next page)

(continued from previous page)

```
framework/dataspace/tests/test_Reaper.py .......                                   ␣
↪ [ 26%]
framework/logicengine/tests/test_cascaded_rules.py ..                              ␣
↪ [ 34%]
framework/logicengine/tests/test_construction.py .....                             ␣
↪ [ 53%]
framework/logicengine/tests/test_facts.py .....                                    ␣
↪ [ 73%]
framework/logicengine/tests/test_pandas_fact.py ..                                 ␣
↪ [ 80%]
framework/logicengine/tests/test_rule_with_negated_fact.py ..                      ␣
↪ [ 88%]
framework/logicengine/tests/test_simple_configuration.py ..                        ␣
↪ [ 96%]
framework/util/tests/test_tsort.py .                                               ␣
↪ [100%]

==================================== 26 passed in 23.86s␣
↪=====================================
```

2.2 Decisionengine_modules

2.2.1 Prerequisites:

In Addition to above installed packages

```
yum install condor
pip3 install htcondor boto boto3 google_auth google-api-python-client gcs-oauth2-boto-
↪plugin
```

2.2.2 Test

```
cd

git clone https://github.com/HEPCloud/decisionengine_modules
python3 -m pytest decisionengine_modules
```

Current status:

```
[root@fermicloud371 tmp]# python3 -m pytest decisionengine_modules
==================================== test session starts␣
↪=====================================
platform linux -- Python 3.6.8, pytest-5.3.5, py-1.8.1, pluggy-0.13.1
rootdir: /root/junjk
collected 85 items

decisionengine_modules/AWS/tests/test_AWSInstancePerformance.py ..                 ␣
↪ [  2%]
decisionengine_modules/AWS/tests/test_AWSJobLimits.py ..                           ␣
↪ [  4%]
decisionengine_modules/AWS/tests/test_AWSOccupancyWithSourceProxy.py ..            ␣
↪ [  7%]
```

(continues on next page)

(continued from previous page)

```
decisionengine_modules/AWS/tests/test_AWSSpotPriceWithSourceProxy.py .. ␣
↪ [  9%]
decisionengine_modules/AWS/tests/test_AWS_figure_of_merit_publisher.py .. ␣
↪ [ 11%]
decisionengine_modules/AWS/tests/test_AWS_price_performance_publisher.py .. ␣
↪ [ 14%]
decisionengine_modules/AWS/tests/test_FigureOfMerit.py ... ␣
↪ [ 17%]
decisionengine_modules/tests/test_AwsBurnRate.py .. ␣
↪ [ 20%]
decisionengine_modules/tests/test_GCEBillingInfo.py .. ␣
↪ [ 22%]
decisionengine_modules/tests/test_GCEFigureOfMerit_publisher.py .. ␣
↪ [ 24%]
decisionengine_modules/tests/test_GCEInstancePerformanceInfo.py .. ␣
↪ [ 27%]
decisionengine_modules/tests/test_GCEPricePerformance_publisher.py .. ␣
↪ [ 29%]
decisionengine_modules/tests/test_GCEResourceLimits.py .. ␣
↪ [ 31%]
decisionengine_modules/tests/test_GceBurnRate.py .. ␣
↪ [ 34%]
decisionengine_modules/tests/test_GceFigureOfMerit.py .. ␣
↪ [ 36%]
decisionengine_modules/tests/test_GceOccupancy.py .. ␣
↪ [ 38%]
decisionengine_modules/tests/test_NerscAllocationInfo.py .. ␣
↪ [ 41%]
decisionengine_modules/tests/test_NerscFigureOfMerit.py .. ␣
↪ [ 43%]
decisionengine_modules/tests/test_NerscFigureOfMerit_publisher.py .. ␣
↪ [ 45%]
decisionengine_modules/tests/test_NerscInstancePerformance.py .. ␣
↪ [ 48%]
decisionengine_modules/tests/test_NerscJobInfo.py .. ␣
↪ [ 50%]
decisionengine_modules/tests/test_factory_client.py .... ␣
↪ [ 55%]
decisionengine_modules/tests/test_factory_entries.py .... ␣
↪ [ 60%]
decisionengine_modules/tests/test_factory_global.py .... ␣
↪ [ 64%]
decisionengine_modules/tests/test_fomorderplugin.py .... ␣
↪ [ 69%]
decisionengine_modules/tests/test_grid_figure_of_merit.py . ␣
↪ [ 70%]
decisionengine_modules/tests/test_htcondor_query.py .... ␣
↪ [ 75%]
decisionengine_modules/tests/test_job_clustering.py ..... ␣
↪ [ 81%]
decisionengine_modules/tests/test_job_clustering_publisher.py .. ␣
↪ [ 83%]
decisionengine_modules/tests/test_job_q.py ... ␣
↪ [ 87%]
decisionengine_modules/tests/test_slots.py .. ␣
↪ [ 89%]
decisionengine_modules/tests/glideinwms/publishers/test_decisionenginemonitor.py ... ␣
↪ [ 92%]
```

(continues on next page)

(continued from previous page)

```
decisionengine_modules/tests/glideinwms/publishers/test_fe_group_classads.py ...      ␣
↪ [ 96%]
decisionengine_modules/tests/glideinwms/publishers/test_glideclientglobal.py ...      ␣
↪ [100%]

====================================== warnings summary␣
↪======================================
/usr/local/lib/python3.6/site-packages/boto/plugin.py:40
  /usr/local/lib/python3.6/site-packages/boto/plugin.py:40: DeprecationWarning: the␣
↪imp module is deprecated in favour of importlib; see the module's documentation for␣
↪alternative uses
    import imp

-- Docs: https://docs.pytest.org/en/latest/warnings.html
=============================== 85 passed, 1 warning in 9.73s␣
↪================================
```

CHAPTER THREE

SOURCE CODE

3.1 Welcome to decisionengine's documentation!

3.1.1 decisionengine package

Subpackages

decisionengine.framework package

Subpackages

decisionengine.framework.configmanager package

Submodules

decisionengine.framework.configmanager.ConfigManager module

class `decisionengine.framework.configmanager.ConfigManager.`**ConfigManager**

Bases: `object`

check_keys(*channel_conf_dict*)

check that channel config has mandatory keys :type data: `dict`

static create(*module_name*, *class_name*, *parameters*)

get_channels()

get_global_config()

get_produces(*channel_config*)

is_updated()

load()

reload()

validate_channel(*channel*)

Validate channels :type channel: `dict`

Module contents

decisionengine.framework.dataspace package

Subpackages

decisionengine.framework.dataspace.datasources package

Submodules

decisionengine.framework.dataspace.datasources.postgresql module

class `decisionengine.framework.dataspace.datasources.postgresql.`**`Postgresql`**(*config_dict*)

Bases: `decisionengine.framework.dataspace.datasource.DataSource`

Implementation of postgresql data source

`_Postgresql__query`(*query_string*, *values=None*, *cursor_factory=None*)

`_abc_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache = <_weakrefset.WeakSet object>`

`_abc_negative_cache_version = 185`

`_abc_registry = <_weakrefset.WeakSet object>`

`_delete`(*sql_query*, *values=None*)

`_insert`(*table_name_or_sql_query*, *record=None*)

`_insert_returning_result`(*table_name_or_sql_query*, *record=None*)

`_remove`(*sql_query*, *values=None*)

`_select`(*query_string*, *values=None*, *cursor_factory=None*)

`_select_dictresult`(*sql_query*, *values=None*)

`_select_getresult`(*sql_query*, *values=None*)

`_select_tuple`(*sql_query*, *values*)

`_update`(*query_string*, *values=None*)

`_update_returning_result`(*query_string*, *values=None*)

`close`()

Close all connections to the database

`connect`()

Create a pool of database connections

`create_tables`()

Create database tables

`delete_data_older_than`(*days*)

Delete data older that days interval :type days: `int` :arg days: remove data older than days interval

`duplicate_datablock`(*taskmanager_id*, *generation_id*, *new_generation_id*)

For the given taskmanager_id, make a copy of the datablock with given generation_id, set the generation_id for the datablock copy

Parameters

- **taskmanager_id** (string) – taskmanager_id for generation to be retrieved
- **generation_id** (int) – generation_id of the data
- **new_generation_id** (int) – generation_id of the new datablock created

get_connection()

get_datablock(*taskmanager_id*, *generation_id*)
Return the entire datablock from the dataproduct table for the given taskmanager_id, generation_id

Parameters

- **taskmanager_id** (string) – taskmanager_id for generation to be retrieved
- **generation_id** (int) – generation_id of the data

get_dataproduct(*taskmanager_id*, *generation_id*, *key*)
Return the data from the dataproduct table for the given taskmanager_id, generation_id, key

Parameters

- **taskmanager_id** (string) – taskmanager_id for generation to be retrieved
- **generation_id** (int) – generation_id of the data
- **key** (string) – key for the value

get_header(*taskmanager_id*, *generation_id*, *key*)
Return the header from the header table for the given taskmanager_id, generation_id, key

Parameters

- **taskmanager_id** (string) – taskmanager_id for generation to be retrieved
- **generation_id** (int) – generation_id of the data
- **key** (string) – key for the value

get_last_generation_id(*taskmanager_name*, *taskmanager_id=None*)
Return last generation id for current task manager or taskmanager w/ task_manager_id.

Parameters

- **name** (string) – task manager name
- **taskmanager_id** (string) – task manager id

get_metadata(*taskmanager_id*, *generation_id*, *key*)
Return the metadata from the metadata table for the given taskmanager_id, generation_id, key

Parameters

- **taskmanager_id** (string) – taskmanager_id for generation to be retrieved
- **generation_id** (int) – generation_id of the data
- **key** (string) – key for the value

get_schema(*table=None*)
Given the table name return it's schema

Parameters **table** (string) – Name of the table

get_taskmanager(*taskmanager_name*, *taskmanager_id=None*)
Retrieve TaskManager :type taskmanager_name: string :arg taskmanager_name: name of taskmanager to retrieve :type taskmanager_id: string :arg taskmanager_id: id of taskmanager to retrieve

insert (*taskmanager_id*, *generation_id*, *key*, *value*, *header*, *metadata*)
Insert data into respective tables for the given taskmanager_id, generation_id, key

Parameters

- **taskmanager_id** (`string`) – taskmanager_id for generation to be retrieved
- **generation_id** (`int`) – generation_id of the data
- **key** (`string`) – key for the value
- **value** (`object`) – Value can be an object or dict
- **header** (`Header`) – Header for the value
- **header** – Metadata for the value

store_taskmanager (*name*, *taskmanager_id*)
Store TaskManager :type taskmanager_name: `string` :arg taskmanager_name: name of taskmanager to retrieve :type taskmanager_id: `string` :arg taskmanager_id: id of taskmanager to retrieve

tables = {'dataproduct': ['taskmanager_id TEXT', 'generation_id INT', 'key TEXT', 'va

update (*taskmanager_id*, *generation_id*, *key*, *value*, *header*, *metadata*)
Update the data in respective tables for the given taskmanager_id, generation_id, key

Parameters

- **taskmanager_id** (`string`) – taskmanager_id for generation to be retrieved
- **generation_id** (`int`) – generation_id of the data
- **key** (`string`) – key for the value
- **value** (`object`) – Value can be an object or dict
- **header** (`Header`) – Header for the value
- **header** – Metadata for the value

`decisionengine.framework.dataspace.datasources.postgresql.`**generate_insert_query** (*table_name*, *keys*)
Generate insert query given table name and list of fields

Parameters

- **table_name** (`str`) – Name of the table to insert into
- **keys** – List of column names

Keys `list`

Return type `str` - insert query

Module contents

Submodules

decisionengine.framework.dataspace.datablock module

class decisionengine.framework.dataspace.datablock.**DataBlock**(*dataspace*, *name*, *taskmanager_id=None*, *generation_id=None*, *sequence_id=None*)

Bases: `object`

_insert(*key*, *value*, *header*, *metadata*)
Insert a new product into database with header and metadata

_setitem(*key*, *value*, *header*, *metadata=None*)
put a product in the database with header and metadata

_update(*key*, *value*, *header*, *metadata*)
Update an existing product in the database with header and metadata

duplicate()
Duplicate the datablock and return this new DataBlock. The intent is that at the point the duplication occurs there is only information from the sources in the DataBlock. This also increments the generation_id of this DataBlock.

TODO: Also update the header and the metadata information TODO: Make this threadsafe

Return type *DataBlock*

get(*key*, *default=None*)
Return the value associated with the key in the database

Return type `dict`

get_header(*key*)
Return the Header associated with the key in the database

Return type *Header*

get_metadata(*key*)
Return the metadata associated with the key in the database

Return type *Metadata*

get_taskmanager(*taskmanager_name*, *taskmanager_id=None*)
Retrieve TaskManager :type taskmanager_name: `string` :arg taskmanager_name: name of taskmanager to retrieve :type taskmanager_id: `string` :arg taskmanager_id: id of taskmanager to retrieve :rtype: :obj: *dict*

The dictionary returned looks like : {'datestamp': datetime.datetime(2017, 12, 20, 17, 37, 17, 503210,

tzinfo=psycopg2.tz.FixedOffsetTimezone(offset=-360, name=None)),

'sequence_id': 135L, 'name': 'AWS_Calculations', 'taskmanager_id': '77B16EB5-C79E-45B0-B1B1-37E846692E1D'}

is_expired(*key=None*)
Check if the dataproduct for a given key or any key is expired

keys()

mark_expired(*expiration_time*)
Set the expiration_time for the current generation of the dataproduct and mark it as expired if expiration_time <= current time

put(*key*, *value*, *header*, *metadata=None*)
Put data into the DataBlock

store_taskmanager(*taskmanager_name*, *taskmanager_id*)
Persist TaskManager, returns sequence number :type taskmanager_name: `string` :type taskmanager_id: :obj: *string* :rtype: `int`

exception `decisionengine.framework.dataspace.datablock.`**ExpiredDataError**
Bases: `Exception`

Errors due to invalid Metadata

class `decisionengine.framework.dataspace.datablock.`**Header**(*taskmanager_id*, *create_time=None*, *expiration_time=None*, *scheduled_create_time=None*, *creator='module'*, *schema_id=None*)
Bases: `collections.UserDict`

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 185

_abc_registry = <_weakrefset.WeakSet object>

default_data_lifetime = 1800

is_valid()
Check if the Header has minimum required information

required_keys = {'create_time', 'creator', 'expiration_time', 'scheduled_create_time',

exception `decisionengine.framework.dataspace.datablock.`**InvalidHeaderError**
Bases: `Exception`

Errors due to invalid Metadata

exception `decisionengine.framework.dataspace.datablock.`**InvalidMetadataError**
Bases: `Exception`

Errors due to invalid Metadata

exception `decisionengine.framework.dataspace.datablock.`**KeyNotFoundError**
Bases: `Exception`

Errors due to invalid Metadata

class `decisionengine.framework.dataspace.datablock.`**Metadata**(*taskmanager_id*, *state='NEW'*, *generation_id=None*, *generation_time=None*, *missed_update_count=0*)
Bases: `collections.UserDict`

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 185

_abc_registry = <_weakrefset.WeakSet object>

required_keys = {'generation_id', 'generation_time', 'missed_update_count', 'state', '

set_state(*state*)
Set the state for the Metadata

valid_states = {'END_CYCLE', 'METADATA_UPDATE', 'NEW', 'START_BACKUP'}

decisionengine.framework.dataspace.datablock.**compress**(*obj*)
Compress python object :param obj: python object :return: compressed object

decisionengine.framework.dataspace.datablock.**decompress**(*zbytes*)
Decompress zipped byte stream, convert to string. :param zbytes: byte stream :return: uncompressed string

decisionengine.framework.dataspace.datablock.**zdumps**(*obj*)
Pickle and compress :param obj: a python object :return: compressed string

decisionengine.framework.dataspace.datablock.**zloads**(*zbytes*)
Decompress and unpickle If input is not compressed attempts to just unpickle it

Parameters **zbytes** – compressed bytes

Returns returns python object

decisionengine.framework.dataspace.datasource module

class decisionengine.framework.dataspace.datasource.**DataSource**(*config*)
Bases: `object`

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 185

_abc_registry = <_weakrefset.WeakSet object>

abstract close()
Close all connections to the database

abstract connect()
Create a pool of database connections

abstract create_tables()
Create database tables

dataproduct_table = 'dataproduct'
Name of the dataproduct table

abstract delete_data_older_than(*days*)
Delete data older that interval :type days: `long` :arg days: remove data older than interval

abstract duplicate_datablock(*taskmanager_id*, *generation_id*, *new_generation_id*)
For the given taskmanager_id, make a copy of the datablock with given generation_id, set the generation_id for the datablock copy

Parameters

- **`taskmanager_id`** (`string`) – taskmanager_id for generation to be retrieved
- **`generation_id`** (`int`) – generation_id of the data
- **`new_generation_id`** (`int`) – generation_id of the new datablock created

`abstract get_datablock`(*taskmanager_id*, *generation_id*)
Return the entire datablock from the dataproduct table for the given taskmanager_id, generation_id

Parameters

- **`taskmanager_id`** (`string`) – taskmanager_id for generation to be retrieved
- **`generation_id`** (`int`) – generation_id of the data

`abstract get_dataproduct`(*taskmanager_id*, *generation_id*, *key*)
Return the data from the dataproduct table for the given taskmanager_id, generation_id, key

Parameters

- **`taskmanager_id`** (`string`) – taskmanager_id for generation to be retrieved
- **`generation_id`** (`int`) – generation_id of the data
- **`key`** (`string`) – key for the value

`abstract get_header`(*taskmanager_id*, *generation_id*, *key*)
Return the header from the header table for the given taskmanager_id, generation_id, key

Parameters

- **`taskmanager_id`** (`string`) – taskmanager_id for generation to be retrieved
- **`generation_id`** (`int`) – generation_id of the data
- **`key`** (`string`) – key for the value

`abstract get_last_generation_id`(*name*, *taskmanager_id=None*)
Return last generation id for current task manager or taskmanager w/ task_manager_id.

Parameters

- **`name`** (`string`) – task manager name
- **`taskmanager_id`** (`string`) – task manager id

`abstract get_metadata`(*taskmanager_id*, *generation_id*, *key*)
Return the metadata from the metadata table for the given taskmanager_id, generation_id, key

Parameters

- **`taskmanager_id`** (`string`) – taskmanager_id for generation to be retrieved
- **`generation_id`** (`int`) – generation_id of the data
- **`key`** (`string`) – key for the value

`abstract get_schema`(*table=None*)
Given the table name return it's schema

Parameters **`table`** (`string`) – Name of the table

`abstract get_taskmanager`(*taskmanager_name*, *taskmanager_id*)
Retrieve TaskManager :type taskmanager_name: `string` :arg taskmanager_name: name of taskmanager to retrieve :type taskmanager_id: `string` :arg taskmanager_id: id of taskmanager to retrieve

header_table = 'header'
Name of the header table

abstract insert(*taskmanager_id*, *generation_id*, *key*, *value*, *header*, *metadata*)
Insert data into respective tables for the given taskmanager_id, generation_id, key

Parameters

- **taskmanager_id** (`string`) – taskmanager_id for generation to be retrieved
- **generation_id** (`int`) – generation_id of the data
- **key** (`string`) – key for the value
- **value** (`object`) – Value can be an object or dict
- **header** (`Header`) – Header for the value
- **header** – Metadata for the value

metadata_table = 'metadata'
Name of the metadata table

abstract store_taskmanager(*taskmanager_name*, *taskmanager_id*)
Store TaskManager :type taskmanager_name: `string` :arg taskmanager_name: name of taskmanager to retrieve :type taskmanager_id: `string` :arg taskmanager_id: id of taskmanager to retrieve

taskmanager_table = 'taskmanager'
Name of the taskmanager table

abstract update(*taskmanager_id*, *generation_id*, *key*, *value*, *header*, *metadata*)
Update the data in respective tables for the given taskmanager_id, generation_id, key

Parameters

- **taskmanager_id** (`string`) – taskmanager_id for generation to be retrieved
- **generation_id** (`int`) – generation_id of the data
- **key** (`string`) – key for the value
- **value** (`object`) – Value can be an object or dict
- **header** (`Header`) – Header for the value
- **header** – Metadata for the value

decisionengine.framework.dataspace.dataspace module

class decisionengine.framework.dataspace.dataspace.**DataSourceLoader**
Bases: `object`

_ds = None

static create_datasource(*module_name*, *class_name*, *config*)

class decisionengine.framework.dataspace.dataspace.**DataSpace**(*config*)
Bases: `object`

DataSpace class is collection of datablocks and provides interface to the database used to store the actual data

_tables_created = False
Description of tables and their columns

close()

delete(*taskmanager_id*, *all_generations=False*)

duplicate_datablock(*taskmanager_id*, *generation_id*, *new_generation_id*)

get_dataproduct(*taskmanager_id*, *generation_id*, *key*)

get_header(*taskmanager_id*, *generation_id*, *key*)

get_last_generation_id(*taskmanager_name*, *taskmanager_id=None*)

get_metadata(*taskmanager_id*, *generation_id*, *key*)

get_taskmanager(*taskmanager_name*, *taskmanager_id=None*)

insert(*taskmanager_id*, *generation_id*, *key*, *value*, *header*, *metadata*)

mark_demented(*taskmanager_id*, *keys*, *generation_id=None*)

mark_expired(*taskmanager_id*, *generation_id*, *key*, *expiry_time*)

store_taskmanager(*name*, *id*)

update(*taskmanager_id*, *generation_id*, *key*, *value*, *header*, *metadata*)

exception `decisionengine.framework.dataspace.dataspace.`**DataSpaceConfigurationError**
Bases: `Exception`

Errors related to database access

exception `decisionengine.framework.dataspace.dataspace.`**DataSpaceConnectionError**
Bases: `Exception`

Errors related to database access

exception `decisionengine.framework.dataspace.dataspace.`**DataSpaceError**
Bases: `Exception`

Errors related to database access

exception `decisionengine.framework.dataspace.dataspace.`**DataSpaceExistsError**
Bases: `Exception`

Errors related to database access

class `decisionengine.framework.dataspace.dataspace.`**Reaper**(*config*)
Bases: `object`

Reaper provides functionality of periodic deletion of data older than retention_interval in days

_reaper_loop(*delay*)

_set_state(*value*)

get_retention_interval()

get_state()

reap()

set_retention_interval(*interval*)

start(*delay=0*)
Start thread with an optional delay to start the thread in X seconds

stop()

class `decisionengine.framework.dataspace.dataspace.`**`Singleton`**

Bases: `type`

Singleton pattern using Metaclass http://stackoverflow.com/questions/6760685/creating-a-singleton-in-python

`_instances = {}`

class `decisionengine.framework.dataspace.dataspace.`**`State`**

Bases: `enum.Enum`

An enumeration.

`ERROR = 7`

`IDLE = 1`

`RUNNING = 3`

`SLEEPING = 4`

`STARTING = 2`

`STOPPED = 6`

`STOPPING = 5`

Module contents

decisionengine.framework.engine package

Submodules

decisionengine.framework.engine.DecisionEngine module

Main loop for Decision Engine. The following environment variable points to decision engine configuration file: `DECISION_ENGINE_CONFIG_FILE` if this environment variable is not defined the `DE-Config.py` file from the ``../tests/etc/` directory will be used.

class `decisionengine.framework.engine.DecisionEngine.`**`DecisionEngine`**(*cfg*, *server_address*, *RequestHandlerClass*)

Bases: `socketserver.ThreadingMixIn`, `xmlrpc.server.SimpleXMLRPCServer`

`_dispatch`(*method*, *params*)

Dispatches the XML-RPC method.

XML-RPC calls are forwarded to a registered function that matches the called XML-RPC method name. If no such function exists then the call is forwarded to the registered instance, if available.

If the registered instance has a _dispatch method then that method will be called with the name of the XML-RPC method and its parameters as a tuple e.g. instance._dispatch('add',(2,3))

If the registered instance does not have a _dispatch method then the instance will be searched to find a matching method and, if found, will be called.

Methods beginning with an '_' are considered private and will not be called.

`get_logger`()

`handle_sighup`(*signum*, *frame*)

reaper_start(*delay*)

reaper_status()

reaper_stop()

reload_config()

rpc_print_product(*product*, *columns=None*, *query=None*)

rpc_print_products()

rpc_reaper_start(*delay=0*)
Start the reaper process after 'delay' seconds. Default 0 seconds delay. :type delay: int

rpc_reaper_status()

rpc_reaper_stop()

rpc_reload_config()

rpc_show_config(*channel=None*)

rpc_start_channel(*channel*)

rpc_start_channels()

rpc_status()

rpc_stop()

rpc_stop_channel(*channel*)

rpc_stop_channels()

start_channel(*channel*)

start_channels()

stop_channel(*channel*)

stop_channels()

class `decisionengine.framework.engine.DecisionEngine.`**RequestHandler**(*request*, *client_address*, *server*)
Bases: `xmlrpc.server.SimpleXMLRPCRequestHandler`

rpc_paths = ('/RPC2',)

class `decisionengine.framework.engine.DecisionEngine.`**RpcServer**(*server_address*, *RequestHandlerClass*)
Bases: `socketserver.ThreadingMixIn`, `xmlrpc.server.SimpleXMLRPCServer`

class `decisionengine.framework.engine.DecisionEngine.`**Worker**(*task_manager*, *config*)
Bases: `multiprocessing.context.Process`

run()
Method to be run in sub-process; can be overridden in sub-class

decisionengine.framework.engine.de_client module

Module contents

decisionengine.framework.modules package

Submodules

decisionengine.framework.modules.LogicEngine module

class decisionengine.framework.modules.LogicEngine.**LogicEngine**(*set_of_parameters*)

Bases: *decisionengine.framework.modules.Module.Module*

evaluate(*data_block*)

decisionengine.framework.modules.Module module

class decisionengine.framework.modules.Module.**Module**(*set_of_parameters*)

Bases: object

get_data_bock()

get_paramaters()

set_data_bock(*data_block*)

decisionengine.framework.modules.Publisher module

class decisionengine.framework.modules.Publisher.**Publisher**(*set_of_parameters*)

Bases: *decisionengine.framework.modules.Module.Module*

consumes(*name_list*)

publish(*data_block=None*)

decisionengine.framework.modules.Source module

class decisionengine.framework.modules.Source.**Source**(*set_of_parameters*)

Bases: *decisionengine.framework.modules.Module.Module*

acquire()

produces(*name_schema_id_list*)

decisionengine.framework.modules.SourceProxy module

Fill in data from another channel data block

class decisionengine.framework.modules.SourceProxy.**SourceProxy**(**args*, ***kwargs*)

Bases: *decisionengine.framework.modules.Source.Source*

Source Proxy Channel configuration using source proxy must have in parameters 'channel_name', defining foreign channel name and 'Dataproducts', defining foreign (and optionally local) data keys. See consumes() doc. Example of source proxy configuration:

"AWSJobLimits" : { "module" : "modules.source_proxy", "name" : "SourceProxy", "parameters": {"channel_name": "channel_aws_config_data",

"Dataproducts":[("aws_instance_limits", "Job_Limits")], "retries": 3, "retry_timeout": 20,

},

"schedule": 360,

},

_get_data(*data_block*, *key*)

acquire()

Overrides Source class method

consumes()

Assumes that self.datakeys has the following structure: is a list of tuples or singletons: [(data_product_name, data_product_name_translation), ...] or [data_product_name, ...]

must_have = ('channel_name', 'Dataproducts')

produces()

Assumes that self.datakeys has the following structure or

decisionengine.framework.modules.SourceProxy.**main**()

Call this a a test unit or use as CLI of this module

decisionengine.framework.modules.SourceProxy.**module_config_info**()

print this module configuration information

decisionengine.framework.modules.SourceProxy.**module_config_template**()

print a template for this module configuration data

decisionengine.framework.modules.Transform module

class decisionengine.framework.modules.Transform.**Transform**(*set_of_parameters*)

Bases: *decisionengine.framework.modules.Module.Module*

consumes(*name_list*)

produces(*name_schema_id_list*)

transform()

decisionengine.framework.modules.de_logger module

Looger to use in all modules

`decisionengine.framework.modules.de_logger.`**`get_logger`**`()`

get default logger - "decision_engine" :rtype: `logging.Logger` - rotating file logger

`decisionengine.framework.modules.de_logger.`**`set_logging`**`(`*log_file_name='/tmp/decision_engine_logs/decision_*
max_file_size=200000000,
max_backup_count=6`)`

Parameters

- **`log_file_name`** (`str`) – log file name
- **`max_file_size`** (`int`) – maximal size of log file. If reached save and start new log.
- **`max_backup_count`** (`int`) – start rotaion after this number is reached

Return type `logging.Logger` - rotating file logger

`decisionengine.framework.modules.de_logger.`**`set_stream_logging`**`(`*logger_name=''*`)`

This is for debugging. Set stream logging for logger.

Parameters **`logger_name`** (`str`) – logger name

Return type `logging.Logger`

Module contents

decisionengine.framework.taskmanager package

Submodules

decisionengine.framework.taskmanager.TaskManager module

Task Manager

`class` `decisionengine.framework.taskmanager.TaskManager.`**`Channel`**`(`*channel_dict*`)`

Bases: `object`

Decision Channel. Instantiates workers according to channel configuration

`class` `decisionengine.framework.taskmanager.TaskManager.`**`TaskManager`**`(`*name*,
task_manager_id,
genera-
tion_id,
chan-
nel_dict,
global_config`)`

Bases: `object`

Task Manager

`data_block_put``(`*data*, *header*, *data_block*`)`

Put data into data block

Parameters

- **`data`** (`dict`) – key, value pairs

- **`header`** (`Header`) – data header
- **`data_block`** (`DataBlock`) – data block

`decision_cycle`()
: Decision cycle to be run periodically (by trigger)

`do_backup`()
: Duplicate current data block and return its copy

  Return type `DataBlock`

`get_state`()

`offline_task_manager`(*current_data_block*)
: offline and stop task manager

`run`()
: Task Manager main loop

`run_logic_engine`(*data_block=None*)
: Run Logic Engine.

  Parameters **`data_block`** (`DataBlock`) – data block

`run_publishers`(*actions*, *facts*, *data_block=None*)
: Run Publishers in main process.

  Parameters **`data_block`** (`DataBlock`) – data block

`run_source`(*src*)
: Get the data from source and put it into the data block

  Parameters **`src`** (`Worker`) – source Worker

`run_transform`(*transform*, *data_block*)
: Run a transform

  Parameters

  - **`transform`** (`Worker`) – source Worker
  - **`data_block`** (`DataBlock`) – data block

`run_transforms`(*data_block=None*)
: Run transforms. So far in main process.

  Parameters **`data_block`** (`DataBlock`) – data block

`set_state`(*state*)

`start_sources`(*data_block=None*)
: Start sources, each in a separate thread

  Parameters **`data_block`** (`DataBlock`) – data block

`stop_task_manager`()
: signal task manager to stop

`wait_for_all`(*events_done*)
: Wait for all sources or transforms to finish

  Parameters **`events_done`** (`list`) – list of events to wait for

`wait_for_any`(*events_done*)
: Wait for any sources to finish

Parameters **events_done** (`list`) – list of events to wait for

class `decisionengine.framework.taskmanager.TaskManager.`**Worker**(*conf_dict*)

Bases: `object`

Provides interface to loadable modules an events to sycronise execution

DEFAULT_SCHEDULE = 300

`decisionengine.framework.taskmanager.TaskManager.`**log_exception**(*logger*, *header_message*)

Module contents

decisionengine.framework.util package

Submodules

decisionengine.framework.util.tsort module

See:

https://en.wikipedia.org/wiki/Topological_sorting

Kahn's topological sorting algorithm

L Empty list that will contain the sorted elements S Set of all nodes with no incoming edge while S is non-empty do

remove a node n from S add n to tail of L for each node m with an edge e from n to m do

remove edge e from the graph if m has no other incoming edges then

insert m into S

if graph has edges then return error (graph has at least one cycle)

else return L (a topologically sorted order)

`decisionengine.framework.util.tsort.`**tsort**(*graph*)

Function implementing Kahn's topological sorting algorithm returns two lists : sorted list and cyclic lost (if graph is acyclic second list is always None)

Return type `list`

Module contents

Module contents

Module contents

3.2 Indices and tables

- genindex
- modindex
- search

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX