decisionengine

Release 1.7.1.dev16+g590bea3f

Fermilab

Sep 30, 2021

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 Notes

HEPCloud's Decision Engine release notes.

The latest release is the designated production release. Decision Engine will support also N-1. New feature development will happen in the development branch and go in the next (N+1) release.

1.1.1 Release 1.7.0

This release features:

- New produces-consumes structure using decorators. This will improve the code quality, improving static checks and reducing the lines of code by removing repetitive boilerplates, especially in the modules.
- Added structured logging. Improved python logging and adoption of structured logs format that will increase the semantinc content of the messages and ease the export of information for dashboards and Elastic Search.
- Added SQLAlchemy object-relational mapper to increase the testability of DB interactions and to allow different database backends. Switching between datasource backends requires dropping all objects if you wish to reuse the tablespace.
- Packaging via setuptools for both decisionengine and decisionengine_modules: Dependencies are not yet fully listed in the RPMs.
- A new, optional, configuration parameter called "channel_name" is available. "channel_name" is one of the keys in the output dictionary of the structured logging and will be used in the upcoming monitoring. If the variable is not defined in the configuration file, then it is taken from the name of the file, e.g. the job_classification.jsonnet config file gives a default "channel_name" value of "job_classification".

Note: Added requirement on SQLAlchemy (for new datasource backend). Non-SQLAlchemy users should ensure the indexes from 13c2f283 are in their database.

Note: Added requirement on prometheus-client. Prometheus will be used as optional monitoring component.

Note: The "channel_name" key in the Source Proxy config dictionaries needs to be changed to "source_channel". "channel_name" is now being used to describe the name of the channel itself, not the name of the channel the Source Proxy is gettting information from.

Issues fixed in this release

- 481: Channel name should be available to all worker types in TaskManager
- 456: Logic engine messages show in the main DE log (1.6.99 post4) prj_testing
- 458: Exception in new SQLAlchemy data source 1.6.99post4
- 455: New postgresql exception in 1.6.99post4 (aka Fixed databese inconsistency silently ignored in v1.6)
- 456: Logic engine messages show in the main DE log (1.6.99 post4)
- 451: Transforms executed in wrong order in 1.6.99.post3
- 367: Test race conditions bug
- 406: Taskmanager doesn't use/honor global log level
- 379: Add postgresql.sql to distributed decisionengine rpm
- 329: Docker container is missing pylint
- 293: Drop requirements.txt setup mode
- 285: Unify ProcessingState with Reaper state management code
- 253: Decision engine can sometimes start up at boot time before network name resolution is working (ae04db5)

Full list of commits since version 1.6.0

f42558df: Updated documentation for 1.7.0 release

029d118a: Updated release notes for 1.7.0 RC4 (1.6.99.post8)

0e19c754: fix SP

810994af: Update release_notes_1.7.rst

fbee95e7: Update release_notes_1.7.rst

68b955b0: Make sure product is a string

ef7a8b96: Automatically adjust PYTHONPATH for tests

e292d388: Updated release notes for 1.7.0 RC3 (1.6.99.post7)

d60b6e4e: new changes for logging with common logger name "channel"

8cdeb67e: Simplify return expression

8fb128d3: Ensure file is "flushed" so name is fully established

7806aa00: Add github CodeQL analysis

9f09bca9: removed modules/LogicEngine.py and corresponding test

b9d28fbf: Cleaner check for *Any*

cc91aa24: Switch to fstring formatting

7bb5b64f: Just return created value rather than store then return

f4847fbe: Combine nested *with* blocks

4ba38bcd: Drop redundant brackets

bdcfe8c9: By convention, pandas is usually imported as *pd*

1dd904ff: Use more traditional expression order cccd31bc: Unused loop vars should start with _

c055a5cd: Drop _*keys* in favor of DB backed *keys*

e8c689b4: Moved prometheus-client requirement to proper place in list

5391500d: Added metrics API module

c2d7835c: Drop unnecessary timeout

c167fc50: Add tests for de-query-tool entry point

efabfeb3: Updated release notes for 1.7.0 RC2 (1.6.99.post6)

b2739c14: moved logging of LogicEngine from decisionengine logger to channel loggers

0c0532f3: Add locks to help ensure data changes are "atomic"

ae63c6ee: Use DB generated known keys so it always matches DB state

b2259e9e: Use public .keys() rather than internal implementation

85b6c3ba: Real world data shows the defaults are fine

95fb3fdf: Further constrain tablespace

3ebe8619: Finish implementation of get_datablock

edbb3568: Add entry point for de-query-tool

fed95c62: adding logging of importlib imports of modules

53e62f03: Sometimes pypy times out on the cleanup.

a44d4bc4: Don't test sqlite on pypy it isn't necessary

b13aa8a9: Some corrections

94c14110: Fix missing defines

5f102095: More detailed testing of datablock

b6c99021: Make sure our sqlite tests have ForeignKeyConditional support

6b76ba7c: Fix typo

6694369d: Ensure dbutils uses transactions

1df400ae: Fix spaces

5278fd99: Raise timout for numpy on pypy

6d0a1a74: Release notes ready for v1.7.0

084f74e1: Initial SQLAlchemy Datasource

3353aa00: Make sure our jsonnet is json synatx valid

402b1c26: Fix transform-ordering problem.

49297573: Fix incorrect packaging of tests at top level

fbfae499: The test_channel loads data once per second.

33f9ade1: Rename taskmanager test nodb

308343e9: Initial modifications for addition of structured logging

6f337b75: Add missing error message

23a4b770: Call fixtures in a cleaner manner for xdist

1f2fe8c4: Add self.config so I can introspect the fixtures later

689c0020: Add missing *config* attrib test

d2732816: Best practices are for fixtues to *yield* vs *return*

accef50a: Seed SQLAlchemy fixtures for later activation

31002bc5: Help define the fixture interlocking

0f5fb129: The pandas 1.3.0 doesn't build against PyPy any longer

a7d18a41: Correctly test datablock construction paths

9af4c144: the *mock* package was a backport for python2.

5ddaff8f: Add another constructor test

9ae9ad13: Make sure if the client says to stop we don't override it

a581cd2b: run pyupgrade against codebase for python3.6

09e4e79c: Handle reaper duplicate shutdowns more cleanly

64d29dc5: Drop pointless cache restore

1c6b2588: Update PyPy to 3.7 for testing

2bae173e: Increase wait for overloaded test workers, update log messages

b67c185c: When aborting CI builds cleanup all processes

6c5d6306: Trim pytest fast functions, add required plugin

8c63ca6b: note why we're ignoring this line

2bd4ecbc: Add a syntax check for the toml files

e2dca404: Sometimes these get stuck

6d012fab: Add in Jenkinsfile pipeline configuration a timeout at stage level

baf07973: Add timeout option to block-while/until

970faf92: Make pre-commit happy

0cea2285: Fix alignment issue

5620c65b: List why we aren't checking

88611d90: Ensure fixtures are cleaned up between invocations

0ba135d2: Setup blank DB for SQLAlchemy tests and prep fixtures

3793e674: Setup pre-commit

9e6d1317: Migrate test_Reaper to pytest fixtures

51df43bf: Cleanup a bunch of pointless whitespace

96e5d069: Fix typo

9f96f418: Setup datablock to use our paramaterized fixture

36ebc66c: Add config for LGTM

c6032e5f: Use topologically sorted transforms to remove some multi-threading.

e063f82a: Drop pointless comma

bfd6689e: Begin prepwork for PEP517

72c5725f: Stub out null source rather than more complex mocking

3b65e5e2: Push Singleton into its own space

fb5b177e: Put fixtures in central location

5ab3cbaa: Add more details to channel startup logs

afe7f7d7: Add log about what DB we are hitting

38034b2c: Let the datasource handle the connections internally

5e03b6fe: Since we are opening an IPv4 socket, just use 127.0.0.1 to check

cac2bef3: Fix missing version requirements

3be8f84f: Add line lenght for autoformater

90e2baad: Protect against inappropriate wait under error condition.

943a17a7: Fix de-client typo and adjust tests accordingly.

3b104eba: Set the logs to DEBUG for testing

4c5564d4: Add another sync method to try and make tests less spotty

66bd81f2: Make sure to encourage updates to tools

d16f04cc: Put postgresql datasource schema into RPM

62b97e79: Fix __str__ so it includes all the data

611ef1f8: Drop pointless lines

5b9e2fb6: Drop unreachable excepts

6991f65f: Restore product-name translation required for some source-proxy cases.

f6258c09: Fixed formatting and updated content

104a0446: Update index.rst

2ed61289: Update index.rst

cb687150: Create release_notes.rst

3b57d4a2: Note new requirement

871af08b: Added 1.7.0 release notes

ce42b802: improved 1.6 release note

583c10fb: fixed rst error

96d4dc1e: Added 1.6.2 release notes, from branch 1.6

13c2f283: Add some helpful indexes to our default schema

29c32571: Log as workers are started

619021c2: One of these tests seems to be spotty, break them out to find which one

29a2c72d: Run the test in a way that gives us colors

4e36bfd2: Drop unused table create logic

5511f69e: Stronger notify state for when we've a lot of watchers.

b6cc7a46: Test the dataspace abstractions

e3b1f594: Better messages about our state

2d2feab9: Drop duplicate tests, leave specifics

8e737329: Add parameter based datasource api tests

5c023aa5: Don’t do debug logs for flake8, they aren’t helpful

f5d1a12f: Setup list of public exports for dataspace.py

7158b422: Merge pull request #365 from jcpunk/bad-update-is-error

cd98cc4a: Update should error out if you try to do it wrongly

eb7907fe: Add option to set taskmanager datestamp and sample usage

e124532c: Make sure the fixture uses the production flow

a8241b6e: Make sure RPM also owns the .egg-info so we don’t confuse the namespaces

da87376e: Ensure the DE server is fully started before running query

622bfacf: Simplify use of our PG fixtures

df98ecdf: Fixed flake8 issue

061ff6cf: decisionengine/framework: stop_channel runs Publisher shutdown methods

3727b80b: Fixup comment to avoid assuming this test uses the DB

d45aaf6b: Fix script path typo

a25a4a30: Fix ABC to match our actual usage

1510b2d1: Address minor linting issues

945e4b16: Fix missing attribute insert

5eace9d5: Add note for how to get modules in place

50a8e268: Add list of packages in the CI env to output

b9cb197d: Sanity check the home directory

cd17223c: Have client provide a hint when you ask for no behavior

95b02365: Fix de-query-tool to support produce/consume model

e660ca72: Update required versions for bugfixes

6863cb81: Fix path error

bb52e8b1: Merge pull request #340 from jcpunk/service-stop

6d7aba95: Drop obsolete files

168ae7aa: Name the tests better

0f60c4e3: Support new produces/consumes/configuration-description infrastructure.

81912469: Add de-query-tool

2a26c944: ExecStopPre is not supported on all systemd instances

67a54d5c: Merge pull request #338 from jcpunk/fix-pytest-postgres

70ab133f: Fixup use of pytest_postgresql for version 3.0.0

f8f4255e: Merge pull request #337 from jcpunk/thread-names

5f49a4f6: Set names for the various parallel code

64da77c6: Merge pull request #327 from jcpunk/datablock-expire de33a60a: Merge pull request #336 from knoepfel/use-toposort

31a8a905: Merge pull request #328 from knoepfel/de-class-inference

410e383d: Merge pull request #331 from jcpunk/reaper-interval-tests

719ff0c8: Test datablock expire funtions

e14c49d8: The 'name' parameter is optional.

7846c9f3: Enable DE class inference based on configuration.

32ab7e44: Use third-party topological sort.

01aa8ae6: Merge pull request #325 from jcpunk/channel-tests

52b48479: Merge pull request #326 from jcpunk/valid-config-tests

8c4749e7: Merge pull request #330 from jcpunk/pylint-actions

a37770c9: Ensure validation testing is tested

d8ab5eb6: Add missing test to ensure the run interval is actually used

0cd9c42b: Also run pylint for extra sanity checks

c5cf1fff: Ensure our errors error out

baf01700: Merge pull request #324 from jcpunk/cleanup-trivial-tests

2a0133aa: Try to cleanup trivial missing coverage

44e0ad6f: Merge pull request #323 from jcpunk/about-coverage

d811f617: Merge pull request #322 from knoepfel/fix-fail-on-error

cb426262: Merge pull request #312 from jcpunk/finish-setuptools

8f6d407d: Merge pull request #316 from jcpunk/abc-coverage

4d0676bb: Merge pull request #317 from vitodb/pylint

d7c43b96: Use regular expression to support fail_on_error feature.

ada66925: add support to run pylint tests

efb1e57b: Finish migration to pure setuptools

bc4720cf: We aren't testing 'unversioned" releases

e4dc35e3: Merge pull request #314 from jcpunk/jsonnet_syntax

87e32c22: Merge pull request #294 from jcpunk/move-reaper

dec85d5e: Merge pull request #319 from jcpunk/task-loop

4108472a: Merge pull request #320 from jcpunk/container-swig

920af1c9: Merge pull request #321 from knoepfel/include-init-files

650dffa7: Don't forget __init__.py files.

1b412e03: The latest m2crypto seems to need swig now

a6e3ab1c: Merge pull request #313 from jcpunk/conf-test

1205636a: Simplify run loop

30e59dc9: fix test_client_with_no_server_verbose unit test for Jenkins CI (#315)

10384a8c: Move reaper into its own place and reuse state logic

940584e4: No real way to test abstract base classes

250c14b1: The *_validate* function doesn't permit missing 'PRODUCES'

5ae1ce9f: Make sure syntax error in config names the problem

b899fa23: Add SourceProxy module test. (#307)

7b3df14c: Increae coverage of utils (#304)

ddba2a31: Fix duplicate entry warning (#311)

915673fa: Test modules minimally (#298)

bc0c21a9: Some repos may error out, don't let them kill the build (#297)

924a7047: doc: add 1.6.1 release notes

b1ab4d31: doc: fix typo

85e5d714: postgresql: do not print stack trace for low level library (#309)

255c6415: Setuptools uses entry return value as an error msg (#303)

2fd8db45: Fix name to match expectations (#305)

9cddb70a: updated release notes

7fe0358e: Error in more clean methods (#300)

84aa506c: Fix a bug in setup.py parsing of requirements. (#301)

a58b61bb: fix typo in release notes

1.1.2 Release 1.6.2

Patch level (bug fix) release.

Issues fixed in this release

Bugs fixed

- DEM 200 (part of it): Invoke correctly channels shutdown: (75eaa90)
- no issue: Use regular expression to support fail_on_error feature (1386d20)

Enhancements:

- Improved CI support (e.g. added pylint tests)
- 217: Add option to de-client –print-product to only print the column names in a data block and-or to print one or more records in key/value format. (c4c7681)

Full list of commits since version 1.6.1

c4c7681: Updated de-query-tool w/ cherry pick of fixes from latest version of PR#332

f964d4b: Fixup use of pytest_postgresql for version 3.0.0

635ffd1: Also run pylint for extra sanity checks

11676ff: Fixed function w/ the same name

b8278f6: Add de-query-tool

75eaa90: Merge pull request #335 from shreyb/publisher_shutdown_from_1.6

77e3d79: Added set_to_shutdown method to TaskManager and accompanying test

1386d20: Merge branch 'knoepfel-fix-fail-on-error' into 1.6

73a18b1: Merge branch 'fix-fail-on-error' of https://github.com/knoepfel/decisionengine into knoepfel-fix-fail-on-error

4f49fb7: Merge branch 'jcpunk-finish-setuptools' into 1.6

a5e5d39: Merge branch 'finish-setuptools' of https://github.com/jcpunk/decisionengine into jcpunk-finish-setuptools

a1ed252: Merge branch 'vitodb-pylint' into 1.6

c8eddda: Merge branch 'pylint' of https://github.com/vitodb/decisionengine into vitodb-pylint Meerging PR#317 to release branch 1.6

d7c43b9: Use regular expression to support fail_on_error feature.

ada6692: add support to run pylint tests

efb1e57: Finish migration to pure setuptools

e4dc35e: Merge pull request #314 from jcpunk/jsonnet_syntax

87e32c2: Merge pull request #294 from jcpunk/move-reaper

dec85d5: Merge pull request #319 from jcpunk/task-loop

4108472: Merge pull request #320 from jcpunk/container-swig

920af1c: Merge pull request #321 from knoepfel/include-init-files

650dffa: Don't forget __init__.py files.

1b412e0: The latest m2crypto seems to need swig now

a6e3ab1: Merge pull request #313 from jcpunk/conf-test

1205636: Simplify run loop

de553a7: fix test_client_with_no_server_verbose unit test for Jenkins CI (#315)

30e59dc: fix test_client_with_no_server_verbose unit test for Jenkins CI (#315)

10384a8: Move reaper into its own place and reuse state logic

250c14b: The *_validate* function doesn't permit missing 'PRODUCES'

5ae1ce9: Make sure syntax error in config names the problem

b899fa2: Add SourceProxy module test. (#307)

7b3df14: Increae coverage of utils (#304)

ddba2a3: Fix duplicate entry warning (#311)

915673f: Test modules minimally (#298)

bc0c21a: Some repos may error out, don't let them kill the build (#297)

924a704: doc: add 1.6.1 release notes

b1ab4d3: doc: fix typo

85e5d71: postgresql: do not print stack trace for low level library (#309)

255c641: Setuptools uses entry return value as an error msg (#303)

2fd8db4: Fix name to match expectations (#305)

9cddb70: updated release notes

7fe0358: Error in more clean methods (#300)

84aa506: Fix a bug in setup.py parsing of requirements. (#301)

a58b61b: fix typo in release notes

33660bf: fixed a typo[locuser@fermicloud462 decisionengine]

1.1.3 Release 1.6.1

Patch level (bug fix) release.

Issues fixed in this release

- 306 : /etc/decisionengine/decision_engine.conf as shipped in RPM is wrong format (de0aef3)
- 275 : Running de-client –stop-channel <channel> results in KeyError (59fb44e)

Full list of commits since version 1.6.0

d7ccd8a : doc: fix typo

ac48e50 : updated release notes

de0aef3 : Fix name to match expectations (#305)

59fb44e : postgresql: do not print stack trace for low level library (#309) (#310)

2162bbe : Setuptools uses entry return value as an error msg (#308)

b0fd9fb : 1.6.0 package backports (#302)

1.1.4 Release 1.6.0

In this release:

- The logic engine has been rewritten in pure python. This removes the last C++ dependency the decision engine had. The build system has been updated accordingly.
- Migrated to setuptools package development library. This build system is the standard vanilla python build system provided with the python distribution. Build configurations have been updated and rpm packaging remains the primary distribution method.
- Completed logging implementation.

- Improvements in error handling and code coverage.
- Improvements in Jenkins and GitHub actions CI/CD pipelines.

Issues fixed in this release

- 44 : Logic Engine doesn't handle missing values gracefully (743effc)
- 253 : Decision engine can sometimes start up at boot time before network name resolution is working (ae04db5)

Full list of commits since version 1.5.0

2551e07 : More coverage for de-client (#296)

dde3945 : Make sure actions either complete in time or die (#295)

381861c : Update Jenkins pipeline configuration (#292)

eb771f4 : Try to cleanup Dockerfile PATH issue (#291)

780cb56 : fix unittest doc

8680942 : update unittest documentation

8154b24 : Fixup sphinx doc (#290)

5f7e13a : enhancements in logging and error handling in dataspace dir (#283)

3d92725 : Add missing runtime requirement (#286)

743effc : Allow conversion from errors to false values in logic-engine expressions. (#284)

124dcab : Inherit version from setuptools_scm if possible (#287)

3669803 : added missing "" as line continuation

761f1d9 : Drop invalid **init**.py

dc0e71b : migrate to setuptools (#264)

3b6f1bf : Make reaper reset state when starting from stopped proc (#280)

b2f9061 : added ISO-8601 format to time in logging. changed name of function for better clarity. (#279)

0a74fe1 : Improved DE client usage (#281)

ebf53e3 : Added shutdown method to Publisher class (#278)

f95ab6d : Address some flake8/black reports (#274)

1c383b7 : Automatically pull in our settings from about.py (#273)

e71f186 : logging and error handling enhancements to taskmanager directory (#277)

7de9ab9 : Increase Reaper log verbosity (#267)

019d245 : Update actions to follow new best practices (#272)

b84e847 : Avoid possible sync issues in reaper startup (#271)

891975f : Remove vestigial C++ files. (#270)

42e5e1f : enhancements in logging and exception handling in newly added logicengine files (#265)

38effe6 : Ensure the scheduler has started the thread before returning (#269)

db54fa1 : Start testing on PyPy with psycopg2cffi (#223) cc44058 : Squashed commit of the following: (#263)

d6548e9 : Enhanced logging in the logicengine directory files (#261)

c341bf7 : Better match our workflow with codecov (#260)

1fbe44d : Use 'new' syntax for forward compat (#259)

2294b0b : Do a limited pin on version requirements (#256)

bcda470 : Python implementation of logic engine (#246)

c6721b4 : address comment on RB

ae04db5 : Add Wants and After (network-online.target) dependency

1a96b14 : Fix action repodata

a70cee8 : Move to CodeCov.io

7b16b4e : Add Wants and Requires dependencies (#258)

76c3670 : Move to CodeCov.io (#254)

e7ba013 : Fix action repodata (#255)

d7e72f2 : revert 3.9 test

b04154b : added 1.5.0 release notes

a03da29 : remove 3.9 to see if documentatoin gets generated

1.1.5 Release 1.5.0

In this release:

- Introduce data product query interface
- Cleanup of Ligic Engine code
- Improvements in error handling
- Improvements in testing and CI

Issues fixed in this release

- 217 , 218 : Add option to de-client –print-product to only print the column names in a data block and-or to print one or more records in key/value format (fe7abcf)
- 240 : Logic Engine call leads to immediate taskmanager segfault exit (d855aa0)
- 239 : implement data product browsing interface (fe9faa9)

Full list of commits since version 1.4.1

d66c54b : Add PEP-0396 metadata (#243)

bfc91a6 : More compat between psycopg2/psycopg2cffi (#248)

f5d31a6 : Cleanup Fixture FIXME (#249)

0dfaf3c : Adding docker documentation (#251)

4b166a2 : Since we are python3 only now, drop python-six compat layer (#252)

fe7abcf : Add format support to de-client (#217) (#241)

df5a3d7 : Add wheel support for easier testing (#247)

7de970d : Add place to inject env if need be (#242)

84e2930 : Fix race in test case (#250)

d855aa0 : Fix fact-lookup to support duplicate names in separate rules. (#245)

51370fb : Resolve fixture 'quickstart' issue (#238)

3ea9129 : Move from TravisCI to raw actions (#235)

fe9faa9 : implement data product browsing interface (#239)

cf0f3c0 : Add support to use custom base docker container to run tests (#234)

d91722f : Compat with psycopg2cffi (#233)

7d15a8c : Test failing source proxy. (#232)

b9a4bbb : Add debug logs for which threads are created #176 (#231)

6e6f4c9 : Updated Jenkins configuration documentation (#229)

2d9fd7b : Log if config passed validation #117 (#230)

60c46d3 : Self-test needs a real namespace to 'import numpy' in new python eval (#228)

a120077 : Test that the doc actually builds during CI (#227)

4b6240a : Extend timeout for coverage combine (#226)

b059696 : Update workflow per changes at github (#225)

7a71cac : Use newer compilers/runtimes (#224)

15ffd93 : Add header for strict includes (#222)

71b141a : Add special PyPy only requirement (#221)

9dbb932 : Move Python C extension to versioned .so file (#220)

ea7ade5 : Migrate from boost-python to pybind11 (#215)

e6b2eae : Add python 3.9 to testing matrix (#219)

04c8f9c : Add the option to print columns types on de-client (#216)

8815dc6 : Logic-engine cleanups (#211)

086d0d5 : fix missing back tick

54cc084 : modified release notes

24744cf : Synchronize access to the task managers (#214)

87a7fda : replde dash with underscore

743d0fd : try sphinx_rtd_theme

18c7909 : added 1.4.0 release notes

ff3d491 : force docker pull when building the docker container to make sure to use an updated base layer (#210)

1.1.6 Release 1.4.1

In this release:

- Bug fixes to 1.4.0 release

Issues fixed in this release

- 213 : de-client hangs under certain circumstances in version 1.4 and greater (race condition) (84ecfe2)

Full list of commits since version 1.4.0

9799b9a : update release version to 1.4.1

84ecfe2 : Synchronize access to the task managers (#214)

751b6b8 : Address data races; remove need to sleep in unit tests (#205)

1.1.7 Release 1.4.0

In this release:

- Improvements in error handling and client/server interactions
- Added log rotation by time
- Improvements in code coverage

Issues fixed in this release

- 153 : Have de-client –print-product return different error message if product does not exist (18a950c)
- 171 : yum update on decision engine rpm from python2 to python3 doesn't undo the symlinks (eb85c97)
- 188 : Channel debug info now leaks into startup.log (99d20a5)
- 208 : Error when trying to run reaper in version 1.4.0 (84eccf3)

Full list of commits since version 1.3

84eccf3 : Fix typo in reaper script. (#209)

d836abf : next RC

926944a : Fix coveralls reporting (#198)

b95c323 : Updating base Dockerfile (#199)

d302e31 : Help jsonnet, which doesn't understand PosixPath objects. (#204)

2d791a7 : Test configuration policies. (#197)

236e27a : Ensure items are returned in a stable order (#202)

e974f5f : add pylinit and pycodestyle (#203)

fbe7616 : Test task manager (#196)

686ca80 : require more recent version of pytest-postgresql (#195)

99d20a5 : Fix double-logging problem. (#192)

4ce3d17 : A set of fixtures to simplify unit tests (#183)

65f8052 : Fix typo (#190)

f3a4be8 : Protect against None workers (#187)

ec310fb : remove py3 from package name

7006489 : bump version to 1.4.0rc

158d835 : decisionengine/framework/modules: Fix SourceProxy retries (#184)

1356bf1 : Add support to test any branch in Jenkins (#182)

692fa8e : Add timeout support for unit test on Jenkins (#181)

e3d6e6a : Updated Jenkins documentation to take into account unit tests timeout parametr (#180)

2586a3e : Configuration redesign (#168)

fac984d : Fix error with DBUtils import. Looks like names of modules changed (#175)

7d661ee : Move postgres-specific implementation to postgres source. (#174)

eb85c97 : Rpm (#173)

10fe843 : Adding log rotation by time (#170)

a8d239b : Various improvements. (#167)

d9b92ee : Ignore vim's *.swp files (#166)

d9f72ef : Fix call to shutdown_timeout (and add sample entry to config) (#165)

3161795 : Add drops for items using tables being dropped (#164)

77d186d : Show output of test runtimes in travis (#163)

81820a4 : Allow server to start with no channels. (#161)

49879a6 : DE server and client usability improvements (#160)

de91c4f : Add tests to default and override config (#158)

14df1f6 : Use python fallthrough for options (#159)

ac64a92 : Drop python 2.7 integration tests since we are python3 only (#157)

d963301 : Update Jenkins pipeline to properly test closing PR (#156)

64248cb : Merge 'runtime' tests into running channel tests (#150)

065ad77 : Adding Jenkins pipeline documentation (#155)

18a950c : fix print-product to report non-existing product as such (#154)

6493735 : Fix invalid attribute name (#152)

d953c6a : Remove unnecessary set_start_method call (#149)

c8c9b65 : guarantee that process is killed so test never hang (#147) f1542b6 : Channel test (#146)

7f349a8 : Fix faulty TaskManager state type (#145)

d50f1c4 : fix logging regression introduced in f5e299969e0611e3480e9fa2782052df… (#142)

becfa26 : Pass the correct type. (#144)

1a60daf : DecisionEngine: fix typo (#143)

9e7b867 : Updating Jenkins pipeline configuration (#140)

e3a6703 : fix regression introduced in f5e299969e0611e3480e9fa2782052df86d7c4ed (#141)

4900bc6 : Restore runtime test. (#139)

0823f3d : Consolidate DE server/client tests into one file. (#138)

4f84435 : A few more access fixes.

160cfd1 : Fix task manager state access.

c00d819 : A few more cleanups.

ec087e2 : Various cleanups

a309ffe : Improvements to DE client CLI.

1.1.8 Release 1.3.0

In this release:

- Introduced Jsonnet based configuration system
- Improved logging
- Improved coverage of datasource

Full list of commits since version 1.2

239e82c : postgresql: improve SQL query (#133)

668eb1f : Update to make the code compatible with both python and JSON based config files (#129)

afd8837 : Configuration-manager fixes (#128)

571e2be : Remove pip installed system python packages

407d9ed : Update Dockerfile

1fefc69 : Implement unit tests for datablock.py (#122)

43c8d7a : Adjust global configuration to include program-option values. (#126)

2840813 : Switch to Jsonnet configuration system (#125)

5c4ae0e : logging changes: added config file and command line interface (#124)

6697f22 : Further config-manager testing and factorizations. (#123)

fa89fd0 : Insulate multiprocessing test from parent environment. (#120)

139a537 : Allow empty base directory for log file. (#119)

f14d40c : Factorize configuration-loading steps. (#118)

e00afee : Enhance testing and error reporting of ConfigManager (#117) c3d1be3 : Python 3 upgrades. (#116)

e7399af : Header fix (#114)

0456abf : Adding editor config file, see https://editorconfig.org/ (#115)

82112d1 : Dockerfile: fetch osg 3.5 repo rpm (#113)

97c21b1 : osg version 3.5 (#112)

33f28a8 : Introduce jsonnet dependency (#110)

3f8b55e : improve server error handling (#108)

f15588e : added 1.2.0 release notes

b433325 : Remove unnecessary 'main' functionality. (#107)

1.1.9 Release 1.2.0

In this release:

- Swithed to python3
- Improved coverage
- Database data retention : added reaper to remove data older than configurable number of days
- Improved logging

decisionengine

3dfe167 : Jenkins pipeline improvements (#106)

22a7073 : pull request for review request 137 (#105)

cafffb2 : Make it possible to run code directly (for tests), and (#100)

802e98b : replace psycog2 witt psycopg2-binary (#101)

573ce8f : Jenkins pipeline improvements (#99)

9d08835 : Run coveralls even under failed state (#97)

bc1df4b : Add tests for PostgreSQL datasource (#71)

c1ac391 : Fix missing py-modules.html (#96)

8dbfdee : Setup gh-pages doc workflow (#94)

cd4a01a : Doc (#93)

673080d : set version to 1.2.0 (for now). Supply conf file that corresponds to (#91)

f912225 : Db (#92)

dc8b68a : Add reaper to the RPC (#83) (#90)

29ade91 : adding .Jenkinsfile with Jenkins pipeline configuration (#86)

c1dfe5c : Don't exclude E1004 from pylint, do exclude line breaks (#89)

440f949 : Fix varname (#88)

313d135 : Compress (#87)

6b8dc4b : Revert "Add reaper to the RPC (#83)" dbea8e5 : Update utils.sh so pytest will complete.

e848316 : Update to postgresql11

7f4b805 : Add reaper to the RPC (#83)

0ba2c51 : remove astpp module and depedencies it pulls in (#81)

6b8eab9 : don't track test coverage of tests (#80)

0da18ec : made reaper.py executable

aca24a3 : make reaper.py executable, make symbolic link to it from /usr/bin (#72)

0202acf : Implementation of data reaper (#70)

16b6be1 : Simple changes for Python 3 deployment (#69)

fd2418c : Fix warnings caught by PEP-8 Speaks.

d16359b : Python 3 (and other) simplications.

3c7b6b7 : Only run Github Actions for python3.6 (#68)

453cbba : Update README.md

b27ed53 : remove unnecessary (and atually harmful) python shebang (#66)

decisionengine_modules

30d928b : clone version 1.2.0 of decisionengine

ae7c5a6 : Jenkins pipeline improvements (#236)

310befd : T198 (#235)

a65886d : Fix import as reported in : https://github.com/HEPCloud/decisionengin… (#232)

93711cc : Run coveralls even if tests fail (#229)

03d763a : Jenkins pipeline improvements (#230)

f48d30f : Fix/223 (#228)

c8aa262 : github ticket 199 (#222)

0323bda : Address : https://github.com/HEPCloud/decisionengine_modules/issues/224 (#226)

62e4df6 : Add support to run CI on Jenkins (#221)

5ab1541 : bump master version to 1.2.0 (for now) (#219)

bc19c65 : decisionengine_modules/NERSC: Added retry loop for NERSC API Calls (#220)

41a50de : Sync up pep8speaks and run_pylint.sh with decisionengine settings (#218)

db4634f : silence pylint error (#217)

1b95141 : Fix whitespace around operator error

746ea38 : ignore W503

8a8b5f4 : remove unused variable

a6668bf : fix PEP8 warnings

13773ee : address pep8 warnings

6bea4ca : silence pylint error

f589895 : Pass sort=True parameter to fix future warning (#215)

a1d0507 : fixing pep8 warning

a10bd17 : debugging one import error

ec501ad : make coveralls.io links work

deab1a7 : T201 (#204)

69f2645 : Add coveragerc

6d8a5f5 : decisionengine_modules/NERSC: Make Nersc API call backward-compatible with old config (#196)

a7e0af9 : Only run Github Actions for python3.6 (#24)

1.1.10 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

The developer documentation is in the GitHub Wiki

Intructions to build the package, or to run unit tests and other CI tests, and to install decisionengine are in the GitHub Wiki as well.

CHAPTER

THREE

JENKINS CI PIPELINE

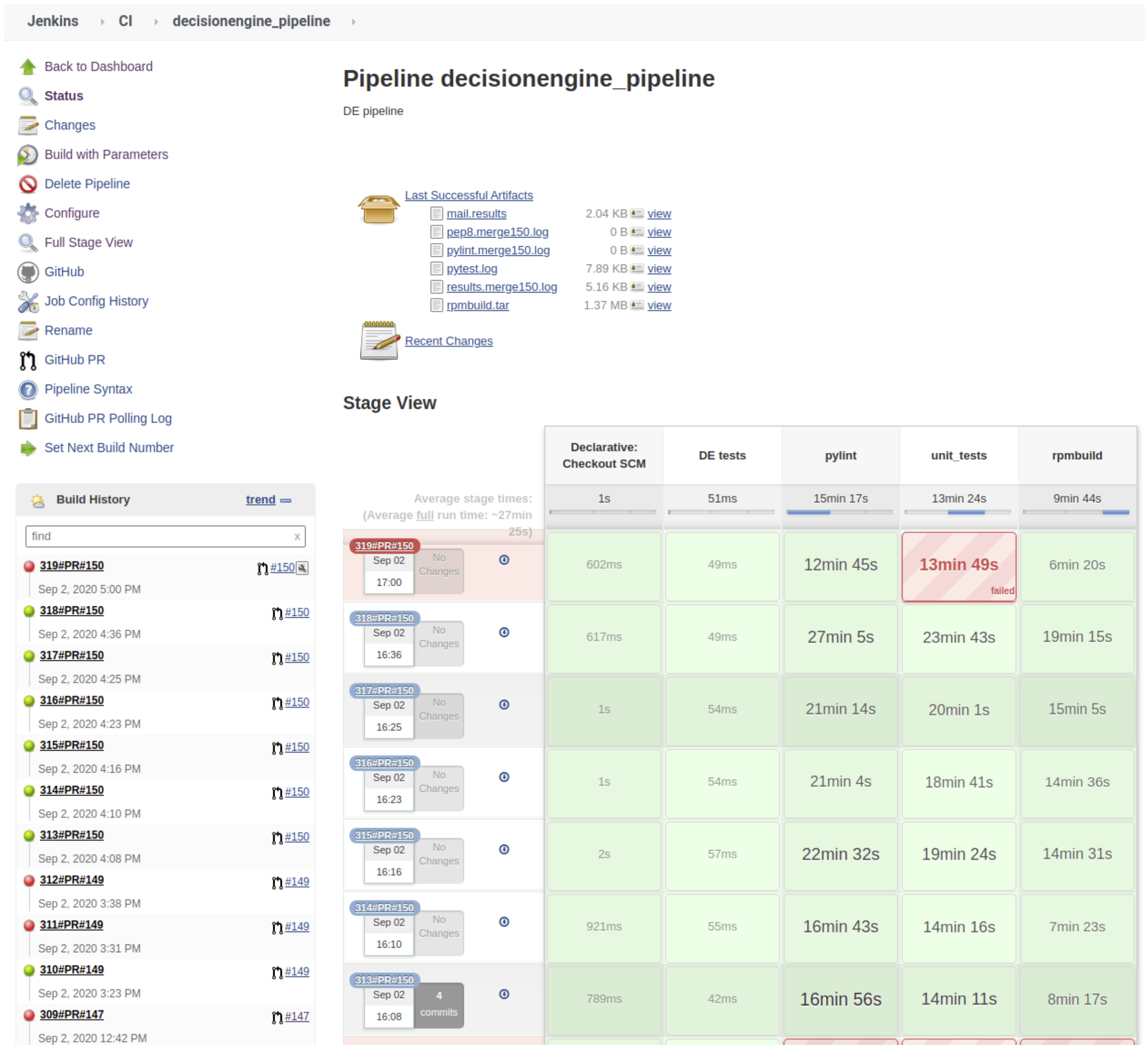
3.1 Decisionengine CI with Jenkins pipeline

Jenkins dashboard with Decisionengine framework CI results is available here.

A CI build is triggered any time a PR is created/closed or a commit is made to an existing PR.
There are also *nightly CI builds* to test a list of predefined branches.

The Jenkins pipeline runs *pylint* and *unit_tests* test suites alongside the *rpmbuild* stage.

The Jenkins dashboard looks like this:



Jenkins › CI › decisionengine_pipeline ›

- Back to Dashboard
- Status
- Changes
- Build with Parameters
- Delete Pipeline
- Configure
- Full Stage View
- GitHub
- Job Config History
- Rename
- GitHub PR
- Pipeline Syntax
- GitHub PR Polling Log
- Set Next Build Number

Pipeline decisionengine_pipeline

DE pipeline

Last Successful Artifacts

mail.results	2.04 KB	view
pep8.merge150.log	0 B	view
pylint.merge150.log	0 B	view
pytest.log	7.89 KB	view
results.merge150.log	5.16 KB	view
rpmbuild.tar	1.37 MB	view

Recent Changes

Build History trend

find

319#PR#150	#150	Sep 2, 2020 5:00 PM
318#PR#150	#150	Sep 2, 2020 4:36 PM
317#PR#150	#150	Sep 2, 2020 4:25 PM
316#PR#150	#150	Sep 2, 2020 4:23 PM
315#PR#150	#150	Sep 2, 2020 4:16 PM
314#PR#150	#150	Sep 2, 2020 4:10 PM
313#PR#150	#150	Sep 2, 2020 4:08 PM
312#PR#149	#149	Sep 2, 2020 3:38 PM
311#PR#149	#149	Sep 2, 2020 3:31 PM
310#PR#149	#149	Sep 2, 2020 3:23 PM
309#PR#147	#147	Sep 2, 2020 12:42 PM

Stage View

	Declarative: Checkout SCM	DE tests	pylint	unit_tests	rpmbuild
Average stage times: (Average full run time: ~27min 25s)	1s	51ms	15min 17s	13min 24s	9min 44s
319#PR#150 Sep 02 17:00 No Changes	602ms	49ms	12min 45s	13min 49s failed	6min 20s
318#PR#150 Sep 02 16:36 No Changes	617ms	49ms	27min 5s	23min 43s	19min 15s
317#PR#150 Sep 02 16:25 No Changes	1s	54ms	21min 14s	20min 1s	15min 5s
316#PR#150 Sep 02 16:23 No Changes	1s	54ms	21min 4s	18min 41s	14min 36s
315#PR#150 Sep 02 16:16 No Changes	2s	57ms	22min 32s	19min 24s	14min 31s
314#PR#150 Sep 02 16:10 No Changes	921ms	55ms	16min 43s	14min 16s	7min 23s
313#PR#150 Sep 02 16:08 4 commits	789ms	42ms	16min 56s	14min 11s	8min 17s

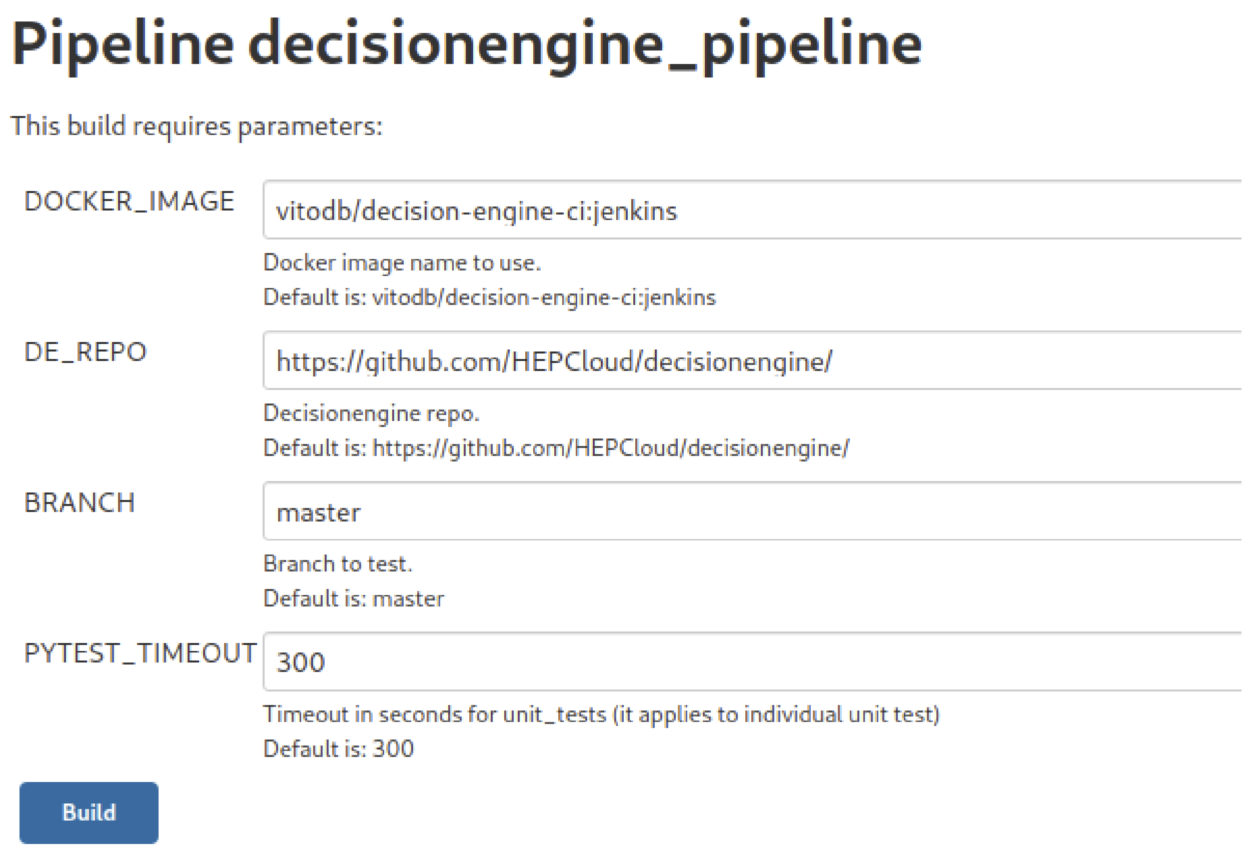
On the bottom left side there is the list of recent CI builds that are named after the PR or the branch tested.
On the bottom right side the dashboard shows for each CI build detailed status for each test suite.

Hovering the mouse over the *status box* for each CI build stage, a tool-tip with a button to access log details shows up.

Next to the build number the symbol ⊙ gives access to a menu with the list of artifacts stored for that build. Those artifacts include logs and the tarball with RPMs.

From the panel on the left side it is possible to access the PR on GitHub by clicking on the PR icon that looks like this #142 .

On occasion it could be useful to trigger a manual CI build to test a branch on the official DE GitHub repository or on the user fork. For this purpose, on the top left panel the user can click on the Build with Parameters button, and this panel shows up



Pipeline decisionengine_pipeline

This build requires parameters:

DOCKER_IMAGE	vitodb/decision-engine-ci:jenkins Docker image name to use. Default is: vitodb/decision-engine-ci:jenkins
DE_REPO	https://github.com/HEPCloud/decisionengine/ Decisionengine repo. Default is: https://github.com/HEPCloud/decisionengine/
BRANCH	master Branch to test. Default is: master
PYTEST_TIMEOUT	300 Timeout in seconds for unit_tests (it applies to individual unit test) Default is: 300

the user can modify these parameters to customize what code to test with the CI build.

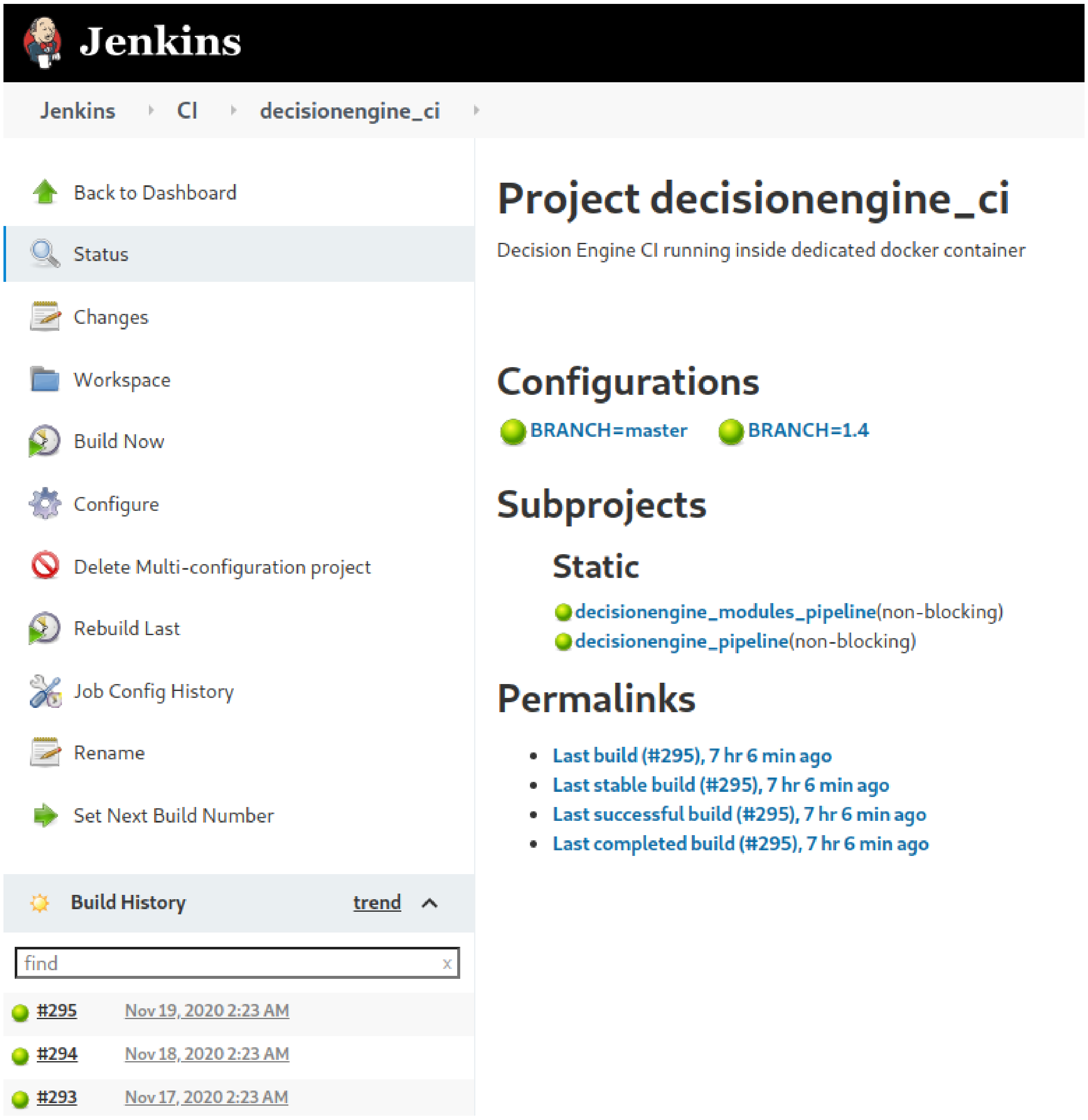
The *DE_REPO* parameter can point to the user fork or to the main repository.
The *BRANCH* parameter can point to the desired branch to test.
The *PYTEST_TIMEOUT* parameter is the timeout in seconds for *unit_tests*.

When ready, by clicking on the *Build* button, the CI build will start.

The pipeline configuration is part of the decisionengine repo.

3.1.1 Nightly CI build configuration

The nightly CI build for Decisionengine framework uses this Jenkins project that triggers a CI build using the Jenkins pipeline described above to test a list of predefined branches.


Jenkins

Jenkins › CI › decisionengine_ci ›

Back to Dashboard
Status
Changes
Workspace
Build Now
Configure
Delete Multi-configuration project
Rebuild Last
Job Config History
Rename
Set Next Build Number

Build History trend

find

#295	Nov 19, 2020 2:23 AM
#294	Nov 18, 2020 2:23 AM
#293	Nov 17, 2020 2:23 AM

Project decisionengine_ci

Decision Engine CI running inside dedicated docker container

Configurations

BRANCH=master BRANCH=1.4

Subprojects

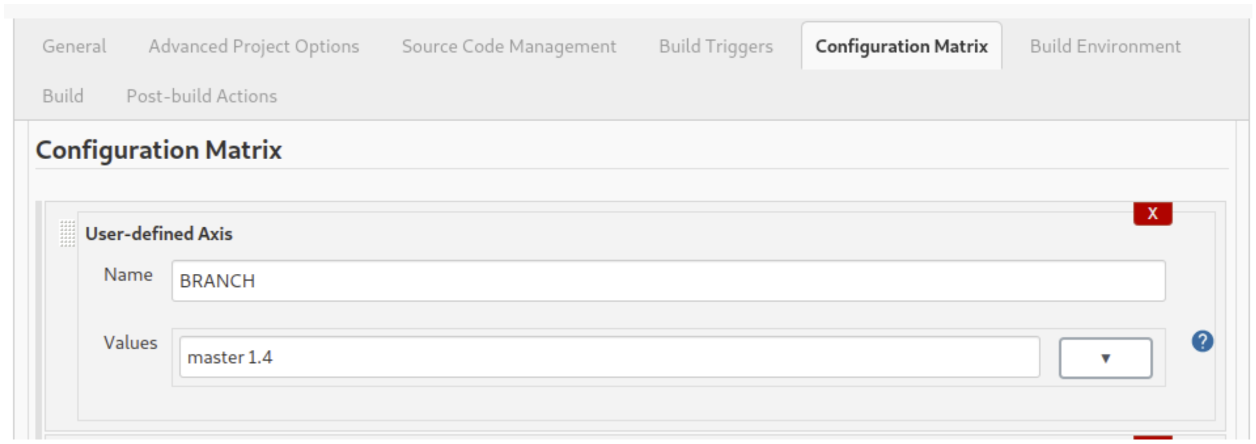
Static

decisionengine_modules_pipeline(non-blocking)
decisionengine_pipeline(non-blocking)

Permalinks

- Last build (#295), 7 hr 6 min ago
- Last stable build (#295), 7 hr 6 min ago
- Last successful build (#295), 7 hr 6 min ago
- Last completed build (#295), 7 hr 6 min ago

Branches to test are defined using the project matrix as shown in the picture below.
Each branch in the list (here *master* and *1.4*) spawns an independent CI build.



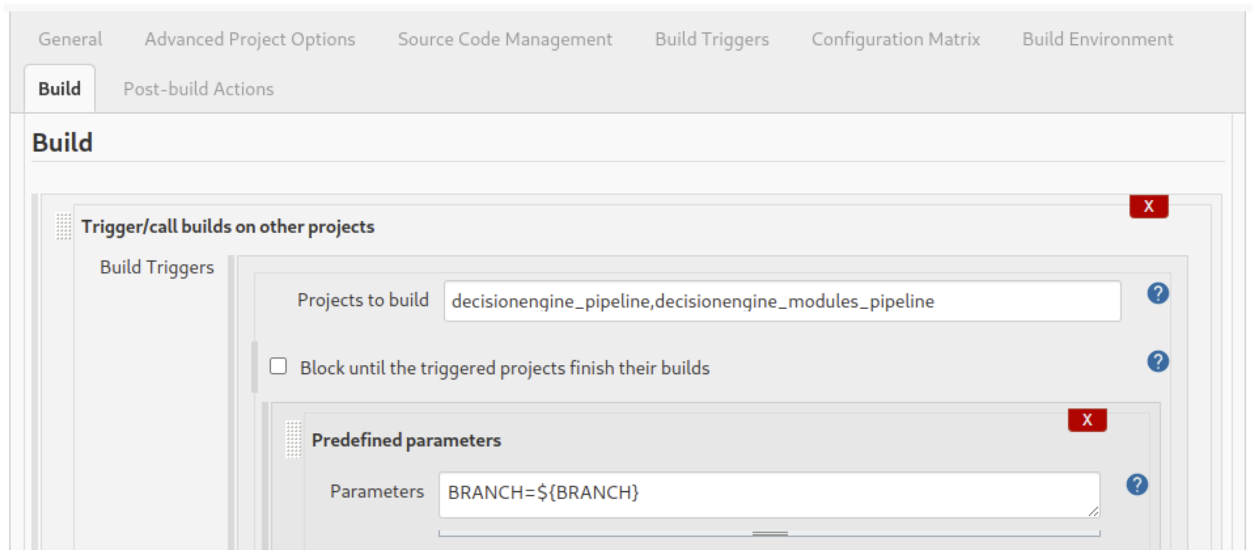
Configuration Matrix

User-defined Axis

Name: BRANCH

Values: master 1.4

In the *Build* section of the configuration it is set the list of Jenkins subprojects to be triggered, in this case we have *decisionengine_pipeline* and *decisionengine_modules_pipeline*.
The *Parameters* text box is used to override parameters of each Jenkins subproject with a custom value.
In total this Jenkins project triggers 4 CI builds, i.e. 2 branches X 2 Jenkins subprojects.



Build

Trigger/call builds on other projects

Build Triggers

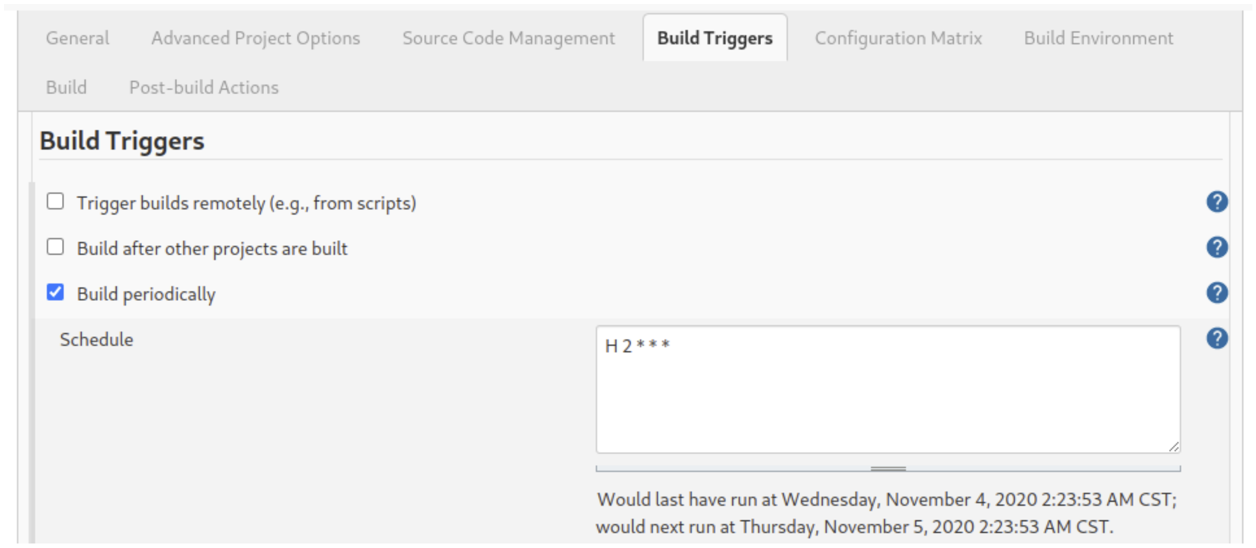
Projects to build: decisionengine_pipeline,decisionengine_modules_pipeline

☐ Block until the triggered projects finish their builds

Predefined parameters

Parameters: BRANCH=${BRANCH}

Finally the *Build Triggers* section is used to setup the schedule for the periodic build,
in this case it is scheduled to run at about 2 AM.
Jenkins will choose the actual time depending on the actual load on the system.



Build Triggers

- [ ] Trigger builds remotely (e.g., from scripts)
- [ ] Build after other projects are built
- [x] Build periodically

Schedule

```
H 2 * * *
```

Would last have run at Wednesday, November 4, 2020 2:23:53 AM CST; would next run at Thursday, November 5, 2020 2:23:53 AM CST.

CHAPTER

FOUR

SOURCE CODE

4.1 Welcome to decisionengine's documentation!

4.1.1 decisionengine package

Subpackages

decisionengine.framework package

Subpackages

decisionengine.framework.config package

Subpackages

decisionengine.framework.config.tests package

Submodules

decisionengine.framework.config.tests.test_config module

`decisionengine.framework.config.tests.test_config.`**`_channel_config_dir`**`(`*relative_dir*`)`

`decisionengine.framework.config.tests.test_config.`**`_global_config_file`**`(`*relative_filename*`)`

`decisionengine.framework.config.tests.test_config.`**`load`**`()`

`decisionengine.framework.config.tests.test_config.`**`test_channel_empty_config`**`(`*load*, *caplog*`)`

`decisionengine.framework.config.tests.test_config.`**`test_channel_empty_dictionary`**`(`*load*, *caplog*`)`

`decisionengine.framework.config.tests.test_config.`**`test_channel_invalid_modules_list`**`(`*load*, *caplog*`)`

`decisionengine.framework.config.tests.test_config.`**`test_channel_invalid_modules_no_keys`**`(`*load*, *caplog*`)`

`decisionengine.framework.config.tests.test_config.`**`test_channel_invalid_modules_string`**`(`*load*, *caplog*`)`

`decisionengine.framework.config.tests.test_config.`**`test_channel_loading`**`(`*caplog*`)` decisionengine.framework.config.tests.test_config.**test_channel_module_missing_all**(*load*, *caplog*)

decisionengine.framework.config.tests.test_config.**test_channel_module_missing_module**(*load*, *caplog*)

decisionengine.framework.config.tests.test_config.**test_channel_module_missing_parameters**(*load*, *caplog*)

decisionengine.framework.config.tests.test_config.**test_channel_names**(*load*)

decisionengine.framework.config.tests.test_config.**test_channel_no_config_files**(*load*)

decisionengine.framework.config.tests.test_config.**test_channel_no_modules**(*load*)

decisionengine.framework.config.tests.test_config.**test_empty_config**(*load*)

decisionengine.framework.config.tests.test_config.**test_minimal_jsonnet_right_extension**(*load*, *capsys*)

decisionengine.framework.config.tests.test_config.**test_minimal_jsonnet_wrong_extension**(*load*, *capsys*)

decisionengine.framework.config.tests.test_config.**test_syntax_error_in_config_names_bad_file**(*load*)

decisionengine.framework.config.tests.test_config.**test_valid_but_empty_config**(*load*)

decisionengine.framework.config.tests.test_de_std module

decisionengine.framework.config.tests.test_de_std.**config**(*basename*, *jpathdirs=None*)

decisionengine.framework.config.tests.test_de_std.**test_combine_one_level**()

decisionengine.framework.config.tests.test_de_std.**test_combine_one_level_skip_proxies**()

decisionengine.framework.config.tests.test_de_std.**test_error_on_duplicate_keys**()

decisionengine.framework.config.tests.test_de_std.**test_jpath**()

decisionengine.framework.config.tests.test_policies module

decisionengine.framework.config.tests.test_policies.**test_channel_config_dir**(*tmp_path*, *monkeypatch*)

decisionengine.framework.config.tests.test_policies.**test_global_config_dir**(*tmp_path*, *monkeypatch*)

decisionengine.framework.config.tests.test_policies.**test_global_config_file**(*tmp_path*, *monkeypatch*)

decisionengine.framework.config.tests.test_policies.**test_valid_dir**(*tmp_path*)

Module contents

Submodules

decisionengine.framework.config.ChannelConfigHandler module

Manager of channel configurations.

The ChannelConfigHandler manages only channel configurations and not the global decision-engine configuration. It is responsible for loading channel configuration files and validating that the channels have the correct configuration artifacts.

`class decisionengine.framework.config.ChannelConfigHandler.`**`ChannelConfigHandler`**`(`*global_config*, *channel_config_dir*`)`

Bases: `object`

`_load_channel``(`*channel_name*, *path*`)`

`get_channels``()`

`load_all_channels``()`

Load all channel configurations inside the stored channel-configuration directory.

Any cached configurations will be dropped prior to reloading.

`load_channel``(`*channel_name*`)`

Load a single configuration for a channel with the supplied name.

The behavior is to read a configuration file whose path is:

<cached channel config. dir>/{channel_name}.jsonnet

where the cached channel-configuration directory was stored whenever the ChannelConfigHandler object was created, and {channel_name} is the value of the supplied method argument.

`print_channel_config``(`*channel*`)`

`decisionengine.framework.config.ChannelConfigHandler.`**`_check_keys`**`(`*channel_conf_dict*`)`

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

`decisionengine.framework.config.ChannelConfigHandler.`**`_make_de_logger`**`(`*global_config*`)`

decisionengine.framework.config.ValidConfig module

ValidConfig represents a valid JSON document.

The decision engine requires each of its configuration files to be valid JSON. This is achieved by either supplying a valid Jsonnet or JSON document upfront.

Vetting of a file for JSON validity happens upon construction of a 'ValidConfig' object. A fully constructed 'ValidConfig' object thus corresponds to a valid JSON document.

`class decisionengine.framework.config.ValidConfig.`**`ValidConfig`**`(`*filename*, *jpathdirs=None*`)`

Bases: `collections.UserDict`

ValidConfig represents a valid JSON configuration in the form of a dictionary.

In addition to the normal dictionary operations, users may call 'dump()' to print out in a string form the JSON configuration.

`_abc_impl = <_abc._abc_data object>`

`dump()`

Print dictionary data to a valid JSON string.

`decisionengine.framework.config.ValidConfig._config_from_file`(*config_file*, *jpaths=None*)

decisionengine.framework.config.policies module

Decision-engine default configuration policies.

For the decision-engine process, the configuration policies are:

- The global configuration file must be named 'decision_engine.jsonnet' and it must reside in (a) a directory that can be accessed through the 'CONFIG_PATH' environment variable, or (b) the /etc/decisionengine directory.
- All channel configurations must reside in (a) a directory accessible through the 'CHANNEL_CONFIG_PATH' environment variable, or (b) a 'config.d' subdirectory of the /etc/decisionengine directory.

The utilities provided in this module provide simple means of accessing the configuration artifacts according to the policies listed above. Please consult the documentation for each function below for more detailed information.

`decisionengine.framework.config.policies.channel_config_dir`(*parent_dir=None*)

Retrieve the channel configuration directory as a pathlib.Path object.

This function returns a path object according to the following precedence rules:

1. If the 'parent_dir' argument is provided, the returned path object will correspond to '{parent_dir}/config.d'.
2. If the 'CHANNEL_CONFIG_PATH' environment variable has been set, the returned path object will correspond to ${CHANNEL_CONFIG_PATH}.
3. If neither 1 or 2 apply, the returned path object corresponds to '{global_config_dir()}/config.d' (see documentation for 'global_config_dir()').

Regardless of the precedence rule used, the returned path object must be a valid directory or an exception will be raised–i.e. if the 'parent_dir' argument is supplied, and the resulting path object is not a valid directory, the function will exit with an exception and not attempt rule 2 or 3.

`decisionengine.framework.config.policies.global_config_dir()`

Retrieve global configuration dir as pathlib.Path object.

This is the directory that houses the 'decision_engine.jsonnet' global configuration file.

This function checks that the 'CONFIG_PATH' variable has been set or will use /etc/decisionengine otherwise. If the path exists as a directory, then the directory path is returned as a string; otherwise an exception is raised.

`decisionengine.framework.config.policies.global_config_file`(*parent_dir=None*)

Return the pathlib.Path object corresponding to the global configuration.

If supplied, the 'parent_dir' is assumed to be the full path corresponding to a directory containing the 'decision_engine.jsonnet' file. If not provided, the global configuration directory is determined based on the behavior of the 'global_config_dir()' function.

An exception is raised if no 'decision_engine.jsonnet' file is found.

`decisionengine.framework.config.policies.valid_dir`(*path*, *scope*)

Throws if the supplied path object is not a directory, otherwise returns the path object.

Module contents

decisionengine.framework.dataspace package

Subpackages

decisionengine.framework.dataspace.datasources package

Subpackages

decisionengine.framework.dataspace.datasources.sqlalchemy_ds package

Submodules

decisionengine.framework.dataspace.datasources.sqlalchemy_ds.datasource_api module

The datasource layer for our abstraction

`class decisionengine.framework.dataspace.datasources.sqlalchemy_ds.datasource_api.`**`SQLAlchemyDS`**`(`*config_dict*

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

A DecisionEngine data source via the SQL Alchemy ORM

{

“dataspace”: {

“datasource”: { “module”: “decisionengine.framework.dataspace.datasources.sqlalchemy_ds”, “name”: “SQLAlchemyDS”, “params”: {

“pool_size”: 5, “max_overflow”: 10, “timeout”: 30,

# url is mandatory, but any *engine* keyword is accepted here. “url”: “dialect[+driver]://user:password@host/dbname”

}

}

}

}

Exceptions should be caught and logged by the caller.

`_abc_impl = <_abc._abc_data object>`

`close``()`

Close all connections to the database

Returns None

`connect``()`

Create a pool of database connections

Returns None

`create_tables``()`

Create database tables

Returns None

`delete_data_older_than`(*days*)
Delete data older that interval

Parameters `days` (*int*) – remove data older than this many days

Returns None

`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` (*str/uuid*) – id of taskmanager to retrieve
- `generation_id` (*int*) – generation id to clone
- `new_generation_id` (*int*) – generation id to create

Returns None

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

Parameters

- `taskmanager_id` (*str/uuid*) – id of taskmanager to retrieve
- `generation_id` (*int*) – generation id to locate

Returns with all set keys and their associated values

Return type dict

`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` (*str/uuid*) – id of taskmanager to retrieve
- `generation_id` (*int*) – generation id to locate
- `key` (*str*) – key for the value

Returns The possibly binary value stored earlier

Return type obj

`get_dataproducts`(*taskmanager_id*, *key=None*)
Return list of all data products associated with with taskmanager_id

Parameters

- `taskmanager_id` (*str/uuid*) – id of taskmanager to retrieve
- `key` (*str*) – key for the value

Returns each element is the matching row as a dict()

Return type tuple

`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` (*str/uuid*) – id of taskmanager to retrieve

- **generation_id** (*int*) – generation id to locate
- **key** (*str*) – key for the value

Returns

fields in order are: taskmanager.taskmanager_id, header.taskmanager_id, header.generation_id, header.key, header.create_time, header.expiration_time, header.scheduled_create_time, header.creator, header.schema_id

Return type tuple

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

Parameters

- **taskmanager_name** (*str*) – name of taskmanager to retrieve
- **taskmanager_id** (*str/uuid*) – id of taskmanager to retrieve

Returns the largest generation stored within the database

Return type int

`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** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to locate
- **key** (*str*) – key for the value

Returns

fields in order are: taskmanager.taskmanager_id, metadata.taskmanager_id, metadata.generation_id, metadata.key, metadata.state, metadata.generation_time, metadata.missed_update_count

Return type tuple

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

`get_taskmanager`(*taskmanager_name*, *taskmanager_id=None*)
Find the task manager by name/uuid in the database get back the primary key.

If multiples match, find highest primary key.

Parameters

- **taskmanager_name** (*str*) – name of taskmanager to retrieve
- **taskmanager_id** (*str/uuid*) – id of taskmanager to retrieve

Returns the matching row, column names as keys

Return type dict

`get_taskmanagers`(*taskmanager_name=None*, *start_time=None*, *end_time=None*)
Find taskmanagers that meet our search

Parameters

- **taskmanager_name** (*str*) – name of taskmanager to retrieve

- **start_time** (*datetime*) – Datetime to confine against
- **end_time** (*datetime*) – Datetime to confine against

Returns each element is a dict() matching row, column names as keys

Return type list

`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** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to create
- **key** (*str*) – key for the value
- **value** (*obj*) – Value can be an object or dict or a binary
- **header** (`datablock.Header`) – Header for the value
- **metadata** (`datablock.Metadata`) – Metadata for the value

Returns None

`reset_connections`()
Reset the connection to the database. So long as self.engine isn't undef, the engine can still make new connections if new db actions happen. It just wont have any open at this time.

Returns None

`store_taskmanager`(*name*, *taskmanager_id*, *datestamp=None*)
Store TaskManager in database

Parameters

- **name** (*str*) – name of taskmanager to retrieve
- **taskmanager_id** (*str/uuid*) – id of taskmanager to retrieve
- **datestamp** (*datetime*) – datetime of created object, defaults to 'now'

Returns the primary key of the row in the database

Return type int

`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** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to update
- **key** (*str*) – key for the value
- **value** (*obj*) – Value can be an object or dict or a binary
- **header** (`datablock.Header`) – Header for the value
- **metadata** (`datablock.Metadata`) – Metadata for the value

Returns None

decisionengine.framework.dataspace.datasources.sqlalchemy_ds.db_schema module

The table layout and utilities for our SQLAlchemy ORM

`class decisionengine.framework.dataspace.datasources.sqlalchemy_ds.db_schema.Base(**kwargs)`

Bases: `object`

The base class of the class hierarchy.

When called, it accepts no arguments and returns a new featureless instance that has no instance attributes and cannot be given any.

`_sa_registry = <sqlalchemy.orm.decl_api.registry object>`

`metadata = MetaData()`

`registry = <sqlalchemy.orm.decl_api.registry object>`

`class decisionengine.framework.dataspace.datasources.sqlalchemy_ds.db_schema.Dataproduct(**kwargs)`

Bases: *sqlalchemy.orm.decl_api.Base*

The PRIMARY KEY on this table isn't used. . . .

Existing code appears to depend on column order.

`_sa_class_manager = {'generation_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'key': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'taskmanager': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'taskmanager_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'value': <sqlalchemy.orm.attributes.InstrumentedAttribute object>}`

`generation_id`

`id`

`key`

`taskmanager`

`taskmanager_id`

`value`

`class decisionengine.framework.dataspace.datasources.sqlalchemy_ds.db_schema.Header(**kwargs)`

Bases: *sqlalchemy.orm.decl_api.Base*

The PRIMARY KEY on this table isn't used. . . .

The existing code has a hard expectation on the time columns being BIGINT rather than datetime objects burried within the classes.

Looks like there was an inital goal of a relationship with the Schema table, but it may not be in use

Existing code appears to depend on column order.

`_sa_class_manager = {'create_time': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'creator': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'expiration_time': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'generation_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'key': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'scheduled_create_time': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'schema_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'taskmanager': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'taskmanager_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>}`

`create_time`

`creator`

`expiration_time`

`generation_id`

`id`

`key`

`scheduled_create_time`

`schema_id`

`taskmanager`

`taskmanager_id`

`class decisionengine.framework.dataspace.datasources.sqlalchemy_ds.db_schema.Metadata(**kwargs)`

Bases: `sqlalchemy.orm.decl_api.Base`

The PRIMARY KEY on this table isn't used. . . .

The existing code has a hard expectation on the state field as a 'text' element.

The existing code has a hard expectation on the time columns being BIGINT rather than datetime objects burried within the classes.

Existing code appears to depend on column order.

`_sa_class_manager = {'generation_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'generation_time': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'key': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'missed_update_count': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'state': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'taskmanager': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'taskmanager_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>}`

`generation_id`

`generation_time`

`id`

`key`

`missed_update_count`

`state`

taskmanager

taskmanager_id

class decisionengine.framework.dataspace.datasources.sqlalchemy_ds.db_schema.**Schema**(***kwargs*)

Bases: *sqlalchemy.orm.decl_api.Base*

This table may not be in use

Has a one-to-many relationship with: Header - may not be in use

_sa_class_manager = {'schema': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'schema_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>}

schema

schema_id

class decisionengine.framework.dataspace.datasources.sqlalchemy_ds.db_schema.**Taskmanager**(***kwargs*)

Bases: *sqlalchemy.orm.decl_api.Base*

Has a one-to-many relationship with: Header Metadata Dataproduct

changes cascade on: Header Metadata Dataproduct

Existing code appears to depend on column order.

_sa_class_manager = {'datestamp': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'name': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'sequence_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'task_dataproduct': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'task_header': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'task_metadata': <sqlalchemy.orm.attributes.InstrumentedAttribute object>, 'taskmanager_id': <sqlalchemy.orm.attributes.InstrumentedAttribute object>}

datestamp

name

sequence_id

task_dataproduct

task_header

task_metadata

taskmanager_id

decisionengine.framework.dataspace.datasources.sqlalchemy_ds.utils module

Code not written by us

decisionengine.framework.dataspace.datasources.sqlalchemy_ds.utils.**add_engine_pidguard**(*engine*)

Based on https://stackoverflow.com/questions/62920507/using-sqlalchemy-connection-pooling-queues-with-python-multiprocess

decisionengine.framework.dataspace.datasources.sqlalchemy_ds.utils.**clone_model**(*model*, ***kwargs*)

Based on https://stackoverflow.com/a/55991358

decisionengine.framework.dataspace.datasources.sqlalchemy_ds.utils.**orm_as_dict**(*obj*)

Based on : https://stackoverflow.com/a/37350445

Module contents

Top level import so we can rationally segment items of the ORM

class decisionengine.framework.dataspace.datasources.sqlalchemy_ds.**SQLAlchemyDS**(*config_dict*)

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

A DecisionEngine data source via the SQL Alchemy ORM

{

"dataspace": {

"datasource": { "module": "decisionengine.framework.dataspace.datasources.sqlalchemy_ds", "name": "SQLAlchemyDS", "params": {

"pool_size": 5, "max_overflow": 10, "timeout": 30,

# url is mandatory, but any *engine* keyword is accepted here. "url": "dialect[+driver]://user:password@host/dbname"

}

}

}

}

Exceptions should be caught and logged by the caller.

_abc_impl = <_abc._abc_data object>

close()

Close all connections to the database

Returns None

connect()

Create a pool of database connections

Returns None

create_tables()

Create database tables

Returns None

delete_data_older_than(*days*)

Delete data older that interval

Parameters **days** (*int*) – remove data older than this many days

Returns None

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** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to clone
- **new_generation_id** (*int*) – generation id to create

Returns None

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

Parameters

- **taskmanager_id** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to locate

Returns with all set keys and their associated values

Return type dict

`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** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to locate
- **key** (*str*) – key for the value

Returns The possibly binary value stored earlier

Return type obj

`get_dataproducts`(*taskmanager_id*, *key=None*)
Return list of all data products associated with with taskmanager_id

Parameters

- **taskmanager_id** (*str/uuid*) – id of taskmanager to retrieve
- **key** (*str*) – key for the value

Returns each element is the matching row as a dict()

Return type tuple

`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** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to locate
- **key** (*str*) – key for the value

Returns

fields in order are: taskmanager.taskmanager_id, header.taskmanager_id, header.generation_id, header.key, header.create_time, header.expiration_time, header.scheduled_create_time, header.creator, header.schema_id

Return type tuple

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

Parameters

- **taskmanager_name** (*str*) – name of taskmanager to retrieve
- **taskmanager_id** (*str/uuid*) – id of taskmanager to retrieve

Returns the largest generation stored within the database

Return type int

`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** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to locate
- **key** (*str*) – key for the value

Returns

fields in order are: taskmanager.taskmanager_id, metadata.taskmanager_id, metadata.generation_id, metadata.key, metadata.state, metadata.generation_time, metadata.missed_update_count

Return type tuple

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

`get_taskmanager`(*taskmanager_name*, *taskmanager_id=None*)
Find the task manager by name/uuid in the database get back the primary key.

If multiples match, find highest primary key.

Parameters

- **taskmanager_name** (*str*) – name of taskmanager to retrieve
- **taskmanager_id** (*str/uuid*) – id of taskmanager to retrieve

Returns the matching row, column names as keys

Return type dict

`get_taskmanagers`(*taskmanager_name=None*, *start_time=None*, *end_time=None*)
Find taskmanagers that meet our search

Parameters

- **taskmanager_name** (*str*) – name of taskmanager to retrieve
- **start_time** (*datetime*) – Datetime to confine against
- **end_time** (*datetime*) – Datetime to confine against

Returns each element is a dict() matching row, column names as keys

Return type list

`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** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to create
- **key** (*str*) – key for the value
- **value** (*obj*) – Value can be an object or dict or a binary

- **header** (`datablock.Header`) – Header for the value
- **metadata** (`datablock.Metadata`) – Metadata for the value

Returns None

`reset_connections()`

Reset the connection to the database. So long as self.engine isn't undef, the engine can still make new connections if new db actions happen. It just wont have any open at this time.

Returns None

`store_taskmanager`(*name*, *taskmanager_id*, *datestamp=None*)

Store TaskManager in database

Parameters

- **name** (*str*) – name of taskmanager to retrieve
- **taskmanager_id** (*str/uuid*) – id of taskmanager to retrieve
- **datestamp** (*datetime*) – datetime of created object, defaults to 'now'

Returns the primary key of the row in the database

Return type int

`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** (*str/uuid*) – id of taskmanager to retrieve
- **generation_id** (*int*) – generation id to update
- **key** (*str*) – key for the value
- **value** (*obj*) – Value can be an object or dict or a binary
- **header** (`datablock.Header`) – Header for the value
- **metadata** (`datablock.Metadata`) – Metadata for the value

Returns None

decisionengine.framework.dataspace.datasources.tests package

Submodules

decisionengine.framework.dataspace.datasources.tests.fixtures module

pytest fixtures/constants

`decisionengine.framework.dataspace.datasources.tests.fixtures.PG_DE_DB_WITHOUT_SCHEMA`(*request: _pytest.fixtures.Fixture*
→
psycopg2.extensions.c

Fixture factory for PostgreSQL.

Parameters **request** – fixture request object

Returns postgresql client

decisionengine.framework.dataspace.datasources.tests.fixtures.**PG_DE_DB_WITH_SCHEMA**(*PG_DE_DB_WITHOUT_S*

Load our PG schema into the database via this fixture so pytest knows the limitations on parallel usage of this database scope.

decisionengine.framework.dataspace.datasources.tests.fixtures.**PG_PROG**(*request: _pytest.fixtures.FixtureRequest*, *tmpdir_factory: _pytest.tmpdir.TempdirFactory*) → Iterator[pytest_postgresql.executor.PostgreS

Process fixture for PostgreSQL.

Parameters **request** – fixture request object

Returns tcp executor

decisionengine.framework.dataspace.datasources.tests.fixtures.**SQLALCHEMY_PG_WITH_SCHEMA**(*PG_DE_DB_WITH*

Get a blank database from pytest_postgresql. Then setup the SQLAlchemy style URL with that DB. The SQLAlchemyDS will create the schema as needed.

decisionengine.framework.dataspace.datasources.tests.fixtures.**SQLALCHEMY_TEMPFILE_SQLITE**(*tmp_path*)

Setup an SQLite database with the pytest tmp_path fixture. Then setup the SQLAlchemy style URL with that DB. The SQLAlchemyDS will create the schema as needed.

decisionengine.framework.dataspace.datasources.tests.fixtures.**datasource**(*request*)

This parameterized fixture will setup up various datasources.

Add datasource objects to DATASOURCES_TO_TEST once they've got our basic schema loaded. And adjust our *if* statements here until we are SQLAlchemy only.

Pytest should take it from there and automatically run it through all the tests using this fixture.

decisionengine.framework.dataspace.datasources.tests.fixtures.**mock_data_block**()

This fixture replaces the standard datablock implementation.

The current DataBlock implementation does not own any data products but forwards them immediately to a backend datasource. The only implemented datasource requires Postgres, which is overkill when needing to test simple data-product communication between modules.

This mock datablock class directly owns the data products, thus avoiding the need for a datasource backend. It is anticipated that a future design of the DataBlock will own the data products, thus making this mock class unnecessary.

decisionengine.framework.dataspace.datasources.tests.test_datasource_api module

This test plan covers a generic dataspace object via pytest parameters.

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_create_tables**(*datasource*)

create_tables() should be safe to call multiple times

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_delete_data_older_than_ar**

Can we delete old entries

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_duplicate_datablock**(*dataso*

Can we duplicate taskmanager1 and all its entries

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_datablock**(*datasource*)

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_dataproduct**(*datasource*)

Can we get the dataproduct by uuid with key

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_dataproduct_not_exist**
Does it error out if we ask for bogus information?

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_dataproducts**(*datasource*
Can we get the dataproducts by uuid and uuid with key

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_dataproducts_not_exis**
Does it error out if we ask for bogus information?

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_header**(*datasource*)
Can we fetch a header?

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_header_not_exist**(*datas*
Does it error out if we ask for a bogus header?

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_last_generation_id**(*dat*
Can we get the last generation id by name or name and uuid

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_last_generation_id_no**
Does it error out if we ask for a bogus taskmanager?

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_metadata**(*datasource*)
Can we fetch a metadata element?

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_metadata_not_exist**(*dat*
Does it error out if we ask for a bogus metadata element?

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_taskmanager_exists**(*dat*
Can I get a taskmanager by name or name and uuid

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_taskmanager_not_exist**
This should error out

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_taskmanagers**(*datasource*
Can I get multimple task managers

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_get_taskmanagers_not_exis**
Do I error out when asking for garbage

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_has_config**(*datasource*)
This should have a *config* dict we can pass to jsonnet

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_insert**(*datasource*)
Can we insert new elements

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_reset_connections**(*datasour*
reset_connections() should be safe to call any time

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_store_taskmanager**(*datasour*
Can we make new entries

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_update**(*datasource*)
Do updates work as expected

decisionengine.framework.dataspace.datasources.tests.test_datasource_api.**test_update_bad**(*datasource*)
Do updates fail to work on bogus taskmanager as expected

decisionengine.framework.dataspace.datasources.tests.test_postgresql module

decisionengine.framework.dataspace.datasources.tests.test_postgresql.**test_generate_insert_query**()

Module contents

Submodules

decisionengine.framework.dataspace.datasources.null module

class decisionengine.framework.dataspace.datasources.null.**NullDataSource**(*config_dict*)

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

Implementation of data source ABC that does nothing

_abc_impl = <_abc._abc_data object>

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 interval :type days: `long` :arg days: remove data older than 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_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_dataproducts`(*taskmanager_id*, *key=None*)
Return list of all data products associated with with taskmanager_id

Parameters **key** (`string`) – data product key

`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

- **taskmanager_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

`get_taskmanagers`(*taskmanager_name=None*, *start_time=None*, *end_time=None*)
Retrieve TaskManagers :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

`reset_connections`()
Drop any cached connections and reconnect to the database

store_taskmanager(*name*, *taskmanager_id*, *datestamp=None*)
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 :type datestamp: `datetime` :arg datestamp: datetime of created object, defaults to 'now'

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 module

class decisionengine.framework.dataspace.datasources.postgresql.**Postgresql**(*config_dict*)
Bases: `decisionengine.framework.dataspace.datasource.DataSource`

Implementation of postgresql data source

__query(*query_string*, *values=None*, *cursor_factory=None*)

_abc_impl = <_abc._abc_data 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_dataproducts`(*taskmanager_id*, *key=None*)
Return list of all data products associated with with taskmanager_id

Parameters **key** (`string`) – data product key

`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

- **taskmanager_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

`get_taskmanagers`(*taskmanager_name=None*, *start_time=None*, *end_time=None*)
Retrieve TaskManagers :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

`reset_connections`()
Drop any cached connections and reconnect to the database

`store_taskmanager`(*name*, *taskmanager_id*, *datestamp=None*)
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 :type datestamp: `datetime` :arg datestamp: datetime of created object, defaults to 'now'

`tables = {'dataproduct': ['taskmanager_id TEXT', 'generation_id INT', 'key TEXT', 'value BLOB'], 'header': ['taskmanager_id TEXT', 'generation_id INT', 'key TEXT', 'create_time REAL', 'expiration_time REAL', 'scheduled_create_time REAL', 'creator TEXT', 'schema_id INT'], 'metadata': ['taskmanager_id TEXT', 'generation_id INT', 'key TEXT', 'state TEXT', 'generation_time REAL', 'missed_update_count INT'], 'schema': ['schema_id INT', 'schema BLOB']}`

`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

decisionengine.framework.dataspace.tests package

Submodules

decisionengine.framework.dataspace.tests.fixtures module

`decisionengine.framework.dataspace.tests.fixtures.`**`PG_DE_DB_WITHOUT_SCHEMA`**(*request: _pytest.fixtures.FixtureRequest*) → psycopg2.extensions.connection

Fixture factory for PostgreSQL.

Parameters **request** – fixture request object

Returns postgresql client

`decisionengine.framework.dataspace.tests.fixtures.`**`PG_DE_DB_WITH_SCHEMA`**(*PG_DE_DB_WITHOUT_SCHEMA*)

Load our PG schema into the database via this fixture so pytest knows the limitations on parallel usage of this database scope.

`decisionengine.framework.dataspace.tests.fixtures.`**`PG_PROG`**(*request: _pytest.fixtures.FixtureRequest*, *tmpdir_factory: _pytest.tmpdir.TempdirFactory*) → Iterator[pytest_postgresql.executor.PostgreSQLExecutor]

Process fixture for PostgreSQL.

Parameters **request** – fixture request object

Returns tcp executor

`decisionengine.framework.dataspace.tests.fixtures.`**`SQLALCHEMY_PG_WITH_SCHEMA`**(*PG_DE_DB_WITHOUT_SCHEMA*

Get a blank database from pytest_postgresql. Then setup the SQLAlchemy style URL with that DB. The SQLAlchemyDS will create the schema as needed.

`decisionengine.framework.dataspace.tests.fixtures.`**`SQLALCHEMY_TEMPFILE_SQLITE`**(*tmp_path*)

Setup an SQLite database with the pytest tmp_path fixture. Then setup the SQLAlchemy style URL with that DB. The SQLAlchemyDS will create the schema as needed.

`decisionengine.framework.dataspace.tests.fixtures.`**`datasource`**(*request*)

This parameterized fixture will setup up various datasources.

Add datasource objects to DATASOURCES_TO_TEST once they've got our basic schema loaded. And adjust our *if* statements here until we are SQLAlchemy only.

Pytest should take it from there and automatically run it through all the tests using this fixture.

decisionengine.framework.dataspace.tests.fixtures.**dataspace**(*request*)

This parameterized fixture will setup up various datasources. Add datasource objects to DATASOURCES_TO_TEST once they've got our basic schema loaded. And adjust our *if* statements here until we are SQLAlchemy only.

Pytest should take it from there and automatically run it through all the tests using this fixture.

decisionengine.framework.dataspace.tests.fixtures.**load_sample_data_into_datasource**(*schema_only_db*)

load our sample test data into a dataspace This is a function not a fixture so you can run it on any datasource providing the right API.

decisionengine.framework.dataspace.tests.test_Reaper module

decisionengine.framework.dataspace.tests.test_Reaper.**config**()

decisionengine.framework.dataspace.tests.test_Reaper.**reaper**(*request*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_fail_bad_config**(*reaper*, *config*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_fail_missing_config**(*reaper*, *config*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_fail_missing_config_key**(*reaper*, *config*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_fail_small_retain**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_fail_small_run_interval**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_fail_start_two_reapers**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_fail_wrong_config_key**(*reaper*, *config*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_just_stop_no_error**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_loop_of_start_stop_in_clumps**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_reap_default_state**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_reaper_can_reap**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_source_fail_can_be_fixed**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_start_delay**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_start_stop**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_start_stop_stop**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_state_can_be_active**(*reaper*)

decisionengine.framework.dataspace.tests.test_Reaper.**test_state_sets_timer_and_uses_it**(*reaper*)

decisionengine.framework.dataspace.tests.test_datablock module

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_constructor**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_duplicate**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_get_dataproducts**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_get_header**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_get_metadata**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_get_taskmanager**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_is_expired**(*dataspace*)

This test just validates the method/function exists. The stub within our default code should be replaced by a class inheriting from it. That class should have more rational return types.

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_is_expired_with_key**(*dataspace*)

This test just validates the method/function exists. The stub within our default code should be replaced by a class inheriting from it. That class should have more rational return types.

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_key_management**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_key_management_change_name**(*dataspa*

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_mark_expired**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_no_key_by_name**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_DataBlock_to_str**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_Header_constructor**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_Header_is_valid**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_Metadata_constructor**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock.**test_Metadata_set_state**(*dataspace*)

decisionengine.framework.dataspace.tests.test_datablock_zlib module

decisionengine.framework.dataspace.tests.test_datablock_zlib.**test_compress**()

decisionengine.framework.dataspace.tests.test_datablock_zlib.**test_zdumps**()

decisionengine.framework.dataspace.tests.test_datablock_zlib.**test_zloads**()

decisionengine.framework.dataspace.tests.test_datasource module

decisionengine.framework.dataspace.tests.test_datasource.**test_has_methods_we_expect**()

decisionengine.framework.dataspace.tests.test_dataspace module

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_dataspace_config_finds_bad`**`()`

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_duplicate_datablock`**`(`*dataspace*`)`
Can we duplicate taskmanager1 and all its entries

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_datablock`**`(`*dataspace*`)`
Can we get the datablock content

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_dataproduct`**`(`*dataspace*`)`
Can we get the dataproduct by uuid with key

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_dataproduct_not_exist`**`(`*dataspace*`)`
Does it error out if we ask for bogus information?

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_dataproducts`**`(`*dataspace*`)`
Can we get the dataproducts by uuid and uuid with key

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_dataproducts_not_exist`**`(`*dataspace*`)`
Does it error out if we ask for bogus information?

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_header`**`(`*dataspace*`)`
Can we fetch a header?

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_header_not_exist`**`(`*dataspace*`)`
Does it error out if we ask for a bogus header?

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_last_generation_id`**`(`*dataspace*`)`
Can we get the last generation id by name or name and uuid

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_last_generation_id_not_exist`**`(`*dataspace*`)`
Does it error out if we ask for a bogus taskmanager?

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_metadata`**`(`*dataspace*`)`
Can we fetch a metadata element?

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_metadata_not_exist`**`(`*dataspace*`)`
Does it error out if we ask for a bogus metadata element?

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_taskmanager_exists`**`(`*dataspace*`)`
Can I get a taskmanager by name or name and uuid

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_taskmanager_not_exists`**`(`*dataspace*`)`
This should error out

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_taskmanagers`**`(`*dataspace*`)`
Can I get multimple task managers

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_get_taskmanagers_not_exist`**`(`*dataspace*`)`
Do I error out when asking for garbage

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_has_config`**`(`*dataspace*`)`
verify our config entry exists

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_insert`**`(`*dataspace*`)`
Can we insert new elements

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_store_taskmanager`**`(`*dataspace*`)`
Can we make new entries

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_update`**`(`*dataspace*`)`
Do updates work as expected

`decisionengine.framework.dataspace.tests.test_dataspace.`**`test_update_bad`**`(`*dataspace*`)`
Do updates fail to work on bogus taskmanager as expected

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

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

`_setitem``(`*key*, *value*, *header*, *metadata=None*`)`
put a 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_dataproducts``(`*key=None*`)`

`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`

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_impl = <_abc._abc_data 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', 'schema_id', 'taskmanager_id'}

exception decisionengine.framework.dataspace.datablock.**InvalidMetadataError**
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_impl = <_abc._abc_data object>

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

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

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

class decisionengine.framework.dataspace.datablock.**ProductRetriever**(*product_name*, *product_type*, *product_source*)

Bases: object

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_impl = <_abc._abc_data 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_dataproducts`(*taskmanager_id*, *key*)
Return list of all data products associated with with taskmanager_id

Parameters **`key`** (`string`) – data product key

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

Parameters

- **`taskmanager_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

`abstract get_taskmanagers`(*taskmanager_name=None*, *start_time=None*, *end_time=None*)
Retrieve TaskManagers :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 **reset_connections**()
Drop any cached connections and reconnect to the database

abstract **store_taskmanager**(*taskmanager_name*, *taskmanager_id*, *datestamp=None*)
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 :type datestamp: `datetime` :arg datestamp: datetime of created object, defaults to 'now'

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.**DataSpace**(*config*)
Bases: `object`

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

close()

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

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

get_datablock(*taskmanager_id*, *generation_id*)

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

get_dataproducts(*taskmanager_id*, *key=None*)

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*)

get_taskmanagers(*taskmanager_name=None*, *start_time=None*, *end_time=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*, *taskmanager_id*, *datestamp=None*)

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

decisionengine.framework.dataspace.maintain module

class decisionengine.framework.dataspace.maintain.**Reaper**(*config*)

Bases: `object`

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

The class attributes indicate a rational set of defaults that shouldn't be altered by user configuration.

MIN_RETENTION_INTERVAL_DAYS = 7

MIN_SECONDS_BETWEEN_RUNS = 7080

_reaper_loop(*delay*)

The thread actually runs this.

reap()

Actually spawn the query to delete the old records. Lock the state as this task doesn't have a cancel option.

property retention_interval

We have data constraints, so use a property to track

property seconds_between_runs

We have data constraints, so use a property to track

start(*delay=0*)

Start thread with an optional delay to start the thread in X seconds

`stop()`
Try to stop the reaper, will block if the reaper cannot be interupted.

Module contents

decisionengine.framework.engine package

Subpackages

decisionengine.framework.engine.tests package

Submodules

decisionengine.framework.engine.tests.fixtures module

pytest defaults

`decisionengine.framework.engine.tests.fixtures.`**`DEServer`**`(`*conf_path=None*, *conf_override=None*, *channel_conf_path=None*, *channel_conf_override=None*, *host='127.0.0.1'*, *port=None*`)`
A DE Server using a private database

`decisionengine.framework.engine.tests.fixtures.`**`PG_DE_DB_WITHOUT_SCHEMA`**`(`*request: _pytest.fixtures.FixtureRequest*`)` → psycopg2.extensions.connection
Fixture factory for PostgreSQL.

Parameters **`request`** – fixture request object

Returns postgresql client

`decisionengine.framework.engine.tests.fixtures.`**`PG_DE_DB_WITH_SCHEMA`**`(`*PG_DE_DB_WITHOUT_SCHEMA*`)`
Load our PG schema into the database via this fixture so pytest knows the limitations on parallel usage of this database scope.

`decisionengine.framework.engine.tests.fixtures.`**`PG_PROG`**`(`*request: _pytest.fixtures.FixtureRequest*, *tmpdir_factory: _pytest.tmpdir.TempdirFactory*`)` → Iterator[pytest_postgresql.executor.PostgreSQLExecutor]
Process fixture for PostgreSQL.

Parameters **`request`** – fixture request object

Returns tcp executor

`decisionengine.framework.engine.tests.fixtures.`**`SQLALCHEMY_PG_WITH_SCHEMA`**`(`*PG_DE_DB_WITHOUT_SCHEMA*`)`
Get a blank database from pytest_postgresql. Then setup the SQLAlchemy style URL with that DB. The SQLAlchemyDS will create the schema as needed.

`decisionengine.framework.engine.tests.fixtures.`**`SQLALCHEMY_TEMPFILE_SQLITE`**`(`*tmp_path*`)`
Setup an SQLite database with the pytest tmp_path fixture. Then setup the SQLAlchemy style URL with that DB. The SQLAlchemyDS will create the schema as needed.

decisionengine.framework.engine.tests.test_client_only module

`decisionengine.framework.engine.tests.test_client_only.`**`test_client_err_returned_as_rc`**`()`
    no de server is running, so –status should error

`decisionengine.framework.engine.tests.test_client_only.`**`test_client_err_returned_verbose_as_rc`**`()`
    no de server is running, so –status should error

`decisionengine.framework.engine.tests.test_client_only.`**`test_client_help`**`(`*capfd*`)`

`decisionengine.framework.engine.tests.test_client_only.`**`test_client_with_no_command_says_use_help`**`()`

`decisionengine.framework.engine.tests.test_client_only.`**`test_client_with_no_server`**`()`

`decisionengine.framework.engine.tests.test_client_only.`**`test_client_with_no_server_verbose`**`()`

`decisionengine.framework.engine.tests.test_client_only.`**`test_exclusive_options`**`()`

decisionengine.framework.engine.tests.test_query_tool_only module

`decisionengine.framework.engine.tests.test_query_tool_only.`**`test_client_err_returned_as_rc`**`()`
    no de server is running, so –status should error

`decisionengine.framework.engine.tests.test_query_tool_only.`**`test_client_err_returned_verbose_as_rc`**`()`
    no de server is running, so –status should error

`decisionengine.framework.engine.tests.test_query_tool_only.`**`test_query_tool_help`**`()`

`decisionengine.framework.engine.tests.test_query_tool_only.`**`test_query_tool_with_no_server`**`()`

`decisionengine.framework.engine.tests.test_query_tool_only.`**`test_query_tool_with_no_server_verbose`**`()`

decisionengine.framework.engine.tests.test_startup module

`decisionengine.framework.engine.tests.test_startup.`**`_check_override`**`(`*arguments*`)`

`decisionengine.framework.engine.tests.test_startup.`**`test_change_port`**`()`

`decisionengine.framework.engine.tests.test_startup.`**`test_default_config`**`()`

Module contents

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`**`(`*global_config*, *channel_config_loader*, *server_address*`)`
    Bases: `socketserver.ThreadingMixIn`, `xmlrpc.server.SimpleXMLRPCServer`

    `_dataframe_to_column_names``(`*df*`)` `_dataframe_to_csv`(*df*)

`_dataframe_to_json`(*df*)

`_dataframe_to_table`(*df*)

`_dataframe_to_vertical_tables`(*df*)

`_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.

`block_until`(*state*, *timeout=None*)

`block_while`(*state*, *timeout=None*)

`get_logger`()

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

`reaper_start`(*delay*)

`reaper_status`()

`reaper_stop`()

`rm_channel`(*channel*, *maybe_timeout*)

`rpc_block_while`(*state_str*, *timeout=None*)

`rpc_get_channel_log_level`(*channel*)

`rpc_get_log_level`()

`rpc_kill_channel`(*channel*, *timeout=None*)

`rpc_print_product`(*product*, *columns=None*, *query=None*, *types=False*, *format=None*)

`rpc_print_products`()

`rpc_query_tool`(*product*, *format=None*, *start_time=None*)

`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_rm_channel`(*channel*, *maybe_timeout*)

`rpc_set_channel_log_level`(*channel*, *log_level*)
Assumes log_level is a string corresponding to the supported logging-module levels.

`rpc_show_config`(*channel*)
Show the configuration for a channel.

rpc_show_de_config()

rpc_start_channel(*channel_name*)

rpc_start_channels()

rpc_status()

rpc_stop()

rpc_stop_channel(*channel*)

rpc_stop_channels()

start_channel(*channel_name*, *channel_config*)

start_channels()

stop_channels()

stop_worker(*worker*, *timeout*)

class decisionengine.framework.engine.DecisionEngine.**RequestHandler**(*request*, *client_address*, *server*)

Bases: `xmlrpc.server.SimpleXMLRPCRequestHandler`

rpc_paths = ('/RPC2',)

class decisionengine.framework.engine.DecisionEngine.**StopState**(*value*)

Bases: `enum.Enum`

An enumeration.

Clean = 2

NotFound = 1

Terminated = 3

decisionengine.framework.engine.DecisionEngine.**_channel_preamble**(*name*)

decisionengine.framework.engine.DecisionEngine.**_create_de_server**(*global_config*, *channel_config_loader*)

Create the DE server with the passed global configuration and config manager

decisionengine.framework.engine.DecisionEngine.**_get_de_conf_manager**(*global_config_dir*, *channel_config_dir*, *options*)

decisionengine.framework.engine.DecisionEngine.**_get_global_config**(*config_file*, *options*)

decisionengine.framework.engine.DecisionEngine.**_start_de_server**(*server*)

Start the DE server and listen forever

decisionengine.framework.engine.DecisionEngine.**main**(*args=None*)

If args is None, sys.argv will be used instead If args is a list, it will be used instead of sys.argv (for unit testing)

decisionengine.framework.engine.DecisionEngine.**parse_program_options**(*args=None*)

If args is a list, it will be used instead of sys.argv

decisionengine.framework.engine.Workers module

`class decisionengine.framework.engine.Workers.`**`Worker`**(*task_manager*, *logger_config*)

Bases: `multiprocessing.context.Process`

Class that encapsulates a channel's task manager as a separate process.

This class' run function is called whenever the process is started. If the process is abruptly terminated–e.g. the run method is pre-empted by a signal or an os._exit(n) call–the Worker object will still exist even if the operating-system process no longer does.

To determine the exit code of this process, use the Worker.exitcode value, provided by the multiprocessing.Process base class.

`get_consumes`()

`get_produces`()

`get_state_name`()

`run`()

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

`wait_until`(*state*, *timeout=None*)

`wait_while`(*state*, *timeout=None*)

`class decisionengine.framework.engine.Workers.`**`Workers`**

Bases: `object`

This class manages and provides access to the task-manager workers.

The intention is that the decision engine never directly interacts with the workers but refers to them via a context manager:

with workers.access() as ws: # Access to ws now protected ws['new_channel'] = Worker(…)

In cases where the decision engine's block_while or block_until methods must be called (e.g. during tests), one should used the unguarded access:

with workers.unguarded_access() as ws: # Access to ws is unprotected ws['new_channel'].wait_until(…)

Calling a blocking method while using the protected context manager (i.e. workers.access()) will likely result in a deadlock.

`class` **`Access`**(*workers*, *lock*)

Bases: `object`

`_update_channel_states`()

`access`()

`unguarded_access`()

decisionengine.framework.engine.de_client module

`decisionengine.framework.engine.de_client.`**`console_scripts_main`**(*args_to_parse=None*)

This is the entry point for the setuptools auto generated scripts. Setuptools thinks a return from this function is an error message.

`decisionengine.framework.engine.de_client.`**`create_parser`**()

`decisionengine.framework.engine.de_client.`**`execute_command_from_args`**(*argsparsed*, *de_socket*)

argsparsed should be from create_parser in this file

`decisionengine.framework.engine.de_client.`**`main`**(*args_to_parse=None*)

If you pass a list of args, they will be used instead of sys.argv

decisionengine.framework.engine.de_query_tool module

`decisionengine.framework.engine.de_query_tool.`**`console_scripts_main`**(*args_to_parse=None*)

This is the entry point for the setuptools auto generated scripts. Setuptools thinks a return from this function is an error message.

`decisionengine.framework.engine.de_query_tool.`**`create_parser`**()

`decisionengine.framework.engine.de_query_tool.`**`execute_command_from_args`**(*argsparsed*, *de_socket*)

Calls the proper function for the arguments passed to de_query_tool.

Parameters

- **`argsparsed`** (*`Namespace`*) – Should be from create_parser in this file.
- **`de_socket`** (*`ServerProxy`*) – RPC Server Proxy.

Returns Output of the command.

Return type str

`decisionengine.framework.engine.de_query_tool.`**`main`**(*args_to_parse=None*)

Main function for de_query_tool

Parameters **`args_to_parse`** (*`list, optional`*) – If you pass a list of args, they will be used instead of sys.argv. Defaults to None.

Returns Query result

Return type str

Module contents

decisionengine.framework.logicengine package

Subpackages

decisionengine.framework.logicengine.tests package

Submodules

decisionengine.framework.logicengine.tests.test_cascaded_rules module

decisionengine.framework.logicengine.tests.test_cascaded_rules.**myengine**()

decisionengine.framework.logicengine.tests.test_cascaded_rules.**test_rule_that_does_not_fire**(*myengine*)

decisionengine.framework.logicengine.tests.test_cascaded_rules.**test_rule_that_fires**(*myengine*)

decisionengine.framework.logicengine.tests.test_construction module

decisionengine.framework.logicengine.tests.test_construction.**test_configuration_with_fact_using_functio**

decisionengine.framework.logicengine.tests.test_construction.**test_configuration_with_numy_facts**()

decisionengine.framework.logicengine.tests.test_construction.**test_default_construction**()
LogicEngine is not default constructible.

decisionengine.framework.logicengine.tests.test_construction.**test_trivial_configuration**()
Logic engine constructed with trivial rules and facts.

decisionengine.framework.logicengine.tests.test_construction.**test_wrong_configuration**()
LogicEngine construction requires rules and facts; if we don’t supply them it is an error.

decisionengine.framework.logicengine.tests.test_duplicate_fact_names module

decisionengine.framework.logicengine.tests.test_duplicate_fact_names.**test_duplicate_fact_names**()

decisionengine.framework.logicengine.tests.test_facts module

decisionengine.framework.logicengine.tests.test_facts.**make_db**(*maximum*)

decisionengine.framework.logicengine.tests.test_facts.**test_compound_fact_with_spaces**()

decisionengine.framework.logicengine.tests.test_facts.**test_fact_using_numpy_array**()

decisionengine.framework.logicengine.tests.test_facts.**test_fact_using_numpy_function**()

decisionengine.framework.logicengine.tests.test_facts.**test_fact_with_fail_on_error**()

decisionengine.framework.logicengine.tests.test_facts.**test_fact_with_nested_names**()

decisionengine.framework.logicengine.tests.test_facts.**test_simple_fact**()

decisionengine.framework.logicengine.tests.test_facts.**test_syntax_error**(*caplog*)

decisionengine.framework.logicengine.tests.test_fail_on_error module

decisionengine.framework.logicengine.tests.test_fail_on_error.**logic_engine_with_fact**(*fact*)

decisionengine.framework.logicengine.tests.test_fail_on_error.**test_conditional_fact**()

decisionengine.framework.logicengine.tests.test_fail_on_error.**test_fact_with_misspecified_attribute**()

decisionengine.framework.logicengine.tests.test_fail_on_error.**test_fail_on_error**(*caplog*)

decisionengine.framework.logicengine.tests.test_fail_on_error.**test_false_fact_with_spaces**()

decisionengine.framework.logicengine.tests.test_fail_on_error.**test_false_literal_fact**()

`decisionengine.framework.logicengine.tests.test_fail_on_error.`**`test_index_error`**`()`

`decisionengine.framework.logicengine.tests.test_fail_on_error.`**`test_misspecified_fact`**`()`

`decisionengine.framework.logicengine.tests.test_fail_on_error.`**`test_true_fact`**`()`

`decisionengine.framework.logicengine.tests.test_fail_on_error.`**`test_true_literal_fact`**`()`

decisionengine.framework.logicengine.tests.test_pandas_fact module

`decisionengine.framework.logicengine.tests.test_pandas_fact.`**`mydata`**`(`*y*`)`
Return a 'datablock' surrogate carrying a Pandas DataFrame, and a parameter named 'y' with value y.

`decisionengine.framework.logicengine.tests.test_pandas_fact.`**`myengine`**`()`

`decisionengine.framework.logicengine.tests.test_pandas_fact.`**`test_rule_that_does_not_fire`**`(`*myengine*`)`
Rules that do not fire do not create entries in the returned actions and newfacts.

`decisionengine.framework.logicengine.tests.test_pandas_fact.`**`test_rule_that_fires`**`(`*myengine*`)`

decisionengine.framework.logicengine.tests.test_rule_with_negated_fact module

`decisionengine.framework.logicengine.tests.test_rule_with_negated_fact.`**`myengine`**`()`

`decisionengine.framework.logicengine.tests.test_rule_with_negated_fact.`**`test_rule_that_does_not_fire`**`(`*mye*
Rules that do not fire do not create entries in the returned actions and newfacts.

`decisionengine.framework.logicengine.tests.test_rule_with_negated_fact.`**`test_rule_that_fires`**`(`*myengine*`)`

decisionengine.framework.logicengine.tests.test_simple_configuration module

`decisionengine.framework.logicengine.tests.test_simple_configuration.`**`myengine`**`()`

`decisionengine.framework.logicengine.tests.test_simple_configuration.`**`test_rule_that_does_not_fire`**`(`*myengi*
Rules that do not fire do not create entries in the returned actions and newfacts.

`decisionengine.framework.logicengine.tests.test_simple_configuration.`**`test_rule_that_fires`**`(`*myengine*`)`

Module contents

Submodules

decisionengine.framework.logicengine.BooleanExpression module

`class` `decisionengine.framework.logicengine.BooleanExpression.`**`BooleanExpression`**`(`*expr*`)`
Bases: `object`

`evaluate``(`*d*`)`
Return the evaluated Boolen value of this expression in the context of the given data 'd'.

`exception` `decisionengine.framework.logicengine.BooleanExpression.`**`LogicError`**
Bases: `TypeError`

`decisionengine.framework.logicengine.BooleanExpression.`**`function_name_from_call`**`(`*callnode*`)`

`decisionengine.framework.logicengine.BooleanExpression.`**`maybe_fail_on_error`**`(`*expr*`)`

decisionengine.framework.logicengine.FactLookup module

class `decisionengine.framework.logicengine.FactLookup.`**`FactLookup`**(*fact_names*, *rules_cfg*)

Bases: `object`

Establishes a policy for looking up a fact based on the given name.

To wit, the first fact with a given name is the one that is used in the evaluation of all subsequent facts.

As an example, consider the following configuration:

> **facts: {** should_publish: "(True)",
>
> }, rules: {
>
> > **publish_1: {** expression: "should_publish", facts: ["should_publish"]
> >
> > }, publish_2: {
> >
> > > expression: "should_publish", actions: ["go_to_press"] facts: ["should_publish"]
> >
> > } retract: {
> >
> > > expression: "not should_publish", facts: ["should_retract"]
>
> }

In the above, the first fact to be evaluated will always be the top-level facts (i.e. those not encapsulated by the 'rules' table). The rules labeled 'publish_1' and 'publish_2' both rely on the 'should_publish' fact in their expressions, and they in turn create their own facts with the same name. FactLookup ensures that 'publish_1' and 'publish_2' will both use the evaluated fact from the top-level 'facts' table.

`rule_for`(*fact_name*)

Selects rule required to evaluate fact with the supplied name.

Parameters **`fact_name`** (*`str`*) – Name of fact for which rule will be selected.

Return type str

Returns Rule name

`sorted_rules`(*rules_cfg*)

Rules sorted according to rule dependencies.

Parameters **`rules_cfg`** (*`dict`*) – rules as specified in logic-engine configuration

Return type list

Returns Rules to be evaluated by the rule engine.

decisionengine.framework.logicengine.LogicEngine module

class `decisionengine.framework.logicengine.LogicEngine.`**`LogicEngine`**(*cfg*)

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

`_create_facts_dataframe`(*newfacts*)

Convert newfacts dict in format below to dataframe with columns ['rule_name', 'fact_name', fact_value']

facts dict format: 'newfacts': {

> **'publish_glidein_requests': {** 'allow_hpc_new': True, 'allow_foo': True
>
> }, 'dummy_rule': {
>
> > 'dummy_new_fact': True

        }

    }

`consumes()`
Return the names of all the items that must be in the DataBlock for the rules to be evaluated.

`evaluate(`*db*`)`
Evaluate our facts and rules, in the context of the given data. db can be any mappable, in particular a DataBlock or dictionary.

Parameters `db` (`DataBlock`) – Products used to evaluate facts.

`evaluate_facts(`*db*`)`

Parameters `db` (`DataBlock`) – Products used to evaluate facts.

Return type dict

Returns Evaluated fact values (e.g. True or False) for each fact name.

`produces()`

`decisionengine.framework.logicengine.LogicEngine.passthrough_configuration(`*publisher_names*`)`
Assembles logic-engine configuration to unconditionally execute all publishers.

decisionengine.framework.logicengine.Rule module

`class decisionengine.framework.logicengine.Rule.Rule(`*rule_name*, *rule_cfg*`)`
Bases: `object`

In-memory representation of logic-engine rule, relying on parsing utilities in BooleanExpression.

`evaluate(`*evaluated_facts*`)`
Evaluates a compiled expression given the supplied facts.

Parameters `evaluated_facts` (*dict*) – Initial fact values (e.g. True or False) for each fact name.

Return type bool

decisionengine.framework.logicengine.RuleEngine module

`class decisionengine.framework.logicengine.RuleEngine.RuleEngine(`*fact_names*, *rules_cfg*`)`
Bases: `object`

Engine responsible for evaluating logic-engine rules.

This class is responsible for (a) forming a sorted set of rules that supports dependencies between them, and (b) evaluating the rules according to a specified fact-lookup policy.

`execute(`*evaluated_facts*`)`
Evaluates all rules given the supplied facts.

Parameters `evaluated_facts` (*dict*) – Initial fact values (e.g. True or False) for each fact name.

Return type tuple

Returns Actions to be taken based on rule evaluation; new facts produced during that evaluation.

Module contents

decisionengine.framework.managers package

Submodules

decisionengine.framework.managers.ChannelManager module

Channel Manager

class decisionengine.framework.managers.ChannelManager.**Channel**(*channel_dict*)

Bases: `object`

Decision Channel. Instantiates runners according to channel configuration

class decisionengine.framework.managers.ChannelManager.**ChannelManager**(*name*, *dataspace*, *generation_id*, *channel_dict*, *global_config*, *subscribe_queue*, *channel_subscribed*)

Bases: *decisionengine.framework.managers.ComponentManager.ComponentManager*

Channel Manager: Runs decision cycle for transforms and publishers

check_for_new_data_products()

Check the incoming data product queue, insert into datablock, and update source_new_data appropriately

decision_cycle()

Decision cycle to be run periodically (by trigger)

do_backup()

Duplicate current data block and return its copy

Return type `DataBlock`

register_with_sources(*channel_id*, *channel_name*, *all_sources*)

Sends registration information for the sources that the channel is interested in to the SourceSubscription-Manager

reset_source_flags(*sources*)

Sets self.source_new_data to False for the sources which are being used by this decision cycle

Parameters **sources** (`list`) – list of source names that are to be set back to the "not-updated" state

run()

Channel 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_transform(*transform*, *data_block*)

Run a transform

Parameters

- **transform** (*TaskRunner*) – source TaskRunner
- **data_block** (DataBlock) – data block

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

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

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_all_sources(*channel_sources*)
Wait for all sources this channel is interested in to finish their execution

Parameters **channel_sources** (list) – list of sources that need to be polled for completion

wait_for_any_source(*channel_sources*)
Waits for any source this channel is interested in to post an updated data block before allowing continuation

Parameters **channel_sources** (list) – list of sources to be watched for updated data

wait_for_registration(*channel_id*)

class decisionengine.framework.managers.ChannelManager.**TaskRunner**(*conf_dict*)
Bases: object

Provides interface to loadable modules and events for synchronization of execution

decisionengine.framework.managers.ChannelManager.**_make_runners_for**(*configs*)

decisionengine.framework.managers.ComponentManager module

Decision Engine ComponentManager (Base class for ChannelManager and SourceManager)

class decisionengine.framework.managers.ComponentManager.**ComponentManager**(*name*, *generation_id*, *global_config*)
Bases: object

Base class for decisionengine components such as Sources and Channels

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

get_loglevel()

get_state()

get_state_name()

get_state_value()

set_loglevel_value(*log_level*)

Assumes log_level is a string corresponding to the supported logging-module levels.

take_offline(*current_data_block*)

offline and stop this component manager

decisionengine.framework.managers.ComponentManager.**create_runner**(*module_name*, *class_name*, *parameters*)

Create instance of dynamically loaded module

decisionengine.framework.managers.SourceManager module

Source Manager

class decisionengine.framework.managers.SourceManager.**Source**(*name*, *source_dict*)

Bases: `object`

Decision Source. Instantiates Source runners according to the provided Source configuration

class decisionengine.framework.managers.SourceManager.**SourceManager**(*name*, *generation_id*, *source_config*, *global_config*, *data_block_queue*)

Bases: `decisionengine.framework.managers.ComponentManager.ComponentManager`

Source Manager: Runs decision cycle for transforms and publishers

data_block_send(*source_name*, *source_id*, *data*, *header*)

run()

class decisionengine.framework.managers.SourceManager.**SourceRunner**(*conf_dict*)

Bases: `object`

Provides interface to loadable modules and events for synchronization of execution

decisionengine.framework.managers.SourceSubscriptionManager module

Source Subscription Manager

class
decisionengine.framework.managers.SourceSubscriptionManager.**SourceSubscriptionManager**

Bases: `threading.Thread`

This implements the communication between Sources and Channels

get_new_subscriptions()

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

send_data_product_to_subscribed(*new_source_info*) class decisionengine.framework.managers.SourceSubscriptionManager.**Subscription**(*channel_manager_id*, *channel_manager_name*, *source_names*)

Bases: object

Module contents

decisionengine.framework.modules package

Subpackages

decisionengine.framework.modules.tests package

Submodules

decisionengine.framework.modules.tests.test_Module module

decisionengine.framework.modules.tests.test_Module.**test_module_structure**()

The module.Module itself is a bit of a skeleton…

decisionengine.framework.modules.tests.test_Publisher module

decisionengine.framework.modules.tests.test_Publisher.**test_publisher_structure**()

The module.publisher itself is a bit of a skeleton…

decisionengine.framework.modules.tests.test_Source module

decisionengine.framework.modules.tests.test_Source.**test_source_structure**()

The module.Source itself is a bit of a skeleton…

decisionengine.framework.modules.tests.test_Transform module

decisionengine.framework.modules.tests.test_Transform.**test_transform_structure**()

The module.Transform itself is a bit of a skeleton…

decisionengine.framework.modules.tests.test_de_logger module

decisionengine.framework.modules.tests.test_de_logger.**log_setup**()

decisionengine.framework.modules.tests.test_de_logger.**test_by_nonsense_is_err**(*log_setup*)

decisionengine.framework.modules.tests.test_de_logger.**test_by_size**(*log_setup*)

decisionengine.framework.modules.tests.test_de_logger.**test_by_time**(*log_setup*)

decisionengine.framework.modules.tests.test_module_decorators module

decisionengine.framework.modules.tests.test_module_decorators.**test_multiple_consumes_declarations**()

decisionengine.framework.modules.tests.test_module_decorators.**test_multiple_produces_declarations**()

decisionengine.framework.modules.tests.test_module_decorators.**test_supports_config**()

decisionengine.framework.modules.tests.test_module_decorators.**test_wrong_product_names**()

decisionengine.framework.modules.tests.test_module_decorators.**test_wrong_product_types**()

decisionengine.framework.modules.tests.test_translate_product_name module

decisionengine.framework.modules.tests.test_translate_product_name.**test_translate_all**()

decisionengine.framework.modules.tests.test_translate_product_name.**test_translate_illegal_characters**()

decisionengine.framework.modules.tests.test_translate_product_name.**test_translate_none**()

decisionengine.framework.modules.tests.test_translate_product_name.**test_translate_simple**()

decisionengine.framework.modules.tests.test_translate_product_name.**test_translate_with_underscores**()

Module contents

Submodules

decisionengine.framework.modules.Module module

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

Bases: object

A skeleton of a module

get_data_block()

get_parameters()

set_data_block(*data_block*)

decisionengine.framework.modules.Module.**consumes**(***kwargs*)

decisionengine.framework.modules.Module.**produces**(***kwargs*)

decisionengine.framework.modules.Module.**verify_products**(*producer*, *data*)

decisionengine.framework.modules.Publisher module

class decisionengine.framework.modules.Publisher.**Parameter**(*name*, *type=None*, *default=None*, *comment=None*)

Bases: object

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

Bases: decisionengine.framework.modules.Module.Module

_consumes = {}

publish(*data_block=None*)

shutdown()

decisionengine.framework.modules.Publisher.**consumes**(***kwargs*)

decisionengine.framework.modules.Publisher.**describe**(*cls*, *program_options=<class 'decisionengine.framework.modules.describe.ModuleProgramOptions'>*)

decisionengine.framework.modules.Publisher.**supports_config**(**args*)

decisionengine.framework.modules.Source module

class decisionengine.framework.modules.Source.**Parameter**(*name*, *type=None*, *default=None*, *comment=None*)

Bases: `object`

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

Bases: `decisionengine.framework.modules.Module.Module`

_produces = {}

acquire()

post_create(*global_config*)

decisionengine.framework.modules.Source.**describe**(*cls*, *sample_config=None*)

decisionengine.framework.modules.Source.**produces**(***kwargs*)

decisionengine.framework.modules.Source.**supports_config**(**args*)

decisionengine.framework.modules.SourceProxy module

Fill in data from another channel data block

class decisionengine.framework.modules.SourceProxy.**SourceProxy**(*config*)

Bases: `decisionengine.framework.modules.Source.Source`

_get_data(*data_block*, *key*)

_supported_config = {'Dataproducts': (<class 'list'>, None, 'List of data products to retrieve.'), 'retries': (<class 'int'>, 10, 'Number of attempts allowed to fetch products.'), 'retry_timeout': (<class 'int'>, 60, 'Number of seconds to wait between retries.'), 'source_channel': (<class 'str'>, None, 'Channel from which to retrieve data products.')}

acquire()

Overrides Source class method

post_create(*global_config*)

decisionengine.framework.modules.Transform module

class decisionengine.framework.modules.Transform.**Parameter**(*name*, *type=None*, *default=None*, *comment=None*)

Bases: object

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

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

_consumes = {}

_produces = {}

transform()

decisionengine.framework.modules.Transform.**consumes**(***kwargs*)

decisionengine.framework.modules.Transform.**describe**(*cls*, *program_options=<class 'decisionengine.framework.modules.describe.ModuleProgramOptions'>*)

decisionengine.framework.modules.Transform.**produces**(***kwargs*)

decisionengine.framework.modules.Transform.**supports_config**(**args*)

decisionengine.framework.modules.de_logger module

Logger to use in all modules

decisionengine.framework.modules.de_logger.**_reset_config**()

Reset the logconf.pylogconfig dictionary

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

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

decisionengine.framework.modules.de_logger.**set_logging**(*log_level*, *file_rotate_by*, *rotation_time_unit='D'*, *rotation_interval=1*, *max_backup_count=6*, *max_file_size=200000000*, *log_file_name='/tmp/decision_engine_logs/decisionengine.log'*

Parameters

- **log_level** (str) – log level
- **file_rotate_by** – files rotation by size or by time
- **rotation_time_unit** (str) – unit of time for file rotation
- **rotation_interval** (int) – time in rotation_time_units between file rotations
- **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 None

decisionengine.framework.modules.describe module

class decisionengine.framework.modules.describe.**ModuleProgramOptions**(*module_spec*, *cls*)

Bases: object

process_args()

class decisionengine.framework.modules.describe.**Parameter**(*name*, *type=None*, *default=None*, *comment=None*)

Bases: object

decisionengine.framework.modules.describe.**_par_default**(*par_type*, *default_value*)

decisionengine.framework.modules.describe.**_par_type**(*par_type*, *default_value*)

decisionengine.framework.modules.describe.**main_wrapper**(*cls*, *program_options=<class 'decisionengine.framework.modules.describe.ModuleProgramOptions'*

decisionengine.framework.modules.describe.**supports_config**(**args*)

decisionengine.framework.modules.logging_configDict module

Global Logger config dictionary used by all loggers (in their own subkeys)

decisionengine.framework.modules.print_description module

decisionengine.framework.modules.print_description.**_print_comment**(*comment*)

decisionengine.framework.modules.print_description.**_print_type**(*type_or_value*)

decisionengine.framework.modules.print_description.**_print_value**(*v*)

decisionengine.framework.modules.print_description.**_spec_from_file_name**(*filename*)

decisionengine.framework.modules.print_description.**print_consumes**(*cls*)

decisionengine.framework.modules.print_description.**print_produces**(*cls*)

decisionengine.framework.modules.print_description.**print_supported_config**(*module_spec*, *cls*)

decisionengine.framework.modules.print_description.**spec_if_main**(*cls*)

decisionengine.framework.modules.translate_product_name module

decisionengine.framework.modules.translate_product_name.**translate**(*spec*)

Break apart the string 'old -> new' into a tuple ('old', 'new')

decisionengine.framework.modules.translate_product_name.**translate_all**(*specs*)

Module contents

decisionengine.framework.taskmanager package

Submodules

decisionengine.framework.taskmanager.ProcessingState module

The ProcessingState class can represent any of the following task-manager states:

> BOOT IDLE ACTIVE STEADY OFFLINE SHUTTINGDOWN SHUTDOWN ERROR

In addition, the class supports 'wait_until(state)' and 'wait_while(state)' methods, which, when called from a different process, block until the state has been entered or exited, respectively.

The 'RUNNING_CONDITIONS' list is a list of states that a thread may have if it is started/starting. The 'STOPPING_CONDITIONS' list is a list of states that a thread may have if it is stopped/stopping. The 'INACTIVE_CONDITIONS' list is a list of states that a thread may have when it is not active

`class decisionengine.framework.taskmanager.ProcessingState.`**`ProcessingState`**`(`*state=State.BOOT*`)`

Bases: `object`

This object tracks the state of a process.

A number of convience wrappers are provided.

Additionally you may use the *.lock* attribute for *with* block to lock the state during specific operations.

`get()`

This function is a minimally locking check to fetch the state.

`get_state_value()`

`has_value``(`*state*`)`

`inactive()`

`property lock`

`probably_running()`

`set``(`*state*`)`

This function will lock (and possibly block) to ensure a consistent change to the state value.

This function can be blocked using the *.lock* to force state sync between threads if need be.

`should_stop()`

`wait_until``(`*state*, *timeout=None*`)`

`wait_while``(`*state*, *timeout=None*`)`

`class decisionengine.framework.taskmanager.ProcessingState.`**`State`**`(`*value*`)`

Bases: `enum.Enum`

An enumeration.

`ACTIVE = 2`

`BOOT = 0`

`ERROR = 7`

`IDLE = 1`

`OFFLINE = 6`

`SHUTDOWN = 5`

`SHUTTINGDOWN = 4`

`STEADY = 3`

decisionengine.framework.taskmanager.TaskManager module

Task Manager

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

Bases: `object`

Decision Channel. Instantiates workers according to channel configuration

class `decisionengine.framework.taskmanager.TaskManager.`**`TaskManager`**(*name*, *generation_id*, *channel_dict*, *global_config*)

Bases: *`decisionengine.framework.managers.ComponentManager.ComponentManager`*

Task Manager

`decision_cycle`()

Decision cycle to be run periodically (by trigger)

`do_backup`()

Duplicate current data block and return its copy

Return type `DataBlock`

`get_consumes`()

`get_produces`()

`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_to_shutdown()

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

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

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*, *base_class*, *channel_name*)

Bases: `object`

Provides interface to loadable modules an events to sycronise execution

decisionengine.framework.taskmanager.TaskManager.**_create_module_instance**(*config_dict*, *base_class*, *channel_name*)

Create instance of dynamically loaded module

decisionengine.framework.taskmanager.TaskManager.**_find_only_one_subclass**(*module*, *base_class*)
Search through module looking for only one subclass of the supplied base_class

decisionengine.framework.taskmanager.TaskManager.**_make_workers_for**(*configs*, *base_class*, *channel_name*)

decisionengine.framework.taskmanager.module_graph module

Ensure no circularities in produces and consumes.

decisionengine.framework.taskmanager.module_graph.**_consumed_products**(**worker_lists*)

decisionengine.framework.taskmanager.module_graph.**_produced_products**(**worker_lists*)

decisionengine.framework.taskmanager.module_graph.**ensure_no_circularities**(*sources*, *transforms*, *publishers*)

Ensures no circularities among data products.

Module contents

decisionengine.framework.tests package

Submodules

decisionengine.framework.tests.ABTransform module

class decisionengine.framework.tests.ABTransform.**ABTransform**(*module_parameters*, **args*, ***kwargs*)

Bases: *decisionengine.framework.modules.Transform.Transform* `_consumes = {'B': None}`

`_produces = {'A': None}`

decisionengine.framework.tests.BATransform module

class decisionengine.framework.tests.BATransform.**BATransform**(*module_parameters*, **args*, ***kwargs*)

Bases: *decisionengine.framework.modules.Transform.Transform*

`_consumes = {'A': None}`

`_produces = {'B': None}`

decisionengine.framework.tests.ErrorOnAcquire module

class decisionengine.framework.tests.ErrorOnAcquire.**ErrorOnAcquire**(*config*)

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

`_produces = {'_placeholder':  None}`

acquire()

decisionengine.framework.tests.FailingPublisher module

class decisionengine.framework.tests.FailingPublisher.**FailingPublisher**(*module_parameters*, **args*, ***kwargs*)

Bases: *decisionengine.framework.modules.Publisher.Publisher*

`_consumes = {'bar':  None}`

publish(*data_block*)

decisionengine.framework.tests.FailingSourceNOP module

class decisionengine.framework.tests.FailingSourceNOP.**SourceWithMissingProduces**(*set_of_parameters*)

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

decisionengine.framework.tests.FailingSourceProxy module

class decisionengine.framework.tests.FailingSourceProxy.**FailingSourceProxy**(*config*)

Bases: *decisionengine.framework.modules.SourceProxy.SourceProxy*

acquire()

Overrides Source class method

decisionengine.framework.tests.ModuleProgramOptions module

class decisionengine.framework.tests.ModuleProgramOptions.**AcquireWithConfig**(*name*)

Bases: object

test(*byte_str*, *expected_stderr=''*)

class decisionengine.framework.tests.ModuleProgramOptions.**AcquireWithSampleConfig**(*name*)

Bases: object

test()

class decisionengine.framework.tests.ModuleProgramOptions.**ConfigTemplate**(*name*)

Bases: object

test(*has_comments=False*)

class decisionengine.framework.tests.ModuleProgramOptions.**Describe**(*name*)

Bases: object

test(*consumes=None*, *produces=None*)

class decisionengine.framework.tests.ModuleProgramOptions.**DescribeAlias**(*alias*, *original*)

Bases: object

test()

class decisionengine.framework.tests.ModuleProgramOptions.**Help**(*name*)

Bases: object

test(*has_sample_config=False*)

decisionengine.framework.tests.ModuleProgramOptions.**_expected_acquire_result**(*name*, *config_file=None*, *multiplier=1*, *channel_name='test1'*)

decisionengine.framework.tests.ModuleProgramOptions.**_expected_config_template**(*name*)

decisionengine.framework.tests.ModuleProgramOptions.**_expected_config_template_with_comments**(*name*)

decisionengine.framework.tests.ModuleProgramOptions.**_expected_help**(*name*)

decisionengine.framework.tests.ModuleProgramOptions.**_expected_source_help**(*name*, *has_sample_config=False*)

decisionengine.framework.tests.ModuleProgramOptions.**_normalize**(*string*)

decisionengine.framework.tests.ModuleProgramOptions.**_run_as_main**(*name*, **program_options*)

decisionengine.framework.tests.PublisherNOP module

class decisionengine.framework.tests.PublisherNOP.**PublisherNOP**(*module_parameters*, **args*, ***kwargs*)

Bases: *decisionengine.framework.modules.Publisher.Publisher*

_consumes = {'bar': <class 'pandas.core.frame.DataFrame'>}

publish(*data_block*)

decisionengine.framework.tests.PublisherWithMissingConsumes module

class decisionengine.framework.tests.PublisherWithMissingConsumes.**PublisherWithMissingConsumes**(*set_of_para*

Bases: *decisionengine.framework.modules.Publisher.Publisher*

decisionengine.framework.tests.SourceAlias module

decisionengine.framework.tests.SourceNOP module

class decisionengine.framework.tests.SourceNOP.**SourceNOP**(*config*)

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

_produces = {'foo': <class 'pandas.core.frame.DataFrame'>}

acquire()

decisionengine.framework.tests.SourceWithSampleConfigNOP module

class decisionengine.framework.tests.SourceWithSampleConfigNOP.**SourceWithSampleConfigNOP**(*config*)

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

_produces = {'foo': <class 'pandas.core.frame.DataFrame'>}

_supported_config = {'channel_name': (<class 'str'>, None, None), 'multiplier': (<class 'int'>, None, None)}

acquire()

decisionengine.framework.tests.SupportsConfigPublisher module

class decisionengine.framework.tests.SupportsConfigPublisher.**SupportsConfig**(*set_of_parameters*)

Bases: *decisionengine.framework.modules.Publisher.Publisher*

_supported_config = {'comment': (<class 'str'>, None, 'Single-line comment'), 'comment_with_nl': (<class 'str'>, None, 'Comment with newline\n'), 'convert_to': (<class 'int'>, 3, None), 'default_only': (<class 'float'>, 2.5, None), 'no_type': (None, None, None), 'only_type': (<class 'int'>, None, None)}

decisionengine.framework.tests.TransformNOP module

class decisionengine.framework.tests.TransformNOP.**TransformNOP**(*module_parameters*, **args*, ***kwargs*)

Bases: *decisionengine.framework.modules.Transform.Transform*

_consumes = {'foo': <class 'pandas.core.frame.DataFrame'>}

_produces = {'bar': <class 'pandas.core.frame.DataFrame'>}

transform(*data_block*)

decisionengine.framework.tests.TransformWithMissingProducesConsumes module

class decisionengine.framework.tests.TransformWithMissingProducesConsumes.**TransformWithMissingProducesC**

Bases: *decisionengine.framework.modules.Transform.Transform*

transform(*data_block*)

decisionengine.framework.tests.WorkingSourceProxy module

class decisionengine.framework.tests.WorkingSourceProxy.**WorkingSourceProxy**(*config*)

Bases: *decisionengine.framework.modules.SourceProxy.SourceProxy*

acquire()

Overrides Source class method

decisionengine.framework.tests.fixtures module

defaults for pytest

decisionengine.framework.tests.fixtures.**DEServer**(*conf_path=None*, *conf_override=None*, *channel_conf_path=None*, *channel_conf_override=None*, *host='127.0.0.1'*, *port=None*)

A DE Server using a private database

decisionengine.framework.tests.fixtures.**PG_DE_DB_WITHOUT_SCHEMA**(*request: _pytest.fixtures.FixtureRequest*) → psycopg2.extensions.connection

Fixture factory for PostgreSQL.

Parameters **request** – fixture request object

Returns postgresql client

decisionengine.framework.tests.fixtures.**PG_DE_DB_WITH_SCHEMA**(*PG_DE_DB_WITHOUT_SCHEMA*)

Load our PG schema into the database via this fixture so pytest knows the limitations on parallel usage of this database scope.

decisionengine.framework.tests.fixtures.**PG_PROG**(*request: _pytest.fixtures.FixtureRequest*, *tmpdir_factory: _pytest.tmpdir.TempdirFactory*) → Iterator[pytest_postgresql.executor.PostgreSQLExecutor]

Process fixture for PostgreSQL.

Parameters **request** – fixture request object

Returns tcp executor

decisionengine.framework.tests.fixtures.**SQLALCHEMY_PG_WITH_SCHEMA**(*PG_DE_DB_WITHOUT_SCHEMA*)

Get a blank database from pytest_postgresql. Then setup the SQLAlchemy style URL with that DB. The SQLAlchemyDS will create the schema as needed.

decisionengine.framework.tests.fixtures.**SQLALCHEMY_TEMPFILE_SQLITE**(*tmp_path*)

Setup an SQLite database with the pytest tmp_path fixture. Then setup the SQLAlchemy style URL with that DB. The SQLAlchemyDS will create the schema as needed.

decisionengine.framework.tests.test_client_errors module

Fixture based DE Server tests of the sample config

`decisionengine.framework.tests.test_client_errors.`**`test_client_cannot_wait_on_bad_state`**`(`*deserver*`)`
   Verify wait is for a valid state

decisionengine.framework.tests.test_client_server module

Fixture based DE Server for the de-client tests

`decisionengine.framework.tests.test_client_server.`**`test_client_print_product`**`(`*deserver*`)`

`decisionengine.framework.tests.test_client_server.`**`test_client_print_product_columns`**`(`*deserver*`)`

`decisionengine.framework.tests.test_client_server.`**`test_client_print_product_columns_query`**`(`*deserver*`)`

`decisionengine.framework.tests.test_client_server.`**`test_client_print_product_json`**`(`*deserver*`)`

`decisionengine.framework.tests.test_client_server.`**`test_client_print_product_not_real`**`(`*deserver*`)`

`decisionengine.framework.tests.test_client_server.`**`test_client_print_product_not_string`**`(`*deserver*`)`
   Make sure the public API is protected against bad values

`decisionengine.framework.tests.test_client_server.`**`test_client_print_product_query`**`(`*deserver*`)`

`decisionengine.framework.tests.test_client_server.`**`test_client_print_product_types`**`(`*deserver*`)`

`decisionengine.framework.tests.test_client_server.`**`test_client_print_product_vertical`**`(`*deserver*`)`

`decisionengine.framework.tests.test_client_server.`**`test_client_status_msg_to_stdout`**`(`*deserver*`)`
   Make sure the actuall client console call goes to stdout

decisionengine.framework.tests.test_defaults module

Fixture based DE Server tests of the sample config

`decisionengine.framework.tests.test_defaults.`**`test_client_can_get_de_server_show_channel_logger_level`**`(`*des*
   Verify unknown channel has NOTSET

`decisionengine.framework.tests.test_defaults.`**`test_client_de_config_is_json`**`(`*deserver*`)`
   Verify config can be fetched in json format

`decisionengine.framework.tests.test_defaults.`**`test_global_channel_log_level_in_config`**`(`*deserver*`)`
   Verify global_channel_log_level setting exists

decisionengine.framework.tests.test_error_on_acquire module

`decisionengine.framework.tests.test_error_on_acquire.`**`test_source_only_channel`**`(`*deserver*`)`

decisionengine.framework.tests.test_module_program_options module

decisionengine.framework.tests.test_module_program_options.**test_acquire_for_sources**()

decisionengine.framework.tests.test_module_program_options.**test_config_templates**()

decisionengine.framework.tests.test_module_program_options.**test_descriptions**()

decisionengine.framework.tests.test_module_program_options.**test_help**()

decisionengine.framework.tests.test_module_program_options.**test_module_alias**()

decisionengine.framework.tests.test_query_tool_server module

Fixture based DE Server for the de-query-tool tests

decisionengine.framework.tests.test_query_tool_server.**test_query_tool_csv**(*deserver*)

decisionengine.framework.tests.test_query_tool_server.**test_query_tool_default**(*deserver*)

decisionengine.framework.tests.test_query_tool_server.**test_query_tool_invalid_product**(*deserver*)

decisionengine.framework.tests.test_query_tool_server.**test_query_tool_json**(*deserver*)

decisionengine.framework.tests.test_query_tool_server.**test_query_tool_since**(*deserver*)

decisionengine.framework.tests.test_reaper module

Fixture based DE Server for the reaper tests

decisionengine.framework.tests.test_reaper.**test_client_can_get_de_server_reaper_start_delay**(*deserver*)
Verify reaper can start with delay

decisionengine.framework.tests.test_reaper.**test_client_can_get_de_server_reaper_status**(*deserver*)
Verify reaper status

decisionengine.framework.tests.test_reaper.**test_client_can_get_de_server_reaper_stop**(*deserver*)
Verify reaper can stop

decisionengine.framework.tests.test_restart_channel module

decisionengine.framework.tests.test_restart_channel.**deserver_mock_data_block**(*mock_data_block*)

decisionengine.framework.tests.test_restart_channel.**test_restart_channel**(*deserver_mock_data_block*)

decisionengine.framework.tests.test_sample_config module

Fixture based DE Server tests of the defaults

decisionengine.framework.tests.test_sample_config.**test_client_can_double_set_de_server_channel_log_level**
Verify set log level to current level isn't an error

decisionengine.framework.tests.test_sample_config.**test_client_can_get_de_server_channel_config**(*deserver*)
Verify config has expected items

decisionengine.framework.tests.test_sample_config.**test_client_can_get_de_server_channel_log_level**(*deserv*
Verify can fetch log level for a channel

decisionengine.framework.tests.test_sample_config.**test_client_can_get_de_server_show_config**(*deserver*)
Verify config has expected items

decisionengine.framework.tests.test_sample_config.**test_client_can_get_de_server_show_logger_level**(*deserv*
Verify can fetch log level

decisionengine.framework.tests.test_sample_config.**test_client_can_get_de_server_status**(*deserver*)
Verify channel enters stable state

decisionengine.framework.tests.test_sample_config.**test_client_can_get_products**(*deserver*)
Verify client can get channel products

decisionengine.framework.tests.test_sample_config.**test_client_can_get_products_no_channels**(*deserver*)
Verify client can get channel products even when none are run

decisionengine.framework.tests.test_sample_config.**test_client_can_kill_one_channel**(*deserver*)
Verify client can kill a single channel

decisionengine.framework.tests.test_sample_config.**test_client_can_kill_one_channel_force**(*deserver*)
Verify client can kill a single channel with force

decisionengine.framework.tests.test_sample_config.**test_client_can_kill_one_channel_timeout**(*deserver*)
Verify client can kill a single channel with timeout

decisionengine.framework.tests.test_sample_config.**test_client_can_set_de_server_channel_log_level**(*deserv*
Verify set log level for a channel

decisionengine.framework.tests.test_sample_config.**test_client_can_start_all_channel**(*deserver*)
Verify client can start all channel

decisionengine.framework.tests.test_sample_config.**test_client_can_start_one_channel**(*deserver*)
Verify client can start a single channel

decisionengine.framework.tests.test_sample_config.**test_client_can_stop_channels**(*deserver*)
Verify client can stop channels

decisionengine.framework.tests.test_sample_config.**test_client_can_stop_one_channel**(*deserver*)
Verify client can stop a single channel

decisionengine.framework.tests.test_sample_config.**test_client_can_stop_server**(*deserver*)
Verify de-client can run –stop

decisionengine.framework.tests.test_sample_config.**test_client_cannot_double_start**(*deserver*)
Verify client cannot double start channels

decisionengine.framework.tests.test_sample_config.**test_client_get_channel_log_fails_cleanly**(*deserver*)
Verify graceful fail on bogus channel

decisionengine.framework.tests.test_sample_config.**test_client_get_non_real_channel**(*deserver*)
Verify config for missing channel does what it should

decisionengine.framework.tests.test_sample_config.**test_client_set_channel_log_fails_cleanly**(*deserver*)
Verify graceful fail on bogus channel

decisionengine.framework.tests.test_sample_config.**test_client_start_non_real_channel**(*deserver*)
Verify start for missing channel does what it should

decisionengine.framework.tests.test_sample_config.**test_client_stop_non_real_channel**(*deserver*)
Verify stop for missing channel does what it should

decisionengine.framework.tests.test_sample_config.**test_client_wait_timeout_works**(*deserver*)
Verify channel enters stable state and timeout works too

decisionengine.framework.tests.test_source_proxy module

Fixture based tests of the SourceProxy module.

`decisionengine.framework.tests.test_source_proxy.`**`test_stop_failing_source_proxy`**(*deserver_fail*)

`decisionengine.framework.tests.test_source_proxy.`**`test_working_source_proxy`**(*deserver*)

decisionengine.framework.tests.test_start_with_bad_channels module

Fixture based DE Server tests of invalid channel configs

`decisionengine.framework.tests.test_start_with_bad_channels.`**`_consumes_not_subset`**(*test_str*)

`decisionengine.framework.tests.test_start_with_bad_channels.`**`_expected_circularity`**(*test_str*)

`decisionengine.framework.tests.test_start_with_bad_channels.`**`_missing_consumes`**(*name*)

`decisionengine.framework.tests.test_start_with_bad_channels.`**`_missing_produces`**(*name*)

`decisionengine.framework.tests.test_start_with_bad_channels.`**`test_client_can_get_products_no_channels`**(*des*
cap

Verify client can get channel products even when none are run

decisionengine.framework.tests.test_start_with_no_channels module

Fixture based DE Server tests of the server without channels, then with them

`decisionengine.framework.tests.test_start_with_no_channels.`**`deserver_mock_data_block`**(*mock_data_block*)

`decisionengine.framework.tests.test_start_with_no_channels.`**`test_start_from_nothing`**(*deserver_mock_data_bloc*

Module contents

decisionengine.framework.util package

Submodules

decisionengine.framework.util.fs module

`decisionengine.framework.util.fs.`**`files_with_extensions`**(*dir_path*, **extensions*)

Return all files in dir_path that match the provided extensions.

If no extensions are given, then all files in dir_path are returned.

Results are sorted by channel name to ensure stable output.

decisionengine.framework.util.metrics module

class decisionengine.framework.util.metrics.**Counter**(*name*, *documentation*, *labelnames=()*, *namespace=''*, *subsystem=''*, *unit=''*, *registry=<prometheus_client.registry.CollectorRegistry object>*, *_labelvalues=None*)

Bases: `prometheus_client.metrics.Counter`

class decisionengine.framework.util.metrics.**Gauge**(**args*, ***kwargs*)

Bases: `prometheus_client.metrics.Gauge`

Override prometheus client Gauge so that muliproccess_mode 'liveall" is the default as opposed to 'all'

_DEFAULT_MULTIPROC_MODE = 'liveall'

__determine_multiprocess_mode_existence(**args*, ***kwargs*)

class decisionengine.framework.util.metrics.**Histogram**(*name*, *documentation*, *labelnames=()*, *namespace=''*, *subsystem=''*, *unit=''*, *registry=<prometheus_client.registry.CollectorRegistry object>*, *_labelvalues=None*, *buckets=(0.005, 0.01, 0.025, 0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1.0, 2.5, 5.0, 7.5, 10.0, inf)*)

Bases: `prometheus_client.metrics.Histogram`

class decisionengine.framework.util.metrics.**Summary**(*name*, *documentation*, *labelnames=()*, *namespace=''*, *subsystem=''*, *unit=''*, *registry=<prometheus_client.registry.CollectorRegistry object>*, *_labelvalues=None*)

Bases: `prometheus_client.metrics.Summary`

decisionengine.framework.util.reaper module

A stand-alone script purges data in database older than specified in configuration. Configuration file has to have this bit added:

{

"dataspace" [{ "retention_interval_in_days"][365,]

"datasource" : { … }

}

}

Can be used in a cron job.

decisionengine.framework.util.reaper.**main**()

decisionengine.framework.util.singleton module

class decisionengine.framework.util.singleton.**ScopedSingleton**

Bases: *decisionengine.framework.util.singleton.Singleton*

Singleton pattern using Metaclass with weak refs

_instances = <WeakValueDictionary>

class decisionengine.framework.util.singleton.**ScopedSingletonABC**(*name*, *bases*, *namespace*, ***kwargs*)

Bases: abc.ABCMeta, *decisionengine.framework.util.singleton.ScopedSingleton*

class decisionengine.framework.util.singleton.**Singleton**

Bases: type

Singleton pattern using Metaclass with strong refs

_instances = {}

class decisionengine.framework.util.singleton.**SingletonABC**(*name*, *bases*, *namespace*, ***kwargs*)

Bases: abc.ABCMeta, *decisionengine.framework.util.singleton.Singleton*

decisionengine.framework.util.sockets module

decisionengine.framework.util.sockets.**get_random_port**()

decisionengine.framework.util.subclasses module

decisionengine.framework.util.subclasses.**_derived_class**(*cls*, *base_class*)

Only matches subclasses that are not equal to the base class.

decisionengine.framework.util.subclasses.**all_subclasses**(*module*, *base_class*)

Return all of a module’s subclasses of the given base class.

Module contents

Submodules

decisionengine.framework.about module

PEP-0396 provides instructions for providing module versions While we are at it, add a few other useful bits

decisionengine.framework.version module

Module contents

decisionengine.tests package

Submodules

decisionengine.tests.test_framework_package module

Make sure decisionengine.framework is a valid python package

`decisionengine.tests.test_framework_package.`**`test_can_import`**`()`

Module contents

Module contents

4.2 Indices and tables

- genindex
- modindex
- search

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX