Eskapade Core Documentation

KPMG Advanced Analytics
Big Data team

Feb 23, 2019

Contents

- Version: 1.0.0
- Released: Jan 2018

Eskapade is a light-weight, python-based data analysis framework, meant for modularizing all sorts of data analysis problems into reusable analysis components.

For the full documentation on Eskapade, including many examples, please go to this link.

The core functionality of Eskapade, namely: the `Link`, `Chain`, `process_manager`, `DataStore`, `ConfigObject` and corresponding tutorials, has now been split off from the growing Eskapade repository, into this new package Eskapade-Core.

For the minimal documentation on Eskapade-Core, please go here.

CHAPTER 1

Release notes

1.1 Version 1.0.0

In Eskapade-Core v1.0.0:

- Eskapade goes parallel. The execution of chains can be forked, so chains can run in parallel! See tutorial esk112 for details.
- The eskapade_bootstrap method has been severely upgraded. Running this command will generate you a new (Eskapade) project directory with a working link, macro, tests, entry-point scripts, and setup.py file!
- escore.eskapade_run() has been made more user-friendly for standalone use with python or jupyter.
- added __main__ function to every newly generated macro, so macros can now be run standalone. In addition, eskapade_generate_macro now makes a python function in the macro.
- The Eskapade DbConnection base class has been migraged to core.
- The DataStore now has a fancy get() function that can assert the type, length, and presence of object.
- Added a new link `ApplyFunc` that applies function to objects in the datastore. Simple but very useful.
- Plus several small updates to existing links in core_ops: `skip_chain_if_empty`, `import_data_store`, `BreakLink`.

1.2 Version 0.9

Version 0.9 of Eskapade-Core (December 2018) is a split off of the `core` and `core_ops` modules of Eskapade v0.9 into a separate package. Eskapade v0.9 builds on top of Eskapade-Core, and focussed on analysis modules.

CHAPTER 2

Installation

2.1 requirements

Eskapade-Core works standalone and is a very light-weight Python3 package, and requires `Python 3.6+`.

2.2 pypi

To install the package from pypi, do:

```
$ pip install Eskapade-Core
```

2.3 github

Alternatively, you can check out the repository from github and install it yourself:

```
$ git clone https://github.com/KaveIO/Eskapade-Core.git eskapade-core
```

To (re)install the python code from your local directory, type from the top directory:

```
$ pip install -e eskapade-core
```

2.4 python

After installation, you can now do in Python:

```
import escore
```

Congratulations, you are now ready to use Eskapade!

CHAPTER 3

Quick run

To see the available examples in Eskapade-Core, do:

```
$ export TUTDIRC=`pip show Eskapade-Core | grep Location | awk '{ print $2"/escore/
↪tutorials" }'`
$ ls -l $TUTDIRC/
```

E.g. you can now run:

```
$ eskapade_run $TUTDIRC/esk101_helloworld.py
```

This documentation here is minimal on purpose. For all examples on using Eskapade links, chains and the DataStore to set up an analysis work flow, please see the Eskapade tutorials section.

For more examples, see the full Eskapade documentation.

CHAPTER 4

Contact and support

Contact us at: kave [at] kpmg [dot] com

Please note that the KPMG Eskapade group provides support only on a best-effort basis.

CHAPTER 5

Contents

5.1 Tutorials

This section briefly describes how to run Eskapade-Core. All command examples can be run from any directory with write access. For more in depth explanations on the functionality of the code-base, try the API docs.

5.1.1 The examples in Eskapade Core

All Eskapade-Core example macros can be found in the tutorials directory. For ease of use, let's make a shortcut to the directory containing the tutorials:

```
$ export TUTDIRC=`pip show Eskapade-Core | grep Location | awk '{ print $2"/escore/
↪tutorials" }'`
$ ls -l $TUTDIRC/
```

The numbering of the example macros is as follows:

- **esk100+: basic macros describing the chains, links, and datastore functionality of Eskapade.** They explain the basic architecture of Eskapade, i.e. how the chains, links, datastore, and process manager interact.

These macros are briefly described below. You are encouraged to run all examples to see what they can do for you!

For all Eskapade tutorial examples, please go to the Eskapade tutorials section at read-the-docs.

Example: Hello World!

Macro 101 runs the Hello World Link. It runs the Link twice using a repeat kwarg, showing how to use kwargs in Links.

```
$ eskapade_run $TUTDIRC/esk101_helloworld.py
```

5.2 Developing and Contributing

5.2.1 Working on Eskapade-Core

You have some cool feature and/or algorithm you want to add to Eskapade. How do you go about it?

First clone Eskapade.

```
git clone https://github.com/KaveIO/Eskapade-Core.git eskapade-core
```

then

```
pip install -e eskapade-core
```

this will install Eskapade-Core in editable mode, which will allow you to edit the code and run it as you would with a normal installation of eskapade.

To make sure that everything works try executing eskapade without any arguments, e.g.

```
eskapade_run
```

or you could just execute the tests using either the eskapade test runner, e.g.

```
cd eskapade-core
eskapade_trial .
```

or

```
cd eskapade
python setup.py test
```

That's it.

5.2.2 Contributing

When contributing to this repository, please first discuss the change you wish to make via issue, email, or any other method with the owners of this repository before making a change. You can find the contact information on the index page.

Note that when contributing that all tests should succeed.

5.2.3 Tips and Tricks

- Enable auto reload in ipython:

```
%load_ext autoreload
```

this will reload modules before executing any user code.

5.3 References

- Web page: https://eskapade.readthedocs.io

- Repository: https://github.com/kaveio/eskapade
- Docker: https://github.com/kaveio/eskapade-environment
- Issues & Ideas: https://github.com/kaveio/eskapade/issues
- Eskapade: http://eskapade.kave.io
- Contact us at: kave [at] kpmg [dot] com

5.4 API

5.4.1 API Documentation

Eskapade-Core

escore package

Subpackages

escore.core package

Submodules

escore.core.definitions module

Project: Eskapade - A python-based package for data analysis.

Created: 2017/02/27

Description: Definitions used in Eskapade runs: * logging levels * return-status codes * default configuration variables * user options

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

Redistribution and use in source and binary forms, with or without modification, are permitted according to the terms listed in the file LICENSE.

class escore.core.definitions.**RandomSeeds**(***kwargs*)

Bases: object

Container for seeds of random generators.

Seeds are stored as key-value pairs and are accessed with getitem and setitem methods. A default seed can be accessed with the key "default". The default seed is also returned if no seed is set for the specified key.

```
>>> import numpy as np
>>> seeds = RandomSeeds(default=999, foo=42, bar=13)
>>> seeds['NumPy'] = 100
>>> np.random.seed(seeds['NumPy'])
>>> print(seeds['nosuchseed'])
999
```

__init__(***kwargs*)

Initialize an instance.

Values of the specified keyword arguments must be integers, which are set as seed values for the corresponding key.

class escore.core.definitions.**StatusCode**
Bases: enum.IntEnum

Return status code enumeration class.

A StatusCode should be returned by the initialize, execute, and finalize methods of links, chains, and the process manager.

The enumerations are:

- Undefined (-1): Default status.
- Success (0 == EX_OK / EXIT_SUCCESS): All OK, i.e. there were no errors.
- RepeatChain (1): Repeat execution of this chain.
- SkipChain (2): Skip this chain: initialize, execute, and finalize.
- BreakChain (3): Skip the further execution of this this, but do perform finalize.
- Recoverable (4): Not OK, but can continue, i.e. there was an error, but the application can recover from it.
- Failure (5): An error occurred and the application cannot recover from it. In this case the application should just quit.

BreakChain = 3

Failure = 5

Recoverable = 4

RepeatChain = 1

SkipChain = 2

Success = 0

Undefined = -1

escore.core.definitions.**set_begin_end_chain_opt**(*opt_key*, *settings*, *args*)
Set begin/end-chain variable from user option.

escore.core.definitions.**set_custom_user_vars**(*opt_key*, *settings*, *args*)
Set custom user configuration variables.

escore.core.definitions.**set_log_level_opt**(*opt_key*, *settings*, *args*)
Set configuration log level from user option.

escore.core.definitions.**set_opt_var**(*opt_key*, *settings*, *args*)
Set configuration variable from user options.

escore.core.definitions.**set_seeds**(*opt_key*, *settings*, *args*)
Set random seeds.

escore.core.definitions.**set_single_chain_opt**(*opt_key*, *settings*, *args*)
Set single-chain variable from user option.

escore.core.element module

Project: Eskapade - A python-based package for data analysis.

Created: 2017/02/27

Description: Base classes for the building blocks of an Eskapade analysis run:

- Link:
- Chain:

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core.element.**Chain**(*name*, *process_manager=None*)

Bases: *escore.core.meta.Processor*, *escore.core.meta.ProcessorSequence*, *escore.core.mixin.TimerMixin*

Execution Chain.

A Chain object contains a collection of links with analysis code. The links in a chain are executed in the order in which the links have been added to the chain. Typically a chain contains all links related to one topic, for example 'validation of a model', or 'data preparation', or 'data quality checks'.

```
>>> from escore import process_manager
>>> from escore import Chain
>>> from escore import analysis
>>>
>>> # Create an IO chain. This is automatically registered with the process␣
↪manager.
>>> io_chain = Chain('Overview')
```

And Links are added to a chain as follows:

```
>>> # add a link to the chain
>>> io_chain.add(analysis.ReadToDf(path='foo.csv', key='foo'))
>>>
>>> # Run everything.
>>> process_manager.run()
```

__init__(*name*, *process_manager=None*)

Initialize chain.

add(*link: escore.core.element.Link*) → None

Add a link to the chain.

Parameters **link** (Link) – The link to add to the chain.

Raises

- **TypeError** – When the link is of an incompatible type.
- **KeyError** – When a Link of the same type and name already exists.

clear()

Clear the chain.

discard(*link: escore.core.element.Link*) → None

Remove a link from the chain.

Parameters **link** (Link) –

Raises **KeyError** – When the processor does not exist.

execute() → escore.core.definitions.StatusCode

Execute links in chain.

Returns Execution status code.

Return type *StatusCode*

finalize() → escore.core.definitions.StatusCode
Finalize links and chain.

Returns Finalization status code.

Rtype StatusCode

get(*link_name: str*) → escore.core.element.Link
Find the link with the given name.

Parameters **link_name** (*str*) – Find a link with the given name.

Returns The chain.

Return type *Chain*

Raises **ValueError** – When the given chain name cannot be found.

initialize() → escore.core.definitions.StatusCode
Initialize chain and links.

Returns Initialization status code.

Return type *StatusCode*

n_links
Return the number of links in the chain.

Returns The number of links in the chain.

Return type int

class escore.core.element.**Link**(*name=None*)
Bases: *escore.core.meta.Processor*, *escore.core.mixin.ArgumentsMixin*, *escore.core.mixin.TimerMixin*

Link base class.

A link defines the content of an algorithm. Any actual link is derived from this base class.

A link usually does three things: - takes data from the datastore - does something to it - writes data back

To take from the data store there is a simple function load() To write to the data store there is a simple function store()

Links are added to a chain as follows:

```
>>> from escore import process_manager
>>> from escore import analysis
>>>
>>> # Create a Chain instance. Note that the chain is automatically registered␣
↪with process manager.
>>> io_chain = Chain('IO')
>>>
>>> # Add a link to the chain
>>> reader = analysis.ReadToDf(name='CsvReader', key='foo')
>>> reader.path = 'foo.csv'
>>> io_chain.add(reader)
>>>
>>> # Run everything.
>>> process_manager.run()
```

__init__(*name=None*)
Initialize link.

execute() → escore.core.definitions.StatusCode
Execute the Link.

This method may be overridden by the user.

Returns Status code.

Return type *StatusCode*

finalize() → escore.core.definitions.StatusCode
Finalize the Link.

This method may be overridden by the user.

Returns Status code.

Return type *StatusCode*

initialize() → escore.core.definitions.StatusCode
Initialize the Link.

This method may be overridden by the user.

Returns Status code.

Type StatusCode

load(*ds*, *read_key=None*)
Read all data from specified source.

Read_key can either be:

- one Link: return statuscode, [data_from_link,...]
- A list of locations: return statuscode, [data,...]
- A list of links with only one output location: return statuscode, [data,...]
- A list of links with multiple output locations: return statuscode, [data,[moredata]...]
- Any mixture of the above

Do something logical with a statuscode if this data does not exist link.if_input_missing = statuscode

Returns a tuple statuscode, [data in same order as read_key]

Return type (*StatusCode*,list)

run() → escore.core.definitions.StatusCode
Initialize, execute, and finalize the Link in one go.

This method is useful for testing purposes, e.g. when developing and testing functionality of a link standalone and interactively.

It is not used internally by Eskapade, where the functions are called individually by the chain, and all links are initialized together before their common execution, and all links in the chain are also finalized together, after their common execution.

Returns Status code.

Return type *StatusCode*

store(*ds*, *data*, *store_key=None*, *force=False*)
Store data back to datastore.

Do something logical with a statuscode if this data already exists link.if_output_exists = statuscode uses self.store_key. If self.store_key is a list of locations, I must sent a list of the same length here

summary()
Print a summary of the main settings of the link.

escore.core.exceptions module

Project: Eskapade - A python-based package for data analysis.

Created: 2016/11/08

Description: Eskapade exceptions.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

exception escore.core.exceptions.**Error**
Bases: Exception

Base class for all Eskapade core exceptions.

exception escore.core.exceptions.**UnknownSetting**
Bases: *escore.core.exceptions.Error*

The user requested an unknown setting.

escore.core.execution module

Project: Eskapade - A python-based package for data analysis.

Created: 2016/11/08

Description: Functions for running and resetting Eskapade machinery

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

escore.core.execution.**eskapade_configure**(*settings=None*)
Configure Eskapade.

This function is called by the eskapade_run function (below), to configure eskapade before running:

- set the configuration object
- set logging level
- set matplotlib backend
- process config file

Parameters **settings** – analysis settings. Of type ConfigObject or string, where the string is the macro path.

escore.core.execution.**eskapade_run**(*settings=None*)
Run Eskapade.

This function is called in the script eskapade_run when run from the cmd line. The working principle of Eskapade is to run chains of custom code chunks (so-called links).

Each chain should have a specific purpose, for example pre-processing incoming data, booking and/or training predictive algorithms, validating these predictive algorithms, evaluating the algorithms.

By using this principle, links can be easily reused in future projects.

Parameters **settings** (ConfigObject) – analysis settings

Returns status of the execution

Return type *StatusCode*

escore.core.execution.**reset_eskapade**(*skip_config=False*)
Reset Eskapade objects.

Parameters **skip_config** (*bool*) – skip reset of configuration object

escore.core.meta module

Project: Eskapade - A python-based package for data analysis.

Created: 2017/09/14

Description:

A collection of (generic) meta classes for some (design) patterns:

- Singleton: Meta class for the Singleton pattern.
- Processor: Meta class with abstract methods initialize, execute, and finalize.
- ProcessorSequence: A simple (processor) sequence container.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core.meta.**Processor**(*name: str*)
Bases: object

Processor metaclass.

__init__(*name: str*)
Initialize the Processor object.

execute()
Execution logic for processor.

finalize()
Finalization logic for processor.

initialize()
Initialization logic for processor.

logger
A logger that emits log messages to an observer.

The logger can be instantiated as a module or class attribute, e.g.

```
>>> logger = Logger()
>>> logger.info("I'm a module logger attribute.")
>>>
>>> class Point(object):
>>>     logger = Logger()
>>>
>>>     def __init__(self, x = 0.0, y = 0.0):
>>>         Point.logger.debug('Initializing {point} with x = {x}  y = {y}',␣
↪point=Point, x=x, y=y)
>>>         self._x = x
>>>         self._y = y
>>>
>>>     @property
>>>     def x(self):
>>>         self.logger.debug('Getting property x = {point._x}', point=self)
>>>         return self._x
>>>
>>>     @x.setter
>>>     def x(self, x):
>>>         self.logger.debug('Setting property y = {point._x}', point=self)
>>>         self._x = x
>>>
>>>     @property
>>>     def y(self):
>>>        self.logger.debug('Getting property y = {point._y}', point=self)
>>>        return self._y
>>>
>>>     @y.setter
>>>     def y(self, y):
>>>         self.logger.debug('Setting property y = {point._y}', point=self)
>>>         self._y = y
>>>
>>> a_point = Point(1, 2)
>>>
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
>>> logger.log_level = LogLevel.DEBUG
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
```

The logger uses PEP-3101 (Advanced String Formatting) with named placeholders, see <https://www.python.org/dev/peps/pep-3101/> and <https://pyformat.info/> for more details and examples.

Furthermore, logging events are only formatted and evaluated for logging levels that are enabled. So, there's no need to check the logging level before logging. It's also efficient.

name

Get the name of processor.

Returns The name of the processor.

Return type str

parent

Get the group parent.

Returns The parent/group processor sequence.

class escore.core.meta.**ProcessorSequence**

Bases: object

A doubly linked processor sequence.

It remembers the order in which processors are added to the sequence. It also checks if a processor already has been added to the sequence.

__init__()
Initialize the ProcessorSequence object.

add(*processor: escore.core.meta.Processor*) → None
Add a processor to the sequence.

Parameters **processor** (Processor) – The processor to add.

Raises **KeyError** – When a processor of the same type and name already exists.

clear() → None
Clear the sequence.

discard(*processor: escore.core.meta.Processor*) → None
Remove a processor from the sequence.

Parameters **processor** (Processor) – The processor to remove.

Raises **KeyError** – When the processor does not exist.

pop(*last: bool = True*) → escore.core.meta.Processor
Return the popped processor. Raise KeyError if empty.

By default a processor is popped from the end of the sequence.

Parameters **last** (*bool*) – Pop processor from the end of the sequence. Default is True.

Returns The pop processor.

Raises **KeyError** – When trying to pop from an empty list.

class escore.core.meta.**Singleton**
Bases: type

Metaclass for singletons.

Any instantiation of a Singleton class yields the exact same object, e.g.:

```
>>> class Klass(metaclass=Singleton):
>>>     pass
>>>
>>> a = Klass()
>>> b = Klass()
>>> a is b
True
```

See https://michaelgoerz.net/notes/singleton-objects-in-python.html.

escore.core.mixin module

Project: Eskapade - A python-based package for data analysis.

Created: 2016/11/08

Classes: ArgumentsMixin, TimerMixin

Description:

Mixin classes:

- ArgumentsMixin: processes/checks arguments and sets them as attributes

- TimerMixin: keeps track of execution time
- ConfigMixin: reads and handles settings from configuration files

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core.mixin.**ArgumentsMixin**
Bases: object

Mixin base class for argument parsing.

Class allows attributes to be accessed as dict items. Plus several argument processing helper functions.

check_arg_callable(**arg_names*, *allow_none=False*)
Check if set of arguments has iterators.

check_arg_iters(**arg_names*, *allow_none=False*)
Check if set of arguments has iterators.

check_arg_opts(*allow_none=False*, ***name_vals*)
Check if argument values are in set of options.

check_arg_types(*recurse=False*, *allow_none=False*, ***name_type*)
Check if set of arguments has correct types.

check_arg_vals(**arg_names*, *allow_none=False*)
Check if set of arguments exists as attributes and values.

check_extra_kwargs(*kwargs*)
Check for residual kwargs.

check_required_args(**arg_names*)
Check if set of arguments exists as attributes.

class escore.core.mixin.**ConfigMixin**(*config_path=None*)
Bases: object

Mixin base class for configuration settings.

__init__(*config_path=None*)
Initialize config settings.

Parameters **config_path** (*str*) – path of configuration file

config_path
Path of configuration file.

get_config(*config_path=None*)
Get settings from configuration file.

Read and return the configuration settings from a configuration file. If the path of this file is not specified as an argument, the value of the "config_path" property is used. If the file has already been read, return previous settings.

Parameters **config_path** (*str*) – path of configuration file

Returns configuration settings read from file

Return type configparser.ConfigParser

Raises **RuntimeError** – if config_path is not set

reset_config()
Remove previously read settings.

class escore.core.mixin.**TimerMixin**
Bases: object

Mixin base class for timing.

__init__()
Initialize timer.

start_timer()
Start run timer.

Start the timer. The timer is used to compute the run time. The returned timer start value has an undefined reference and should, therefore, only be compared to other timer values.

Returns start time in seconds

Return type float

stop_timer(*start_time=None*)
Stop the run timer.

Stop the timer. The timer is used to compute the run time. The elapsed time since the timer start is returned.

Parameters **start_time** (*float*) – function start_time input

Returns time difference with start in seconds

Return type float

total_time()
Return the total run time.

Returns total time in seconds

Return type float

escore.core.persistence module

Project: Eskapade - A python-based package for data analysis.

Created: 2016/11/08

Description: Utility class and functions to get correct io path, used for persistence of results

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

escore.core.persistence.**create_dir**(*dir_path*)
Create directory.

Parameters **dir_path** (*str*) – directory path

escore.core.persistence.**io_dir**(*io_type*, *io_conf=None*)
Construct directory path.

Parameters

- **io_type** (*str*) – type of result to store, e.g. data, macro, results.
- **io_conf** – IO configuration object

Returns directory path

Return type str

escore.core.persistence.**io_path**(*io_type*, *sub_path*, *io_conf=None*)
Construct directory path with sub path.

Parameters

- **io_type** (*str*) – type of result to store, e.g. data, macro, results.
- **sub_path** (*str*) – sub path to be included in io path
- **io_conf** – IO configuration object

Returns full path to directory

Return type str

escore.core.persistence.**record_file_number**(*file_name_base*, *file_name_ext*, *io_conf=None*)
Get next prediction-record file number.

Parameters

- **file_name_base** (*str*) – base file name
- **file_name_ext** (*str*) – file name extension
- **io_conf** – I/O configuration object

Returns next prediction-record file number

Return type int

escore.core.persistence.**repl_whites**(*name*)
Replace whitespace in names.

escore.core.process_manager module

Project: Eskapade - A python-based package for data analysis.

Class: ProcessManager

Created: 2016/11/08

Description: The ProcessManager class is the heart of Eskapade. It performs initialization, execution, and finalization of analysis chains.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core.process_manager.**ProcessManager**
Bases: *escore.core.meta.Processor*, *escore.core.meta.ProcessorSequence*, *escore.core.mixin.TimerMixin*

Eskapade run process manager.

ProcessManager is the event processing loop of Eskapade. It initializes, executes, and finalizes the analysis chains. There is, under normal circumstances, only one ProcessManager instance.

Here's an simple but illustrative analysis example:

```
>>> from escore import process_manager, Chain, Link, StatusCode
>>>
>>> # A chain automatically registers itself with process_manager.
>>> one_plus_one_chain = Chain('one_plus_one')
>>>
>>> class OnePlusOne(Link):
>>>     def execute(self):
>>>         self.logger.info('one plus one = {result}', result=(1+1))
>>>         return StatusCode.Success
>>>
>>> one_plus_one_chain.add(link=OnePlusOne())
>>>
>>> two_plus_two_chain = Chain('two_plus_two')
>>>
>>> class TwoPlusTwo(Link):
>>>     def execute(self):
>>>         self.logger.info('two plus two = {result}', result=(2+2))
>>>         return StatusCode.Success
>>>
>>> two_plus_two_chain.add(TwoPlusTwo())
>>>
>>> process_manager.run()
```

Ideally the user will not need to interact directly with the process manager. The magic is taken care of by the eskapade_run entry point.

__init__()
Initialize ProcessManager instance.

add(*chain: escore.core.element.Chain*) → None
Add a chain to the process manager.

Parameters **chain** (Chain) – The chain to add to the process manager.

Raises

- **TypeError** – When the chain is of an incompatible type.
- **KeyError** – When a chain of the same type and name already exists.

clear()
"Clear/remove all chains.

execute()
Execute all chains in order.

Returns status code of execution attempt

Return type *StatusCode*

execute_macro(*filename*, *copyfile=True*)
Execute an input python configuration file.

A copy of the configuration file is stored for bookkeeping purposes.

Parameters

- **filename** (*str*) – the path of the python configuration file
- **copyfile** (*bool*) – back up the macro for bookkeeping purposes

Raises **Exception** – if input configuration file cannot be found

finalize()
Finalize the process manager manager.

Returns status code of finalize attempt

Return type *StatusCode*

get(*chain_name: str*) → escore.core.element.Chain
Find the chain with the given name.

Parameters **chain_name** (*str*) – Find a chain with the given name.

Returns The chain.

Return type *Chain*

Raises **ValueError** – When the given chain name cannot be found.

get_service_tree()
Create tree of registered process-service classes.

Returns service tree

Return type dict

get_services()
Get set of registered process-service classes.

Returns service set

Return type set

import_services(*io_conf*, *chain=None*, *force=None*, *no_force=None*)
Import process services from files.

Parameters

- **io_conf** (*dict*) – I/O config as returned by ConfigObject.io_conf
- **chain** (*str*) – name of chain for which data was persisted
- **force** (*bool or list*) – force import if service already registered
- **no_force** (*list*) – do not force import of services in this list

initialize()
Initialize the process manager.

Initializes the process manager by configuring its chains. After initialization the configuration is printed.

Returns status code of initialize attempt

Return type *StatusCode*

n_chains
Return the number of chains in the process manager.

Returns The number of links in the chain.

Return type int

persist_services(*io_conf*, *chain=None*)
Persist process services in files.

Parameters

- **io_conf** (*dict*) – I/O config as returned by ConfigObject.io_conf
- **chain** (*str*) – name of chain for which data is persisted

print_chains()
Print all chains defined in the manager.

print_services()
Print registered process services.

remove_all_services()
Remove all registered process services.

remove_service(*service_cls*, *silent=False*)
Remove specified process service.

Parameters

- **service_cls** (ProcessServiceMeta) – service to remove
- **silent** (*bool*) – don't complain if service is not registered

reset()
Reset the process manager.

Resetting comprises removing the chains and closing any open connections/sessions.

run() → escore.core.definitions.StatusCode
Run process manager.

Returns Status code of run execution.

Return type *StatusCode*

service(*service_spec*)
Get or register process service.

Parameters **service_spec** (ProcessServiceMeta *or* ProcessService) – class (instance) to register

Returns registered instance

Return type *ProcessService*

summary()
Print process-manager summary.

Print a summary of the chains, links, and some analysis settings defined in this configuration.

escore.core.process_services module

Project: Eskapade - A python-based package for data analysis.

Created: 2017/02/27

Description: Base class and core implementations of run-process services

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core.process_services.**ConfigObject**
Bases: *escore.core.process_services.ProcessService*

Configuration settings for Eskapade.

The ConfigObject is a dictionary meant for containing global settings of Eskapade. Settings are set in the configuration macro of an analysis, or on the command line.

The ConfigObject is a dictionary meant only for storing global settings of Eskapade. In general, it is accessed through the process manager.

Example usage:

```
>>> # first set logging output level.
>>> from escore.logger import Logger, LogLevel
>>> logger = Logger()
>>> logger.log_level = LogLevel.DEBUG
```

Obtain the ConfigObject from any location as follows:

```
>>> from escore import process_manager
>>> from escore import ConfigObject
>>> settings = process_manager.service(ConfigObject)
```

One can treat the ConfigObject as any other dictionary:

```
>>> settings['foo'] = 'bar'
>>> foo = settings['foo']
```

Write the ConfigObject to a pickle file with:

```
>>> settings.persist_in_file(file_path)
```

And reload from the pickle file with:

```
>>> settings = ConfigObject.import_from_file(file_path)
```

A ConfigObject pickle file can be read in by Eskapade with the command line option (-u).

class IoConfig(***input_config*)
Bases: dict

Configuration object for I/O operations.

__init__(***input_config*)
Initialize IoConfig instance.

Print()
Print a summary of the settings.

__init__()
Initialize ConfigObject instance.

add_macros(*macro_paths*)
Add configuration macros for Eskapade run.

copy()
Perform a shallow copy of self.

Returns copy

get(*setting: str*, *default: Any = None*) → object
Get value of setting. If it does not exists return the default value.

Parameters

- **setting** – The setting to get.
- **default** – The default value of the setting.

Returns The value of the setting or None if it does not exist.

io_base_dirs() → dict
Get configured base directories.

Returns base directories

Return type dict

io_conf()
Get I/O configuration.

The I/O configuration contains storage locations and basic analysis info.

Returns I/O configuration

Return type *IoConfig*

set_user_opts(*parsed_args*)
Set options specified by user on command line.

Parameters **parsed_args** (*argparse.Namespace*) – parsed user arguments

class escore.core.process_services.**DataStore**
Bases: *escore.core.process_services.ProcessService*, dict

Store for transient data sets and related objects.

The data store is a dictionary meant for storing transient data sets or any other objects. Links can take one or several data sets as input, transform them or use them as input for a model, and store the output back again in the datastore, to be picked up again by any following link.

Example usage:

```
>>> # first set logging output level.
>>> from escore.logger import Logger, LogLevel
>>> logger = Logger()
>>> logger.log_level = LogLevel.DEBUG
```

Obtain the global datastore from any location as follows:

```
>>> from escore import process_manager
>>> from escore import DataStore
>>>
>>>
>>> ds = process_manager.service(DataStore)
```

One can treat the datastore as any other dict:

```
>>> ds['a'] = 1
>>> ds['b'] = 2
>>> ds['0'] = 3
>>> a = ds['a']
```

Write the datastore to a pickle file with:

```
>>> ds.persist_in_file(file_path)
```

And reload from the pickle file with:

```
>>> ds = DataStore.import_from_file(file_path)
```

Print()
Print a summary the data store contents.

get(*key: str*, *default: Any = None*, *assert_type: Any = None*, *assert_len: bool = False*, *assert_in: bool = False*) → object
Get value of setting. If it does not exists return the default value.

Parameters

- **key** – The key of object to get.
- **default** – The default value of the key in case not found.
- **assert_type** – if set, check object for given type or tuple of types. If fails, raise TypeError.
- **assert_len** – if true, check that object has length greater than 0. If fails, raise TypeError or AssertionError.
- **assert_in** – if true, assert that key is known.

Returns The value of the key or None if it does not exist.

class escore.core.process_services.**ForkStore**
Bases: *escore.core.process_services.ProcessService*

Dict for sharing objects between forked processes.

The ForkStore is a dictionary meant for sharing data sets or any other objects between forked processed. During execute, links in the same chain can take one or several data sets as input, transform them or use them as input for a model, and store the output back again, to be picked up again by another forked process. The ForkStore will not be persisted.

Example usage:

Obtain the global forkstore from any location as follows:

```
>>> from escore import process_manager, ForkStore
>>> fs = process_manager.service(ForkStore)
```

One can treat the datastore as any other dict:

```
>>> fs['a'] = 1
>>> fs['b'] = 2
>>> fs['0'] = 3
>>> a = fs['a']
```

Print()
Print a summary the shared fork objects.

__init__()
Initialize ForkStore instance.

clear()
Clear fork store dictionary

copy()
Perform a shallow copy of self.

Returns copy

get(*key: str*, *default: Any = None*) → object
Get value of key. If it does not exists return the default value.

Parameters

- **key** – The key to get.

- **default** – The default value of the key.

Returns The value of the key or None if it does not exist.

wait_until_unlocked()
Wait until unlocked

class escore.core.process_services.**ProcessService**
Bases: object

Base class for process services.

__init__()
Initialize service instance.

classmethod create()
Create an instance of this service.

Returns service instance

Return type *ProcessService*

finish()
Finish current processes.

This function can be implemented by a process-service implementation to finish running processes and clean up to prepare for a reset of the process manager. This would typically involve deleting large objects and closing files and database connections.

classmethod import_from_file(*file_path*)
Import service instance from a Pickle file.

Parameters **file_path** (*str*) – path of Pickle file

Returns imported service instance

Return type *ProcessService*

Raises RuntimeError, TypeError

logger
A logger that emits log messages to an observer.

The logger can be instantiated as a module or class attribute, e.g.

```
>>> logger = Logger()
>>> logger.info("I'm a module logger attribute.")
>>>
>>> class Point(object):
>>>     logger = Logger()
>>>
>>>     def __init__(self, x = 0.0, y = 0.0):
>>>         Point.logger.debug('Initializing {point} with x = {x}  y = {y}',␣
↪point=Point, x=x, y=y)
>>>         self._x = x
>>>         self._y = y
>>>
>>>     @property
>>>     def x(self):
>>>         self.logger.debug('Getting property x = {point._x}', point=self)
>>>         return self._x
>>>
>>>     @x.setter
```

(continues on next page)

(continued from previous page)

```
>>>     def x(self, x):
>>>         self.logger.debug('Setting property y = {point._x}', point=self)
>>>         self._x = x
>>>
>>>     @property
>>>     def y(self):
>>>        self.logger.debug('Getting property y = {point._y}', point=self)
>>>        return self._y
>>>
>>>     @y.setter
>>>     def y(self, y):
>>>         self.logger.debug('Setting property y = {point._y}', point=self)
>>>         self._y = y
>>>
>>> a_point = Point(1, 2)
>>>
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
>>> logger.log_level = LogLevel.DEBUG
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
```

The logger uses PEP-3101 (Advanced String Formatting) with named placeholders, see <https://www.python.org/dev/peps/pep-3101/> and <https://pyformat.info/> for more details and examples.

Furthermore, logging events are only formatted and evaluated for logging levels that are enabled. So, there's no need to check the logging level before logging. It's also efficient.

persist_in_file (*file_path*)
Persist service instance in Pickle file.

Parameters **file_path** (*str*) – path of Pickle file

class escore.core.process_services.**ProcessServiceMeta**
Bases: type

Meta class for process-services base class.

persist
Flag to indicate if service can be persisted.

escore.core.run_utils module

Project: Eskapade - A python-based package for data analysis.

Created: 2017/04/11

Description: Utilities for Eskapade run

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

Redistribution and use in source and binary forms, with or without modification, are permitted according to the terms listed in the file LICENSE.

escore.core.run_utils.**create_arg_parser**()
Create parser for user arguments.

An argparse parser is created and returned, ready to parse arguments specified by the user on the command line.

Returns argparse.ArgumentParser

Module contents

escore.core_ops package

Subpackages

escore.core_ops.links package

Submodules

escore.core_ops.links.apply module

Project: Eskapade - A python-based package for data analysis.

Class: DsApply

Created: 2018-06-30

Description: Simple link to execute functions, to which datastore has been passed.

Helps in the development of links.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.apply.**DsApply**(***kwargs*)

Bases: *escore.core.element.Link*

Simple link to execute functions to which datastore is passed.

__init__(***kwargs*)

Initialize an instance.

Parameters

- **name** (*str*) – name of link
- **apply** (*list*) – list of functions to execute at execute(), to which datastore is passed

execute()

Execute the link.

Returns status code of execution

Return type *StatusCode*

escore.core_ops.links.assert_in_ds module

Project: Eskapade - A python-based package for data analysis.

Class: AssertInDs

Created: 2016/11/08

Description: Algorithm that asserts that items exists in the datastore

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

Redistribution and use in source and binary forms, with or without modification, are permitted according to the terms listed in the file LICENSE.

class escore.core_ops.links.assert_in_ds.**AssertInDs**(***kwargs*)
Bases: *escore.core.element.Link*

Asserts that specified item(s) exists in the datastore.

__init__(***kwargs*)
Initialize link instance.

Store the configuration of link AssertInDs

Parameters

- **name** (*str*) – name of link
- **keySet** (*lst*) – list of keys to check

execute()
Execute the link.

escore.core_ops.links.break_link module

Project: Eskapade - A python-based package for data analysis.

Class: Break

Created: 2017/02/26

Description: Algorithm to send break signal to process manager and halt execution

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

Redistribution and use in source and binary forms, with or without modification, are permitted according to the terms listed in the file LICENSE.

class escore.core_ops.links.break_link.**Break**(***kwargs*)
Bases: *escore.core.element.Link*

Halt execution.

Link sends failure signal and halts execution of process manager. Break the execution of the processManager at a specific location by simply adding this link at any location in a chain.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **send_break** (*bool*) – if true, send StatusCode.BreakChain (skip execution of rest of chain). Default is false, then sends StatusCode.Failure (exit program).

execute()
Execute the link.

escore.core_ops.links.ds_object_deleter module

Project: Eskapade - A python-based package for data analysis.

Class: DsObjectDeleter

Created: 2016/11/08

Description: Algorithm to delete objects from the datastore.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.ds_object_deleter.**DsObjectDeleter**(***kwargs*)
Bases: *escore.core.element.Link*

Delete objects from the datastore.

Delete objects from the DataStore by the key they are under, or keeps only the data by the specified keys.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **deletion_keys** (*list*) – keys to clear. Overwrites clear_all to false.
- **deletion_classes** (*list*) – delete object(s) by class type.
- **keep_only** (*lsst*) – keys to keep. Overwrites clear_all to false.
- **clear_all** (*bool*) – clear all key-value pairs in the datastore. Default is true.

execute()
Execute the link.

initialize()
Initialize the link.

escore.core_ops.links.ds_to_ds module

Project: Eskapade - A python-based package for data analysis.

Class: DsToDs

Created: 2016/11/08

Description: Algorithm to move, copy, or remove an object in the datastore.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.ds_to_ds.**DsToDs**(***kwargs*)
Bases: *escore.core.element.Link*

Link to move, copy, or remove an object in the datastore.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from data store

- **store_key** (*str*) – key of data to store in data store
- **move** (*bool*) – move read_key item to store_key. Default is true.
- **copy** (*bool*) – if True the read_key key, value pair will not be deleted. Default is false.
- **remove** (*bool*) – if True the item corresponding to read_key key will be deleted. Default is false.

execute()
Execute the link.

initialize()
Initialize the link.

escore.core_ops.links.event_looper module

Project: Eskapade - A python-based package for data analysis.

Class: EventLooper

Created: 2016/11/08

Description: EventLooper algorithm processes input lines and reprints them, e.g. to use with map/reduce

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.event_looper.**EventLooper**(***kwargs*)
Bases: *escore.core.element.Link*

Event looper algorithm processes input lines and reprints or stores them.

Input lines are taken from sys.stdin, processed, and printed on screen.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **filename** (*str*) – file name where the strings are located (txt or similar). Default is None. (optional)
- **store_key** (*str*) – key to collect in datastore. If set lines are collected. (optional)
- **line_processor_set** (*list*) – list of functions to apply to input lines. (optional)
- **sort** (*bool*) – if true, sort lines before storage (optional)
- **unique** (*bool*) – if true, keep only unique lines before storage (optional),
- **skip_line_beginning_with** (*list*) – skip line if it starts with any of the list. input is list of strings. Default is ['#'] (optional)

execute()
Process all incoming lines.

No output is printed except for lines that are passed on, such that the output lines can be picked up again by another parser.

finalize()
Close open file if present.

initialize()
Perform basic checks of configured attributes.

escore.core_ops.links.hello_world module

Project: Eskapade - A python-based package for data analysis.

Class: HelloWorld

Created: 2017/01/31

Description: Algorithm to do print Hello {}!

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.hello_world.**HelloWorld**(***kwargs*)
Bases: *escore.core.element.Link*

Defines the content of link HelloWorld.

__init__(***kwargs*)
Store the configuration of link HelloWorld.

Parameters

- **name** (*str*) – name assigned to the link
- **hello** (*str*) – name to print in Hello World! Defaults to 'World'
- **repeat** (*int*) – repeat print statement N times. Default is 1

execute()
Execute the link.

escore.core_ops.links.import_data_store module

Project: Eskapade - A python-based package for data analysis.

Class: ImportDataStore

Created: 2018-03-17

Description: Algorithm to import datastore from external pickle file

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.import_data_store.**ImportDataStore**(***kwargs*)
Bases: *escore.core.element.Link*

Link to import datastore from external pickle file.

Import can happen at initialize() or execute(). Default is initialize()

__init__(***kwargs*)
Initialize an instance of the datastore importer link.

Parameters

- **name** (*str*) – name of link
- **path** (*str*) – path of the datastore pickle file to import
- **update** (*bool*) – if true update the existing datastore, don't replace it. Default is false.
- **import_at_initialize** (*bool*) – if false, perform datastore import at execute. Default is true, at initialize.

execute()
Execute the link.

Returns status code of execution

Return type *StatusCode*

import_and_update_datastore()
Import and update the datastore

initialize()
Initialize the link.

Returns status code of execution

Return type *StatusCode*

escore.core_ops.links.ipython_embed module

Project: Eskapade - A python-based package for data analysis.

Class: IPythonEmbed

Created: 2017/02/26

Description: Link that starts up a python console during execution for debugging.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.ipython_embed.**IPythonEmbed**(***kwargs*)
Bases: *escore.core.element.Link*

Link to start up a python console.

Start up a python console by simply adding this link at any location in a chain. Note: not an ipython console, but regular python console.

__init__(***kwargs*)
Initialize link instance.

Parameters **name** (*str*) – name of link

execute()
Execute the link.

escore.core_ops.links.line_printer module

Project: Eskapade - A python-based package for data analysis.

Class: LinePrinter

Created: 2017/02/21

Description: Simple algorithm to pick up lines and reprint them.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.line_printer.**LinePrinter**(***kwargs*)

Bases: *escore.core.element.Link*

LinePrinter picks up lines from the datastore and prints them.

__init__(***kwargs*)

Set up the configuration of link LinePrinter.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store

execute()

Execute the link.

No output is printed except for lines that are passed on, such that the output lines can be picked up again by another parser.

initialize()

Initialize the link.

escore.core_ops.links.print_ds module

Project: Eskapade - A python-based package for data analysis.

Class: PrintDs

Created: 2016/11/08

Description: Algorithm to print the content of the datastore.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.print_ds.**PrintDs**(***kwargs*)

Bases: *escore.core.element.Link*

Print the content of the datastore.

__init__(***kwargs*)

Initialize link instance.

Parameters

- **name** (*str*) – name of link

- **keys** (*list*) – keys of items to print explicitly.

execute()
Execute the link.

Print overview of the datastore in current state.

escore.core_ops.links.repeat_chain module

Project: Eskapade - A python-based package for data analysis.

Class: RepeatChain

Created: 2016/11/08

Description: Algorithm that sends "repeat this chain" signal to processManager, until ready.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.repeat_chain.**RepeatChain**(***kwargs*)
Bases: *escore.core.element.Link*

Algorithm that sends signal to processManager to repeat the current chain.

__init__(***kwargs*)
Link that sends signal to processManager to repeat the current chain.

Sents a RepeatChain deenums.StatusCode signal.

Parameters

- **name** (*str*) – name of link
- **listen_to** (*list*) – repeat this chain if given key is present in ConfigObject and set to true. E.g. this key is set by readtods link when looping over files.
- **maxcount** (*int*) – repeat this chain until max count has been reacher. Default is -1 (off).

execute()
Execute the link.

initialize()
Initialize the link.

escore.core_ops.links.skip_chain_if_empty module

Project: Eskapade - A python-based package for data analysis.

Class: SkipChainIfEmpty

Created: 2016/11/08

Description: Algorithm to skip to the next Chain if input dataset is empty

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.skip_chain_if_empty.**SkipChainIfEmpty**(***kwargs*)
Bases: *escore.core.element.Link*

Sends a SkipChain enums.StatusCode signal when an appointed dataset is empty.

This signal causes that the Processs Manager to step immediately to the next Chain. Input collections can be either mongo collections or dataframes in the datastore.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **collection_set** (*list*) – datastore keys holding the datasets to be checked. If any of these is empty, the chain is skipped.
- **skip_missing** (*bool*) – skip the chain if the dataframe is not present in the datastore. Default is True.
- **skip_zero_len** (*bool*) – skip the chain if the object is found in the datastore but has zero length. Default is True.
- **check_at_initialize** (*bool*) – perform dataset empty is check at initialize. Default is true.

check_collection_set()
Check existence of collection in the datastore, and check if they are empty.

All collections need to be both present and not empty for the chain to be continued.

execute()
Execute the link.

Skip to the next Chain if any of the input dataset is empty.

initialize()
Initialize the link.

escore.core_ops.links.to_ds_dict module

Project: Eskapade - A python-based package for data analysis.

Class: ToDsDict

Created: 2016/11/08

Description: Algorithm to store one object in the DataStore dict during run time.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class escore.core_ops.links.to_ds_dict.**ToDsDict**(***kwargs*)
Bases: *escore.core.element.Link*

Stores one object in the DataStore dict during run time.

__init__(***kwargs*)
Link to store one external object in the DataStore dict during run time.

Parameters

- **name** (*str*) – name of link
- **store_key** (*str*) – key of object to store in data store
- **obj** – object to store
- **force** (*bool*) – overwrite if already present in datastore. default is false. (optional)
- **at_initialize** (*bool*) – store at initialize of link. Default is false.
- **at_execute** (*bool*) – store at execute of link. Default is true.
- **copydict** (*bool*) – if true and obj is a dict, copy all key value pairs into datastore. Default is false.

do_storage(*ds*)
Perform storage in datastore.

Function makes a distinction been dicts and any other object.

execute()
Execute the link.

initialize()
Initialize the link.

Module contents

class escore.core_ops.links.**AssertInDs**(**kwargs*)
Bases: *escore.core.element.Link*

Asserts that specified item(s) exists in the datastore.

__init__(**kwargs*)
Initialize link instance.

Store the configuration of link AssertInDs

Parameters

- **name** (*str*) – name of link
- **keySet** (*lst*) – list of keys to check

execute()
Execute the link.

class escore.core_ops.links.**Break**(**kwargs*)
Bases: *escore.core.element.Link*

Halt execution.

Link sends failure signal and halts execution of process manager. Break the execution of the processManager at a specific location by simply adding this link at any location in a chain.

__init__(**kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **send_break** (*bool*) – if true, send StatusCode.BreakChain (skip execution of rest of chain). Default is false, then sends StatusCode.Failure (exit program).

execute()
Execute the link.

class escore.core_ops.links.**DsObjectDeleter**(**kwargs)
Bases: *escore.core.element.Link*

Delete objects from the datastore.

Delete objects from the DataStore by the key they are under, or keeps only the data by the specified keys.

__init__(**kwargs)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **deletion_keys** (*list*) – keys to clear. Overwrites clear_all to false.
- **deletion_classes** (*list*) – delete object(s) by class type.
- **keep_only** (*lsst*) – keys to keep. Overwrites clear_all to false.
- **clear_all** (*bool*) – clear all key-value pairs in the datastore. Default is true.

execute()
Execute the link.

initialize()
Initialize the link.

class escore.core_ops.links.**DsToDs**(**kwargs)
Bases: *escore.core.element.Link*

Link to move, copy, or remove an object in the datastore.

__init__(**kwargs)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from data store
- **store_key** (*str*) – key of data to store in data store
- **move** (*bool*) – move read_key item to store_key. Default is true.
- **copy** (*bool*) – if True the read_key key, value pair will not be deleted. Default is false.
- **remove** (*bool*) – if True the item corresponding to read_key key will be deleted. Default is false.

execute()
Execute the link.

initialize()
Initialize the link.

class escore.core_ops.links.**EventLooper**(**kwargs)
Bases: *escore.core.element.Link*

Event looper algorithm processes input lines and reprints or stores them.

Input lines are taken from sys.stdin, processed, and printed on screen.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **filename** (*str*) – file name where the strings are located (txt or similar). Default is None. (optional)
- **store_key** (*str*) – key to collect in datastore. If set lines are collected. (optional)
- **line_processor_set** (*list*) – list of functions to apply to input lines. (optional)
- **sort** (*bool*) – if true, sort lines before storage (optional)
- **unique** (*bool*) – if true, keep only unique lines before storage (optional),
- **skip_line_beginning_with** (*list*) – skip line if it starts with any of the list. input is list of strings. Default is ['#'] (optional)

execute()
Process all incoming lines.

No output is printed except for lines that are passed on, such that the output lines can be picked up again by another parser.

finalize()
Close open file if present.

initialize()
Perform basic checks of configured attributes.

class escore.core_ops.links.**HelloWorld**(***kwargs*)
Bases: *escore.core.element.Link*

Defines the content of link HelloWorld.

__init__(***kwargs*)
Store the configuration of link HelloWorld.

Parameters

- **name** (*str*) – name assigned to the link
- **hello** (*str*) – name to print in Hello World! Defaults to 'World'
- **repeat** (*int*) – repeat print statement N times. Default is 1

execute()
Execute the link.

class escore.core_ops.links.**IPythonEmbed**(***kwargs*)
Bases: *escore.core.element.Link*

Link to start up a python console.

Start up a python console by simply adding this link at any location in a chain. Note: not an ipython console, but regular python console.

__init__(***kwargs*)
Initialize link instance.

Parameters **name** (*str*) – name of link

execute()
Execute the link.

class escore.core_ops.links.**LinePrinter**(***kwargs*)
Bases: *escore.core.element.Link*

LinePrinter picks up lines from the datastore and prints them.

__init__(***kwargs*)
Set up the configuration of link LinePrinter.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store

execute()
Execute the link.

No output is printed except for lines that are passed on, such that the output lines can be picked up again by another parser.

initialize()
Initialize the link.

class escore.core_ops.links.**PrintDs**(***kwargs*)
Bases: *escore.core.element.Link*

Print the content of the datastore.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **keys** (*list*) – keys of items to print explicitly.

execute()
Execute the link.

Print overview of the datastore in current state.

class escore.core_ops.links.**RepeatChain**(***kwargs*)
Bases: *escore.core.element.Link*

Algorithm that sends signal to processManager to repeat the current chain.

__init__(***kwargs*)
Link that sends signal to processManager to repeat the current chain.

Sents a RepeatChain deenums.StatusCode signal.

Parameters

- **name** (*str*) – name of link
- **listen_to** (*list*) – repeat this chain if given key is present in ConfigObject and set to true. E.g. this key is set by readtods link when looping over files.
- **maxcount** (*int*) – repeat this chain until max count has been reacher. Default is -1 (off).

execute()
Execute the link.

initialize()
Initialize the link.

class escore.core_ops.links.**SkipChainIfEmpty**(***kwargs*)
Bases: *escore.core.element.Link*

Sends a SkipChain enums.StatusCode signal when an appointed dataset is empty.

This signal causes that the Processs Manager to step immediately to the next Chain. Input collections can be either mongo collections or dataframes in the datastore.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **collection_set** (*list*) – datastore keys holding the datasets to be checked. If any of these is empty, the chain is skipped.
- **skip_missing** (*bool*) – skip the chain if the dataframe is not present in the datastore. Default is True.
- **skip_zero_len** (*bool*) – skip the chain if the object is found in the datastore but has zero length. Default is True.
- **check_at_initialize** (*bool*) – perform dataset empty is check at initialize. Default is true.

check_collection_set()
Check existence of collection in the datastore, and check if they are empty.

All collections need to be both present and not empty for the chain to be continued.

execute()
Execute the link.

Skip to the next Chain if any of the input dataset is empty.

initialize()
Initialize the link.

class escore.core_ops.links.**ToDsDict**(***kwargs*)
Bases: *escore.core.element.Link*

Stores one object in the DataStore dict during run time.

__init__(***kwargs*)
Link to store one external object in the DataStore dict during run time.

Parameters

- **name** (*str*) – name of link
- **store_key** (*str*) – key of object to store in data store
- **obj** – object to store
- **force** (*bool*) – overwrite if already present in datastore. default is false. (optional)
- **at_initialize** (*bool*) – store at initialize of link. Default is false.
- **at_execute** (*bool*) – store at execute of link. Default is true.
- **copydict** (*bool*) – if true and obj is a dict, copy all key value pairs into datastore. Default is false.

do_storage(*ds*)
Perform storage in datastore.

Function makes a distinction been dicts and any other object.

execute()
Execute the link.

initialize()
Initialize the link.

class escore.core_ops.links.**DsApply**(***kwargs*)
Bases: *escore.core.element.Link*

Simple link to execute functions to which datastore is passed.

__init__(***kwargs*)
Initialize an instance.

Parameters

- **name** (*str*) – name of link
- **apply** (*list*) – list of functions to execute at execute(), to which datastore is passed

execute()
Execute the link.

Returns status code of execution

Return type *StatusCode*

class escore.core_ops.links.**ImportDataStore**(***kwargs*)
Bases: *escore.core.element.Link*

Link to import datastore from external pickle file.

Import can happen at initialize() or execute(). Default is initialize()

__init__(***kwargs*)
Initialize an instance of the datastore importer link.

Parameters

- **name** (*str*) – name of link
- **path** (*str*) – path of the datastore pickle file to import
- **update** (*bool*) – if true update the existing datastore, don't replace it. Default is false.
- **import_at_initialize** (*bool*) – if false, perform datastore import at execute. Default is true, at initialize.

execute()
Execute the link.

Returns status code of execution

Return type *StatusCode*

import_and_update_datastore()
Import and update the datastore

initialize()
Initialize the link.

Returns status code of execution

Return type *StatusCode*

class escore.core_ops.links.**ForkExample**(**kwargs)
Bases: *escore.core.element.Link*

Defines the content of link.

__init__(**kwargs)
Initialize an instance.

Parameters

- **name** (*str*) – name of link
- **store_key** (*str*) – key of object to store in data store

execute()
Execute the link.

Returns status code of execution

Return type *StatusCode*

finalize()
Finalized the link.

Returns status code of finalize

Return type *StatusCode*

initialize()
Initialize the link.

Returns status code of initialization

Return type *StatusCode*

class escore.core_ops.links.**ForkDataCollector**(**kwargs)
Bases: *escore.core.element.Link*

Defines the content of link.

__init__(**kwargs)
Initialize an instance.

Parameters

- **name** (*str*) – name of link
- **keys** (*list*) – functions to apply (list of dicts) - 'key_ds' (string): input key in datastore - 'key_fs' (string, optional): output key in forkstore - 'func': function to apply, optional - 'append': if key_ds points to a list, append each item to list in forkstore. Default is True. - 'args' (tuple, optional): args for 'func' - 'kwargs' (dict, optional): kwargs for 'func'

execute()
Execute the link.

Returns status code of execution

Return type *StatusCode*

finalize()
Finalize the link.

Returns status code of finalization

Return type *StatusCode*

initialize()
Initialize the link.

Returns status code of initialization

Return type *StatusCode*

class escore.core_ops.links.**ApplyFunc**(**args*, ***kwargs*)
Bases: *escore.core.element.Link*

Algorithm that applies a function to an object in the datastore.

__init__(**args*, ***kwargs*)
Initialize an instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **default** – The default value of the key in case not found.
- **assert_type** – if set, check object for given type or tuple of types. If fails, raise TypeError.
- **assert_len** (*bool*) – if true, check that object has length greater than 0. If fails, raise TypeError or AssertionError.
- **assert_in** (*bool*) – assert that key is known, default is true.
- **func** – function to execute
- **args** – all args are passed pass to function as args.
- **kwargs** – all other key word arguments are passed on to the function as kwargs.
- **store_key** (*str*) – key of output data to store in data store

execute()
Execute the link.

Returns status code of execution

Return type *StatusCode*

initialize()
Initialize the link.

Returns status code of initialization

Return type *StatusCode*

class escore.core_ops.links.**SkipChainIfPresent**(***kwargs*)
Bases: *escore.core.element.Link*

Sends a SkipChain enums.StatusCode signal when a requested dataset is present.

This signal causes that the Processs Manager to step immediately to the next Chain.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link

- **collection_set** (*list*) – datastore keys holding the datasets to be checked. If all of these are present, the chain is skipped.
- **check_at_initialize** (*bool*) – if false, perform dataset present check at execute. Default is true.
- **assert_len** (*bool*) – assert that each collection has a length greatet than zero. Default is true.
- **assert_type** (*tuple*) – types to assert for each object. Default is ().

check_collection_set()
Check existence of collections in the datastore, and check that they are all present.

Collections need to be both present and not empty.

- For pandas dataframes the additional option 'skip_chain_when_key_not_in_ds' exists. Meaning, skip the chain as well if the dataframe is not present in the datastore.

execute()
Execute the link.

Skip to the next Chain if all of the input collections are present.

initialize()
Initialize the link.

Module contents

escore.logger package

Module contents

Submodules

escore.entry_points module

Project: Eskapade - A python-based package for data analysis.

Created: 2017-08-08

Description: Collection of eskapade entry points

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

escore.entry_points.**eskapade_bootstrap**()
Generate Eskapade project structure.

escore.entry_points.**eskapade_generate_link**()
Generate Eskapade link.

By default does not create init file.

escore.entry_points.**eskapade_generate_macro**()
Generate Eskapade macro.

escore.entry_points.**eskapade_generate_notebook**()
Generate Eskapade notebook.

escore.entry_points.**eskapade_run**()
Run Eskapade.

Top-level entry point for an Eskapade run started from the command line. Arguments specified by the user are parsed and converted to settings in the configuration object. Optionally, an interactive Python session is started when the run is finished.

escore.entry_points.**eskapade_trial**()
Run Eskapade tests.

We will keep this here until we've completed switch to pytest or nose and tox. We could also keep it, but I don't like the fact that packages etc. are hard coded. Gotta come up with a better solution.

escore.exceptions module

Project: Eskapade - A python-based package for data analysis.

Created: 2017/03/31

Description: Eskapade exceptions

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

exception escore.exceptions.**AmbiguousFileType**(*path*)
Bases: Exception

Exception raised if file type cannot be inferred

__init__(*path*)
Raise the exception

Parameters **path** (*str*) – the path to file

exception escore.exceptions.**MissingPackageError**(*message=''*, *required_by=''*)
Bases: Exception

Exception raised if third-party package is missing.

__init__(*message=''*, *required_by=''*)
Set missing-package arguments.

Parameters

- **message** (*str*) – message to show when raised
- **required_by** (*str*) – info on component that requires the package

exception escore.exceptions.**UnhandledFileType**(*path*, *f_ext*, *file_type*)
Bases: Exception

Exception raised if file type is not handled

__init__(*path*, *f_ext*, *file_type*)
Raise the exception

Parameters

- **path** (*str*) – the path to file
- **f_ext** (*str*) – file extension as determined by splitting the path string
- **file_type** (*str*) – user set file type. Options are {'npy', 'npz'}

escore.resources module

Project: Eskapade - A python-based package for data analysis.

Created: 2017/08/23

Description: Collection of helper functions to get fixtures, i.e. test data, ROOT/RooFit libs, and tutorials. These are mostly used by the (integration) tests.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

escore.resources.**template**(*name: str*) → str
Return the full path filename of a tutorial.

Parameters **name** (*str*) – The name of the template.

Returns The full path filename of the tutorial.

Return type str

Raises **FileNotFoundError** – If the template cannot be found.

escore.resources.**tutorial**(*name: str*) → str
Return the full path filename of a tutorial.

Parameters **name** (*str*) – The name of the tutorial.

Returns The full path filename of the tutorial.

Return type str

Raises **FileNotFoundError** – If the tutorial cannot be found.

escore.utils module

Project: Eskapade - A python-based package for data analysis.

Created: 2016/11/08

Description: Utility functions to collect Eskapade python modules e.g. functions to get correct Eskapade file paths and env variables

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

escore.utils.**check_interactive_backend**()
Check whether an interactive backend is required

escore.utils.**collect_python_modules**()
Collect Eskapade Python modules.

escore.utils.**get_env_var**(*key*)
Retrieve Eskapade-specific environment variables.

Parameters **key** (*str*) – Eskapade-specific key to variable

Returns environment variable value

Return type str

escore.utils.**in_ipynb**()
Detect whether an Jupyter/Ipython-kernel is being run

Raises NameError

escore.utils.**in_tty**()
Detect whether running in a terminal

escore.utils.**set_matplotlib_backend**(*backend=None*, *batch=None*, *silent=True*)
Set Matplotlib backend.

Parameters

- **backend** (*str*) – backend to set
- **batch** (*bool*) – require backend to be non-interactive
- **silent** (*bool*) – do not raise exception if backend cannot be set

Raises RuntimeError

escore.version module

THIS FILE IS AUTO-GENERATED BY ESKAPADE SETUP.PY.

Module contents

5.5 Indices and tables

- genindex
- modindex

Python Module Index

e

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

P

R

S

T

U

W