Ebonite

Release 0.7.0

Oct 19, 2020

Contents

CHAPTER 1

Installation

At the command line:

```
pip install ebonite
```

CHAPTER 2

Readme

Ebonite is a framework for machine learning lifecycle. For now, it's main focus is on model deployment, but in future it will cover more areas.

Ebonite consists of three main modules and some extensions modules.

2.1 Ebonite Core

This module is responsible for model analysis and model persisting. If you use vanilla ebonite, this is mainly what you are working with.

Main model analysis API abstractions are

- *DatasetHook* - hook for dataset analysis
- *DatasetType* - dataset descriptor
- *ModelHook* - hook for model understanding
- *ModelWrapper* - model wrapper for different ml libraries
- *ModelIO* - model input / output for different ml model serializers

Main model persisting abstractions are

- *MetadataRepository* - Repository to store model metadata (like sql database)
- *ArtifactRepository* - Repository to store model artifacts (like s3 or nexus)
- *Ebonite* - Main user-facing ebonite client class

Also these helper functions available:

- *create_model()* - creates *Model* instance from model object and sample data

2.2 Ebonite Build

Build module is responsible for building and running images. For now, ebonite supports only docker images, but it is possible to implement some classes to add support for other types of things that can be built.

Here are build abstractions:

- *ProviderBase* - provides files for the image. Builtin implementations: *MLModelProvider* which generates files from *Model* object. Also there is *MLModelMultiProvider* if you want multiple models in one image.
- *BuilderBase* - builds images from files generated by *ProviderBase*. Builtin implementation: `DockerBuilder` builds docker images.
- *RunnerBase* - runs images. Builtin implementation: `DockerRunner` - runs docker images locally or on remote server.

Also, these helper functions are available:

- `run_docker_img()` - runs docker image.

2.3 Ebonite Runtime

Runtime module is responsible for code that runs inside containers.

Here are runtime abstractions:

- *Interface* - an object with some exposed methods. Builtin implementation: `ModelInterface` which is created dynamically with *model_interface()*.
- *InterfaceLoader* - loads *Interface* instance. Builtin implementation: *ModelLoader*
- *Server* - gets an instance of *Interface* and exposes it's methods via some protocol. Builtin implementations: *FlaskServer* and `AIOHTTPServer` - all expose methods as http POST endpoints.

Also, these helper functions are available:

- *run_model_server()* - create `ModelInterface` from model and runs debug *Server*.

2.4 Ebonite Extensions

Ebonite can be extended in any way, just write the code. But there are already some builtin extensions that provide integrations with different python libraries. Those extensions loads automatically.

Note: Some of them loads if you have corresponding libraries installed, some of them loads only if you directly import corresponding library.

Extensions are loaded via *ExtensionLoader*.

Here are builtin extensions:

- `aiohttp` - `AIOHTTPServer` server
- `catboost` - support for CatBoost library
- *flask* - *FlaskServer* server

- `imageio` - support for working with image payload
- `lightgbm` - support for LightGBM library
- `numpy` - support for numpy data types
- `pandas` - support for pandas data types
- `s3` - s3 *`ArtifactRepository`* implementation
- `sklearn` - support for scikit-learn models
- `sqlalchemy` - sql *`MetadataRepository`* implementation
- `tensorflow` - support for tensorflow 1.x models
- `tensorflow_v2` - support for tensorflow 2.x models
- `torch` - support for torch models
- `xgboost` - support for xgboost models

Ebonite is customizable, and every module has it's own abstractions one can implement.

CHAPTER 3

Usage

3.1 Quickstart

Ebonite can be used to reproduce arbitrary machine learning model in different environments.

Note: Don't forget to install requirements for this example: `pip install pandas scikit-learn flask flasgger`

For instance, you can train sklearn model (code):

```
reg = LogisticRegression()
data = pd.DataFrame([[1, 0], [0, 1]], columns=['a', 'b'])
reg.fit(data, [1, 0])
```

To use ebonite you need to create Ebonite client (code):

```
ebnt = ebonite.Ebonite.local(clear=True)
```

Now you need to create task to push your model into (code):

```
ebnt.create_model(reg, data, model_name='mymodel',
```

Great, now you can reproduce this model in different environment using this code (code):

```
model = ebnt.get_model(project='my_project', task='regression_is_my_profession',␣
↪model_name='mymodel')
```

And start a server that processes inference request like this (code):

```
from ebonite.runtime import run_model_server
run_model_server(model)
```

Or create and start a docker container like this (code):

```
uild docker image from model and run it
.build_and_run_instance(model, "sklearn_model_service",
                        runner_kwargs={'detach': False},
```

Full code can be found in examples/sklearn_model.

3.1.1 Other examples

More examples available here:

- Jupyter Notebook guide
- TensorFlow 2.0
- etc

3.2 Persisting Models

After you got yourself a *Model* instance, you can persist it to a repository. For that, you need a *Task* instance to add model to. *Task* is a container for models trained for the same problem. For example, if you did some experiments, you'll push each experiment as a *Model* to the same *Task*.

Each *Task* belongs to a *Project*, which is just container for tasks.

To create and persist projects, tasks and models, you need ebonite client, which is and instance of *Ebonite*. Ebonite client is a composition of two repository implementations: *MetadataRepository* and *ArtifactRepository*.

MetadataRepository is where all the metadata goes, you look at it as a SQL database (we actually have an sql implementation).

ArtifactRepository is where all model binaries go. It can be any file storage like s3, ftp and so on.

You can manually create client with `Ebonite(metadata_repository, artifact_repository)`, or use one of the factory methods: *local()* for local client (metadata will just a json file, and artifacts will be just plain files in local file system), *inmemory()* for in-memory repositories.

Also there is a *custom_client()* to setup your own repositories.

You can use `MetadataRepository.type` value as for metadata argument.

Available implementations:

- local - *LocalMetadataRepository*
- sqlalchemy - `SQLAlchemyMetaRepository`

You can use `ArtifactRepository.type` value as for artifact argument.

Available implementations:

- local - *LocalArtifactRepository*
- inmemory - *InMemoryArtifactRepository*
- s3 - `S3ArtifactRepository`

Let's create local ebonite client (code):

```
ebnt = ebonite.Ebonite.local(clear=True)
```

Now, create project and task for our model (code):

```
#  then push it to repositories. this will create .ebonite dir with metadata.json and artifacts dir
```

And push model into it (code):

```

```

Now, if you take a look at `.ebonite` directory, you'll find a `metadata.json` file with your project, task and model.

Congratulations, you persisted your model. This process is absolutely the same if you choose other repository implementations. Take a look at examples/remote_example for an example with remote repositories.

3.3 Building and running docker images

The easiest way to build docker image from *Model* is to use `build_image()`. If you need more customizable solution and/or don't need image metadata persistence you can use :`DockerBuilder` class manually. However if you need to customize even more, you can manually implement *ProviderBase* and *BuilderBase* classes.

3.4 Adding custom analyzers

To add support for new ML library or new types of data, you need to implement a hook for analyzer and the type it produces.

3.4.1 Model support

For models, you need to implement *BindingModelHook*, *ModelWrapper* and *ModelIO*. *BindingModelHook* should check an object if it is the object that you want to add support for (for example, check it's base module to be the library you providing support for). Result of `_wrapper_factory()` must be an instance of *ModelWrapper* implementation you provided. We recommend to mixin *TypeHookMixin* to simplify hook implementation. Even if it's not possible please provide *valid_types* value anyway. In *ModelWrapper* you must implement `_exposed_methods_mapping()` method and constructor which creates corresponding *ModelIO* subclass instance. In *ModelIO* you must implement `dump()` and `load()` methods.

3.4.2 Data type support

For data types, you need to implement *DatasetHook* and *DatasetType*. *DatasetHook* should check an object if it is the object that you want to add support for (for example, check it's base module to be the library you providing support for). Result of *process()* must be an instance of *DatasetType* implementation you provided. In *DatasetType* you must implement methods `serialize()`, `deserialize()` and `get_spec()`.

3.4.3 Tips

If you want better understating of what is going on, check some of the extensions, for example `lightgbm` provides these implementations for both model and data type.

Also, check out *analyzer* for some convenient mixins.

3.5 Adding custom repositories

To start using ebonite, put a bucket on your head.

3.6 Adding custom servers

To start using ebonite, put a bucket on your head.

CHAPTER 4

ebonite package

ebonite.**load_extensions**(**exts*)

Load extensions

Parameters **exts** – list of extension main modules

class ebonite.**Ebonite**(*meta_repo: ebonite.repository.metadata.base.MetadataRepository*, *artifact_repo: ebonite.repository.artifact.base.ArtifactRepository*, *dataset_repo: ebonite.repository.dataset.base.DatasetRepository = None*)

Bases: object

Main entry point for ebonite

This is the client for Ebonite API. It can save, load and build Models, Tasks and Projects. Ebonite instance can be obtained from factory methods like *local()* for local client, *inmemory()* for inmemory client.

You can save client config with *save_client_config()* and later restore it with *from_config_file()*

Parameters

- **meta_repo** – *MetadataRepository* instance to save metadata
- **artifact_repo** – *ArtifactRepository* instance to save artifacts

default_server = None

default_env = None

push_model(*model: ebonite.core.objects.core.Model*, *task: ebonite.core.objects.core.Task = None*) → ebonite.core.objects.core.Model

Pushes *Model* instance into metadata and artifact repositories

Parameters

- **model** – *Model* instance
- **task** – *Task* instance to save model to. Optional if model already has

task :return: same saved *Model* instance

create_model(*model_object*, *model_input*, *model_name: str = None*, ***, *project_name: str = 'default_project'*, *task_name: str = 'default_task'*, ***kwargs*)
This function creates ebonite model. Creates model, task and project (if needed) and pushes it to repo

Parameters

- **model_object** – object containing model.
- **model_input** – model input.
- **model_name** – model name to create.
- **project_name** – project name.
- **task_name** – task name.
- **kwargs** – other arguments for model

Returns *Model* instance representing

get_model(*model_name: str, task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None, load_artifacts: bool = True*) → ebonite.core.objects.core.Model
Load model from repository

Parameters

- **model_name** – model name to load
- **task** – *Task* instance or task name to load model from
- **project** – *Project* instance or project name to load task from
- **load_artifacts** – if True, load model artifact into wrapper

Returns *Model* instance

create_image(*obj*, *name: str = None*, *task: ebonite.core.objects.core.Task = None*, *server: ebonite.runtime.server.base.Server = None*, *environment: ebonite.core.objects.core.RuntimeEnvironment = None*, *debug=False*, *skip_build=False*, *builder_args: Dict[str, object] = None*, ***kwargs*) → ebonite.core.objects.core.Image
Builds image of model service and stores it to repository

Parameters

- **obj** – model/list of models/pipeline or any object that has existing Hook for it to wrap into service
- **name** – name of image to build
- **task** – task to put image into
- **server** – server to build image with
- **environment** – env to build for
- **debug** – flag to build debug image
- **skip_build** – wheter to skip actual image build
- **builder_args** – kwargs for image.build
- **kwargs** – additional kwargs for builder

Returns *Image* instance representing built image

create_instance(*image: ebonite.core.objects.core.Image*, *name: str = None*, *environment: ebonite.core.objects.core.RuntimeEnvironment = None*, *run=False*, *runner_kwargs: Dict[str, object] = None*, ***instance_kwargs*) → ebonite.core.objects.core.RuntimeInstance

Runs model service instance and stores it to repository

Parameters

- **image** – image to run instance from
- **name** – name of instance to run
- **environment** – environment to run instance in, if no given *localhost* is used
- **run** – whether to automatically run instance after creation
- **runner_kwargs** – additional parameters for runner
- **instance_kwargs** – additional parameters for instance

Returns *RuntimeInstance* instance representing run instance

build_and_run_instance(*obj*, *name: str = None*, *task: ebonite.core.objects.core.Task = None*, *environment: ebonite.core.objects.core.RuntimeEnvironment = None*, *builder_kwargs: Dict[str, object] = None*, *runner_kwargs: Dict[str, object] = None*, *instance_kwargs: Dict[str, object] = None*) → ebonite.core.objects.core.RuntimeInstance

Builds image of model service, immediately runs service and stores both image and instance to repository

Parameters

- **obj** – buildable object to wrap into service
- **name** – name of image and instance to be built and run respectively
- **task** – task to put image into
- **environment** – environment to run instance in, if no given *localhost* is used
- **builder_kwargs** – additional kwargs for builder
- **runner_kwargs** – additional parameters for runner. Full list can be seen in https://docker-py.readthedocs.io/en/stable/containers.html
- **instance_kwargs** – additional parameters for instance

Returns *RuntimeInstance* instance representing run instance

classmethod local(*path=None*, *clear=False*) → ebonite.client.base.Ebonite

Get an instance of *Ebonite* that stores metadata and artifacts on local filesystem

Parameters

- **path** – path to storage dir. If None, *.ebonite* dir is used
- **clear** – if True, erase previous data from storage

classmethod inmemory() → ebonite.client.base.Ebonite

Get an instance of *Ebonite* with inmemory repositories

classmethod custom_client(*metadata: Union[str, ebonite.repository.metadata.base.MetadataRepository], artifact: Union[str, ebonite.repository.artifact.base.ArtifactRepository], meta_kwargs: dict = None, artifact_kwargs: dict = None*) → ebonite.client.base.Ebonite

Create custom Ebonite client from metadata and artifact repositories.

Parameters

- **metadata** – *MetadataRepository* instance or pyjackson subtype type name
- **artifact** – *ArtifactRepository* instance or pyjackson subtype type name
- **meta_kwargs** – kwargs for metadata repo __init__ if subtype type name was provided
- **artifact_kwargs** – kwargs for artifact repo __init__ if subtype type name was provided

Returns *Ebonite* instance

classmethod from_config_file(*filepath*) → ebonite.client.base.Ebonite
Read and create Ebonite instance from config file

Parameters **filepath** – path to read config from

Returns *Ebonite* instance

save_client_config(*filepath*)
Save current client config to a file

Parameters **filepath** – path to file

get_default_server()

Returns Default server implementation for this client

get_default_environment()
Creates (if needed) and returns default runtime environment

Returns saved instance of *RuntimeEnvironment*

push_environment(*environment: ebonite.core.objects.core.RuntimeEnvironment*) → ebonite.core.objects.core.RuntimeEnvironment
Creates runtime environment in the repository

Parameters **environment** – runtime environment to create

Returns created runtime environment

Exception *errors.ExistingEnvironmentError* if given runtime environment has the same name as existing

get_environment(*name: str*) → Optional[ebonite.core.objects.core.RuntimeEnvironment]
Finds runtime environment by name.

Parameters **name** – expected runtime environment name

Returns found runtime environment if exists or *None*

get_environments() → List[ebonite.core.objects.core.RuntimeEnvironment]
Gets a list of runtime environments

Returns found runtime environments

get_image(*image_name: str, task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → Optional[Image]
Finds image by name in given model, task and project.

Parameters

- **image_name** – expected image name
- **task** – task to search for image in
- **project** – project to search for image in

Returns found image if exists or *None*

get_images(*task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → List[Image]
Gets a list of images in given model, task and project

Parameters

- **task** – task to search for images in
- **project** – project to search for images in

Returns found images

get_instance(*instance_name: str, image: Union[int, Image], environment: Union[int, RuntimeEnvironment]*) → Optional[ebonite.core.objects.core.RuntimeInstance]
Finds instance by name in given image and environment.

Parameters

- **instance_name** – expected instance name
- **image** – image (or id) to search for instance in
- **environment** – environment (or id) to search for instance in

Returns found instance if exists or *None*

get_instances(*image: Union[int, Image] = None, environment: Union[int, RuntimeEnvironment] = None*) → List[ebonite.core.objects.core.RuntimeInstance]
Gets a list of instances in given image or environment

Parameters

- **image** – image (or id) to search for instances in
- **environment** – environment (or id) to search for instances in

Returns found instances

get_models(*task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → List[Model]
Gets a list of models in given project and task

Parameters

- **task** – task to search for models in
- **project** – project to search for models in

Returns found models

get_or_create_project(*name: str*) → ebonite.core.objects.core.Project
Creates a project if not exists or gets existing project otherwise.

Parameters **name** – project name

Returns project

get_or_create_task(*project: str, task_name: str*) → ebonite.core.objects.core.Task
Creates a task if not exists or gets existing task otherwise.

Parameters

- **project** – project to search/create task in
- **task_name** – expected name of task

Returns created/found task

get_pipeline(*pipeline_name: str, task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → Optional[Pipeline]
Finds model by name in given task and project.

Parameters

- **pipeline_name** – expected pipeline name
- **task** – task to search for pipeline in
- **project** – project to search for pipeline in

Returns found pipeline if exists or *None*

get_pipelines(*task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → List[Pipeline]
Gets a list of pipelines in given project and task

Parameters

- **task** – task to search for models in
- **project** – project to search for models in

Returns found pipelines

get_project(*name: str*) → Optional[ebonite.core.objects.core.Project]
Finds project in the repository by name

Parameters **name** – name of the project to return

Returns found project if exists or *None*

get_projects() → List[ebonite.core.objects.core.Project]
Gets all projects in the repository

Returns all projects in the repository

get_task(*project: Union[int, str, core.Project], task_name: str*) → Optional[Task]
Finds task with given name in given project

Parameters

- **project** – project to search for task in
- **task_name** – expected name of task

Returns task if exists or *None*

get_tasks(*project: Union[int, str, core.Project]*) → List[Task]
Gets a list of tasks for given project

Parameters **project** – project to search for tasks in

Returns project tasks

delete_project(*project: ebonite.core.objects.core.Project*, *cascade: bool = False*)
” Deletes project and(if required) all tasks associated with it from metadata repository

Parameters

- **project** – project to delete
- **cascade** – whether should project be deleted with all associated tasks

Returns Nothing

delete_task(*task: ebonite.core.objects.core.Task*, *cascade: bool = False*)
” Deletes task from metadata

Parameters

- **task** – task to delete
- **cascade** – whether should task be deleted with all associated objects

Returns Nothing

delete_model(*model: ebonite.core.objects.core.Model*, *force: bool = False*)
" Deletes model from metadata and artifact repositories

Parameters

- **model** – model to delete
- **force** – whether model artifacts' deletion errors should be ignored, default is false

Returns Nothing

delete_pipeline(*pipeline: ebonite.core.objects.core.Pipeline*)
"Deletes pipeline from metadata

Parameters **pipeline** – pipeline to delete

delete_image(*image: ebonite.core.objects.core.Image*, *meta_only: bool = False*, *cascade: bool = False*)
" Deletes existing image from metadata repository and image provider

Parameters

- **image** – image ot delete
- **meta_only** – should image be deleted only from metadata
- **cascade** – whether to delete nested RuntimeInstances

delete_instance(*instance: ebonite.core.objects.core.RuntimeInstance*, *meta_only: bool = False*)
" Stops instance of model service and deletes it from repository

Parameters

- **instance** – instance to delete
- **meta_only** – only remove from metadata, do not stop instance

Returns nothing

delete_environment(*environment: ebonite.core.objects.core.RuntimeEnvironment*, *meta_only: bool = False*, *cascade: bool = False*)
" Deletes environment from metadata repository and(if required) stops associated instances

Parameters

- **environment** – environment to delete
- **meta_only** – wheter to only delete metadata
- **cascade** – Whether should environment be deleted with all associated instances

Returns Nothing

create_dataset(*data*, *target=None*)

create_metric(*metric_obj*)

ebonite.**start_runtime**(*loader=None*, *server=None*)
Starts Ebonite runtime for given (optional) loader and (optional) server

Parameters

- **loader** – loader of model to start Ebonite runtime for, if not given class specified in `config.Runtime.LOADER` is used
- **server** – server to use for Ebonite runtime, default is a flask-based server, if not given class specified in `config.Runtime.SERVER` is used

Returns nothing

`ebonite.`**create_model**(*model_object*, *input_data*, *model_name: str = None*, *params: Dict[str, Any] = None*, *description: str = None*, ***kwargs*) → ebonite.core.objects.core.Model

Creates Model instance from arbitrary model objects and sample of input data

Parameters

- **model_object** – model object (function, sklearn model, tensorflow output tensor list etc)
- **input_data** – sample of input data (numpy array, pandas dataframe, feed dict etc)
- **model_name** – name for model in database, if not provided will be autogenerated
- **params** – dict with arbitrary parameters. Must be json-serializable
- **description** – text description of this model
- **kwargs** – other arguments for model (see Model.create)

Returns *Model* instance

4.1 Subpackages

4.1.1 ebonite.build package

class `ebonite.build.`**RunnerBase**

Bases: `object`

instance_type() → Type[ebonite.core.objects.core.RuntimeInstance.Params]

Returns subtype of *RuntimeInstance.Params* supported by this runner

create_instance(*name: str*, ***kwargs*) → ebonite.core.objects.core.RuntimeInstance.Params

Creates new runtime instance on given name and args

Parameters **name** – name of instance to use

Returns created *RuntimeInstance.Params* subclass instance

run(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *image: ebonite.core.objects.core.Image.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)

Runs given image on given environment with params given by instance

Parameters

- **instance** – instance params to use for running
- **image** – image to base instance on
- **env** – environment to run on

is_running(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*) → bool

Checks that given instance is running on given environment

Parameters

- **instance** – instance to check running of
- **env** – environment to check running on

Returns "is running" flag

stop(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Stops running of given instance on given environment

Parameters

- **instance** – instance to stop running of
- **env** – environment to stop running on

logs(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*) → Generator[str, None, None]
Exposes logs produced by given instance while running on given environment

Parameters

- **instance** – instance to expose logs for
- **env** – environment to expose logs from

Returns generator of log strings or string with logs

instance_exists(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*) → bool
Checks if instance exists in environment

Parameters

- **instance** – instance params to check
- **env** – environment to check in

Returns boolean flag

remove_instance(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Removes instance

Parameters

- **instance** – instance params to remove
- **env** – environment to remove from

class ebonite.build.**BuilderBase**
Bases: object

Abstract class for building images from ebonite objects

create_image(*name: str*, *environment: ebonite.core.objects.core.RuntimeEnvironment*, ***kwargs*) → ebonite.core.objects.core.Image.Params
Abstract method to create image

build_image(*buildable: ebonite.core.objects.core.Buildable*, *image: ebonite.core.objects.core.Image.Params*, *environment: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Abstract method to build image

delete_image(*image: ebonite.core.objects.core.Image.Params*, *environment: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Abstract method to delete image

image_exists(*image: ebonite.core.objects.core.Image.Params*, *environment: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Abstract method to check if image exists

class ebonite.build.**PythonBuildContext**(*provider: ebonite.build.provider.base.PythonProvider*)
Bases: object

Basic class for building python images from ebonite objects

Parameters **provider** – A ProviderBase instance to get distribution from

class ebonite.build.**PipelineProvider**(*pipeline: ebonite.core.objects.core.Pipeline*, *server: ebonite.runtime.server.base.Server*, *debug: bool = False*)
Bases: ebonite.build.provider.base.PythonProvider

Provider to build service from Pipeline object

Parameters

- **pipeline** – Pipeline instance to build from
- **server** – Server instance to build with
- **debug** – Whether to run image in debug mode

get_requirements() → ebonite.core.objects.requirements.Requirements
Returns union of model, server and loader requirements

get_sources()
Returns model metadata file and sources of custom modules from requirements

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Return model binaries

get_python_version()

Returns version of python that produced model

class ebonite.build.**MLModelMultiProvider**(*models: List[ebonite.core.objects.core.Model]*, *server: ebonite.runtime.server.base.Server*, *debug: bool = False*)
Bases: *ebonite.build.provider.ml_model.MLModelProvider*

Provider to put multiple models in one service

Parameters

- **models** – List of Model instances
- **server** – Server instance to build with
- **debug** – Debug for instance

get_requirements()
Returns union of server, loader and all models requirements

get_sources()
Returns models meta file and custom requirements

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Returns binaries of models artifacts

get_python_version()

Returns version of python that produced model

class ebonite.build.**MLModelProvider**(*model: ebonite.core.objects.core.Model*, *server: ebonite.runtime.server.base.Server*, *debug: bool = False*)

Bases: ebonite.build.provider.base.PythonProvider

Provider to build service from Model object

Parameters

- **model** – Model instance to build from
- **server** – Server instance to build with
- **debug** – Whether to run image in debug mode

get_requirements() → ebonite.core.objects.requirements.Requirements
Returns union of model, server and loader requirements

get_sources()
Returns model metadata file and sources of custom modules from requirements

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Return model binaries

get_python_version()

Returns version of python that produced model

Subpackages

ebonite.build.builder package

class ebonite.build.builder.**BuilderBase**

Bases: object

Abstract class for building images from ebonite objects

create_image(*name: str*, *environment: ebonite.core.objects.core.RuntimeEnvironment*, ***kwargs*) → ebonite.core.objects.core.Image.Params
Abstract method to create image

build_image(*buildable: ebonite.core.objects.core.Buildable*, *image: ebonite.core.objects.core.Image.Params*, *environment: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Abstract method to build image

delete_image(*image: ebonite.core.objects.core.Image.Params*, *environment: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Abstract method to delete image

image_exists(*image: ebonite.core.objects.core.Image.Params*, *environment: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Abstract method to check if image exists

class ebonite.build.builder.**PythonBuildContext**(*provider: ebonite.build.provider.base.PythonProvider*)

Bases: object

Basic class for building python images from ebonite objects

Parameters **provider** – A ProviderBase instance to get distribution from

ebonite.build.provider package

class ebonite.build.provider.**ProviderBase**
Bases: object

Base class for providers

get_sources() → Dict[str, str]
Abstract method for text files

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Abstract method for binaries

get_env() → Dict[str, str]
Abstract method for environment variables

get_options() → Dict[str, str]
Abstract method for additional build options

class ebonite.build.provider.**PythonProvider**(*server: ebonite.runtime.server.base.Server*, *loader: ebonite.runtime.interface.base.InterfaceLoader*, *debug: bool = False*)
Bases: ebonite.build.provider.base.ProviderBase

Provider for python-based builds. Includes python version and requirements

Parameters

- **server** – Server instance to build with
- **loader** – InterfaceLoader instance to build with
- **debug** – Whether to run image in debug mode

get_python_version()
Returns current python version

get_requirements() → ebonite.core.objects.requirements.Requirements
Abstract method for python requirements

get_env() → Dict[str, str]
Get env variables for image

get_options() → Dict[str, str]
Abstract method for additional build options

class ebonite.build.provider.**MLModelProvider**(*model: ebonite.core.objects.core.Model*, *server: ebonite.runtime.server.base.Server*, *debug: bool = False*)
Bases: ebonite.build.provider.base.PythonProvider

Provider to build service from Model object

Parameters

- **model** – Model instance to build from
- **server** – Server instance to build with
- **debug** – Whether to run image in debug mode

get_requirements() → ebonite.core.objects.requirements.Requirements
Returns union of model, server and loader requirements

get_sources()
Returns model metadata file and sources of custom modules from requirements

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Return model binaries

get_python_version()

Returns version of python that produced model

class ebonite.build.provider.**MLModelMultiProvider**(*models: List[ebonite.core.objects.core.Model], server: ebonite.runtime.server.base.Server, debug: bool = False*)
Bases: *ebonite.build.provider.ml_model.MLModelProvider*

Provider to put multiple models in one service

Parameters

- **models** – List of Model instances
- **server** – Server instance to build with
- **debug** – Debug for instance

get_requirements()
Returns union of server, loader and all models requirements

get_sources()
Returns models meta file and custom requirements

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Returns binaries of models artifacts

get_python_version()

Returns version of python that produced model

class ebonite.build.provider.**PipelineProvider**(*pipeline: ebonite.core.objects.core.Pipeline, server: ebonite.runtime.server.base.Server, debug: bool = False*)
Bases: ebonite.build.provider.base.PythonProvider

Provider to build service from Pipeline object

Parameters

- **pipeline** – Pipeline instance to build from
- **server** – Server instance to build with
- **debug** – Whether to run image in debug mode

get_requirements() → ebonite.core.objects.requirements.Requirements
Returns union of model, server and loader requirements

get_sources()
Returns model metadata file and sources of custom modules from requirements

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Return model binaries

get_python_version()

Returns version of python that produced model

Submodules

ebonite.build.provider.ml_model module

ebonite.build.provider.ml_model.**read**(*path*)

class ebonite.build.provider.ml_model.**MLModelProvider**(*model: ebonite.core.objects.core.Model*, *server: ebonite.runtime.server.base.Server*, *debug: bool = False*)

Bases: ebonite.build.provider.base.PythonProvider

Provider to build service from Model object

Parameters

- **model** – Model instance to build from
- **server** – Server instance to build with
- **debug** – Whether to run image in debug mode

get_requirements() → ebonite.core.objects.requirements.Requirements
Returns union of model, server and loader requirements

get_sources()
Returns model metadata file and sources of custom modules from requirements

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Return model binaries

get_python_version()

Returns version of python that produced model

class ebonite.build.provider.ml_model.**ModelBuildable**(*model_id: Union[int, ebonite.core.objects.core.Model]*, *server_type: str*, *debug: bool = False*)

Bases: *ebonite.build.provider.utils.BuildableWithServer*, *ebonite.core.objects.core.WithMetadataRepository*

task
property to get task (can be None, whick forces to provide task manually)

model

get_provider() → ebonite.build.provider.ml_model.MLModelProvider
Abstract method to get a provider for this Buildable

type = 'ebonite.build.provider.ml_model.ModelBuildable'

class ebonite.build.provider.ml_model.**BuildableModelHook**

Bases: *ebonite.core.analyzer.buildable.BuildableHook*, ebonite.core.analyzer.base.TypeHookMixin

valid_types = [<class 'ebonite.core.objects.core.Model'>]

process(*obj*, ***kwargs*)
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

ebonite.build.provider.ml_model_multi module

class ebonite.build.provider.ml_model_multi.**MLModelMultiProvider**(*models: List[ebonite.core.objects.core.Model], server: ebonite.runtime.server.base.Server, debug: bool = False*)

Bases: *ebonite.build.provider.ml_model.MLModelProvider*

Provider to put multiple models in one service

Parameters

- **models** – List of Model instances
- **server** – Server instance to build with
- **debug** – Debug for instance

get_requirements()
Returns union of server, loader and all models requirements

get_sources()
Returns models meta file and custom requirements

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Returns binaries of models artifacts

get_python_version()

Returns version of python that produced model

class ebonite.build.provider.ml_model_multi.**MultiModelBuildable**(*model_ids: List[Union[int, ebonite.core.objects.core.Model]], server_type: str, debug: bool = False*)

Bases: *ebonite.build.provider.utils.BuildableWithServer*, *ebonite.core.objects.core.WithMetadataRepository*

task
property to get task (can be None, which forces to provide task manually)

models

get_provider() → ebonite.build.provider.ml_model_multi.MLModelMultiProvider
Abstract method to get a provider for this Buildable

type = 'ebonite.build.provider.ml_model_multi.MultiModelBuildable'

class ebonite.build.provider.ml_model_multi.**BuildableMultiModelHook**
Bases: *ebonite.core.analyzer.buildable.BuildableHook*, ebonite.core.analyzer.base.CanIsAMustHookMixin

must_process(*obj*) → bool
Must return True if obj must be processed by this hook. "must" means you sure that no other hook should handle this object, for example this hook is for sklearn objects and obj is exactly that.

Parameters **obj** – object to analyze

Returns True or False

process(*obj*, ***kwargs*)
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

ebonite.build.provider.pipeline module

ebonite.build.provider.pipeline.**read**(*path*)

class ebonite.build.provider.pipeline.**PipelineProvider**(*pipeline: ebonite.core.objects.core.Pipeline*, *server: ebonite.runtime.server.base.Server*, *debug: bool = False*)
Bases: ebonite.build.provider.base.PythonProvider

Provider to build service from Pipeline object

Parameters

- **pipeline** – Pipeline instance to build from
- **server** – Server instance to build with
- **debug** – Whether to run image in debug mode

get_requirements() → ebonite.core.objects.requirements.Requirements
Returns union of model, server and loader requirements

get_sources()
Returns model metadata file and sources of custom modules from requirements

get_artifacts() → ebonite.core.objects.artifacts.ArtifactCollection
Return model binaries

get_python_version()

Returns version of python that produced model

class ebonite.build.provider.pipeline.**PipelineBuildable**(*pipeline_id: Union[int, ebonite.core.objects.core.Pipeline]*, *server_type: str*, *debug: bool = False*)
Bases: *ebonite.build.provider.utils.BuildableWithServer*

task
property to get task (can be None, whick forces to provide task manually)

pipeline

get_provider() → ebonite.build.provider.pipeline.PipelineProvider
Abstract method to get a provider for this Buildable

type = 'ebonite.build.provider.pipeline.PipelineBuildable'

class ebonite.build.provider.pipeline.**BuildableModelHook**
Bases: *ebonite.core.analyzer.buildable.BuildableHook*, ebonite.core.analyzer.base.TypeHookMixin

valid_types = [<class 'ebonite.core.objects.core.Pipeline'>]

process(*obj*, ***kwargs*)
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

ebonite.build.provider.utils module

class ebonite.build.provider.utils.**BuildableWithServer**(*server_type: str*)
Bases: *ebonite.core.objects.core.Buildable*

server

type = 'ebonite.build.provider.utils.BuildableWithServer'

ebonite.build.runner package

class ebonite.build.runner.**RunnerBase**
Bases: object

instance_type() → Type[ebonite.core.objects.core.RuntimeInstance.Params]

Returns subtype of *RuntimeInstance.Params* supported by this runner

create_instance(*name: str*, ***kwargs*) → ebonite.core.objects.core.RuntimeInstance.Params
Creates new runtime instance on given name and args

Parameters **name** – name of instance to use

Returns created *RuntimeInstance.Params* subclass instance

run(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *image: ebonite.core.objects.core.Image.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Runs given image on given environment with params given by instance

Parameters

- **instance** – instance params to use for running
- **image** – image to base instance on
- **env** – environment to run on

is_running(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*) → bool
Checks that given instance is running on given environment

Parameters

- **instance** – instance to check running of
- **env** – environment to check running on

Returns "is running" flag

stop(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Stops running of given instance on given environment

Parameters

- **instance** – instance to stop running of
- **env** – environment to stop running on

logs(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*) → Generator[str, None, None]
Exposes logs produced by given instance while running on given environment

Parameters

- **instance** – instance to expose logs for
- **env** – environment to expose logs from

Returns generator of log strings or string with logs

instance_exists(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*) → bool
Checks if instance exists in environment

Parameters

- **instance** – instance params to check
- **env** – environment to check in

Returns boolean flag

remove_instance(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Removes instance

Parameters

- **instance** – instance params to remove
- **env** – environment to remove from

4.1.2 ebonite.client package

class ebonite.client.**Ebonite**(*meta_repo: ebonite.repository.metadata.base.MetadataRepository*, *artifact_repo: ebonite.repository.artifact.base.ArtifactRepository*, *dataset_repo: ebonite.repository.dataset.base.DatasetRepository = None*)

Bases: object

Main entry point for ebonite

This is the client for Ebonite API. It can save, load and build Models, Tasks and Projects. Ebonite instance can be obtained from factory methods like *local()* for local client, *inmemory()* for inmemory client.

You can save client config with *save_client_config()* and later restore it with *from_config_file()*

Parameters

- **meta_repo** – *MetadataRepository* instance to save metadata
- **artifact_repo** – *ArtifactRepository* instance to save artifacts

default_server = None

default_env = None

push_model(*model: ebonite.core.objects.core.Model*, *task: ebonite.core.objects.core.Task = None*) → ebonite.core.objects.core.Model

Pushes *Model* instance into metadata and artifact repositories

Parameters

- **model** – *Model* instance
- **task** – *Task* instance to save model to. Optional if model already has

task :return: same saved *Model* instance

create_model(*model_object*, *model_input*, *model_name: str = None*, ***, *project_name: str = 'default_project'*, *task_name: str = 'default_task'*, ***kwargs*)

This function creates ebonite model. Creates model, task and project (if needed) and pushes it to repo

Parameters

- **model_object** – object containing model.
- **model_input** – model input.
- **model_name** – model name to create.
- **project_name** – project name.
- **task_name** – task name.
- **kwargs** – other arguments for model

Returns *Model* instance representing

get_model(*model_name: str, task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None, load_artifacts: bool = True*) → ebonite.core.objects.core.Model

Load model from repository

Parameters

- **model_name** – model name to load
- **task** – *Task* instance or task name to load model from
- **project** – *Project* instance or project name to load task from
- **load_artifacts** – if True, load model artifact into wrapper

Returns *Model* instance

create_image(*obj*, *name: str = None*, *task: ebonite.core.objects.core.Task = None*, *server: ebonite.runtime.server.base.Server = None*, *environment: ebonite.core.objects.core.RuntimeEnvironment = None*, *debug=False*, *skip_build=False*, *builder_args: Dict[str, object] = None*, ***kwargs*) → ebonite.core.objects.core.Image

Builds image of model service and stores it to repository

Parameters

- **obj** – model/list of models/pipeline or any object that has existing Hook for it to wrap into service
- **name** – name of image to build
- **task** – task to put image into
- **server** – server to build image with
- **environment** – env to build for
- **debug** – flag to build debug image
- **skip_build** – wheter to skip actual image build
- **builder_args** – kwargs for image.build
- **kwargs** – additional kwargs for builder

Returns *Image* instance representing built image

create_instance(*image: ebonite.core.objects.core.Image*, *name: str = None*, *environment: ebonite.core.objects.core.RuntimeEnvironment = None*, *run=False*, *runner_kwargs: Dict[str, object] = None*, ***instance_kwargs*) → ebonite.core.objects.core.RuntimeInstance

Runs model service instance and stores it to repository

Parameters

- **image** – image to run instance from
- **name** – name of instance to run
- **environment** – environment to run instance in, if no given *localhost* is used
- **run** – whether to automatically run instance after creation
- **runner_kwargs** – additional parameters for runner
- **instance_kwargs** – additional parameters for instance

Returns *RuntimeInstance* instance representing run instance

build_and_run_instance(*obj*, *name: str = None*, *task: ebonite.core.objects.core.Task = None*, *environment: ebonite.core.objects.core.RuntimeEnvironment = None*, *builder_kwargs: Dict[str, object] = None*, *runner_kwargs: Dict[str, object] = None*, *instance_kwargs: Dict[str, object] = None*) → ebonite.core.objects.core.RuntimeInstance

Builds image of model service, immediately runs service and stores both image and instance to repository

Parameters

- **obj** – buildable object to wrap into service
- **name** – name of image and instance to be built and run respectively
- **task** – task to put image into
- **environment** – environment to run instance in, if no given *localhost* is used
- **builder_kwargs** – additional kwargs for builder
- **runner_kwargs** – additional parameters for runner. Full list can be seen in https://docker-py.readthedocs.io/en/stable/containers.html
- **instance_kwargs** – additional parameters for instance

Returns *RuntimeInstance* instance representing run instance

classmethod local(*path=None*, *clear=False*) → ebonite.client.base.Ebonite
Get an instance of *Ebonite* that stores metadata and artifacts on local filesystem

Parameters

- **path** – path to storage dir. If None, *.ebonite* dir is used
- **clear** – if True, erase previous data from storage

classmethod inmemory() → ebonite.client.base.Ebonite
Get an instance of *Ebonite* with inmemory repositories

classmethod custom_client(*metadata: Union[str, ebonite.repository.metadata.base.MetadataRepository], artifact: Union[str, ebonite.repository.artifact.base.ArtifactRepository], meta_kwargs: dict = None, artifact_kwargs: dict = None*) → ebonite.client.base.Ebonite
Create custom Ebonite client from metadata and artifact repositories.

Parameters

- **metadata** – *MetadataRepository* instance or pyjackson subtype type name
- **artifact** – *ArtifactRepository* instance or pyjackson subtype type name
- **meta_kwargs** – kwargs for metadata repo __init__ if subtype type name was provided
- **artifact_kwargs** – kwargs for artifact repo __init__ if subtype type name was provided

Returns *Ebonite* instance

classmethod from_config_file(*filepath*) → ebonite.client.base.Ebonite
Read and create Ebonite instance from config file

Parameters **filepath** – path to read config from

Returns *Ebonite* instance

save_client_config(*filepath*)
Save current client config to a file

Parameters **filepath** – path to file

get_default_server()

Returns Default server implementation for this client

get_default_environment()
Creates (if needed) and returns default runtime environment

Returns saved instance of *RuntimeEnvironment*

push_environment(*environment: ebonite.core.objects.core.RuntimeEnvironment*) → ebonite.core.objects.core.RuntimeEnvironment
Creates runtime environment in the repository

Parameters **environment** – runtime environment to create

Returns created runtime environment

Exception *errors.ExistingEnvironmentError* if given runtime environment has the same name as existing

get_environment(*name: str*) → Optional[ebonite.core.objects.core.RuntimeEnvironment]
Finds runtime environment by name.

Parameters **name** – expected runtime environment name

Returns found runtime environment if exists or *None*

get_environments() → List[ebonite.core.objects.core.RuntimeEnvironment]
Gets a list of runtime environments

Returns found runtime environments

get_image(*image_name: str, task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → Optional[Image]
Finds image by name in given model, task and project.

Parameters

- **image_name** – expected image name
- **task** – task to search for image in
- **project** – project to search for image in

Returns found image if exists or *None*

get_images(*task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → List[Image]
Gets a list of images in given model, task and project

Parameters

- **task** – task to search for images in
- **project** – project to search for images in

Returns found images

get_instance(*instance_name: str, image: Union[int, Image], environment: Union[int, RuntimeEnvironment]*) → Optional[ebonite.core.objects.core.RuntimeInstance]
Finds instance by name in given image and environment.

Parameters

- **instance_name** – expected instance name
- **image** – image (or id) to search for instance in
- **environment** – environment (or id) to search for instance in

Returns found instance if exists or *None*

get_instances(*image: Union[int, Image] = None, environment: Union[int, RuntimeEnvironment] = None*) → List[ebonite.core.objects.core.RuntimeInstance]
Gets a list of instances in given image or environment

Parameters

- **image** – image (or id) to search for instances in
- **environment** – environment (or id) to search for instances in

Returns found instances

get_models(*task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → List[Model]
Gets a list of models in given project and task

Parameters

- **task** – task to search for models in
- **project** – project to search for models in

Returns found models

get_or_create_project(*name: str*) → ebonite.core.objects.core.Project
Creates a project if not exists or gets existing project otherwise.

Parameters **name** – project name

Returns project

get_or_create_task(*project: str*, *task_name: str*) → ebonite.core.objects.core.Task
Creates a task if not exists or gets existing task otherwise.

Parameters

- **project** – project to search/create task in
- **task_name** – expected name of task

Returns created/found task

get_pipeline(*pipeline_name: str, task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → Optional[Pipeline]
Finds model by name in given task and project.

Parameters

- **pipeline_name** – expected pipeline name
- **task** – task to search for pipeline in
- **project** – project to search for pipeline in

Returns found pipeline if exists or *None*

get_pipelines(*task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → List[Pipeline]
Gets a list of pipelines in given project and task

Parameters

- **task** – task to search for models in
- **project** – project to search for models in

Returns found pipelines

get_project(*name: str*) → Optional[ebonite.core.objects.core.Project]
Finds project in the repository by name

Parameters **name** – name of the project to return

Returns found project if exists or *None*

get_projects() → List[ebonite.core.objects.core.Project]
Gets all projects in the repository

Returns all projects in the repository

get_task(*project: Union[int, str, core.Project], task_name: str*) → Optional[Task]
Finds task with given name in given project

Parameters

- **project** – project to search for task in
- **task_name** – expected name of task

Returns task if exists or *None*

get_tasks(*project: Union[int, str, core.Project]*) → List[Task]
Gets a list of tasks for given project

Parameters **project** – project to search for tasks in

Returns project tasks

delete_project(*project: ebonite.core.objects.core.Project*, *cascade: bool = False*)
" Deletes project and(if required) all tasks associated with it from metadata repository

Parameters

- **project** – project to delete
- **cascade** – whether should project be deleted with all associated tasks

Returns Nothing

delete_task(*task: ebonite.core.objects.core.Task*, *cascade: bool = False*)
" Deletes task from metadata

Parameters

- **task** – task to delete
- **cascade** – whether should task be deleted with all associated objects

Returns Nothing

delete_model(*model: ebonite.core.objects.core.Model*, *force: bool = False*)
" Deletes model from metadata and artifact repositories

Parameters

- **model** – model to delete
- **force** – whether model artifacts' deletion errors should be ignored, default is false

Returns Nothing

delete_pipeline(*pipeline: ebonite.core.objects.core.Pipeline*)
"Deletes pipeline from metadata

Parameters **pipeline** – pipeline to delete

delete_image(*image: ebonite.core.objects.core.Image*, *meta_only: bool = False*, *cascade: bool = False*)
" Deletes existing image from metadata repository and image provider

Parameters

- **image** – image ot delete
- **meta_only** – should image be deleted only from metadata
- **cascade** – whether to delete nested RuntimeInstances

delete_instance(*instance: ebonite.core.objects.core.RuntimeInstance*, *meta_only: bool = False*)
" Stops instance of model service and deletes it from repository

Parameters

- **instance** – instance to delete
- **meta_only** – only remove from metadata, do not stop instance

Returns nothing

delete_environment(*environment: ebonite.core.objects.core.RuntimeEnvironment*, *meta_only: bool = False*, *cascade: bool = False*)
" Deletes environment from metadata repository and(if required) stops associated instances

Parameters

- **environment** – environment to delete
- **meta_only** – wheter to only delete metadata
- **cascade** – Whether should environment be deleted with all associated instances

Returns Nothing

create_dataset(*data*, *target=None*)

create_metric(*metric_obj*)

ebonite.client.**create_model**(*model_object*, *input_data*, *model_name: str = None*, *params: Dict[str, Any] = None*, *description: str = None*, ***kwargs*) → ebonite.core.objects.core.Model
Creates Model instance from arbitrary model objects and sample of input data

Parameters

- **model_object** – model object (function, sklearn model, tensorflow output tensor list etc)
- **input_data** – sample of input data (numpy array, pandas dataframe, feed dict etc)
- **model_name** – name for model in database, if not provided will be autogenerated
- **params** – dict with arbitrary parameters. Must be json-serializable
- **description** – text description of this model
- **kwargs** – other arguments for model (see Model.create)

Returns *Model* instance

Submodules

ebonite.client.autogen module

ebonite.client.autogen.**find_exposed_methods**(*base_class*, *new_only=True*) → List[ebonite.client.expose.ExposedMethod]

ebonite.client.autogen.**patch**(*classes*, *filename*, *dry_run=True*)

ebonite.client.autogen.**clear**(*filename*, *dry_run=True*)

ebonite.client.autogen.**main**()

ebonite.client.expose module

class ebonite.client.expose.**ExposedMethod**(*name: str = None*)
Bases: object

original_name

generate_code()
Generate method code

get_declaration()

get_signature()

ebonite.client.expose.**get_exposed_method**(*f*) → Optional[ebonite.client.expose.ExposedMethod]

4.1.3 ebonite.core package

Subpackages

ebonite.core.analyzer package

class ebonite.core.analyzer.**Hook**
Bases: abc.ABC

Base class for Hooks

can_process(*obj*) → bool
Must return True if obj can be processed by this hook

Parameters **obj** – object to analyze

Returns True or False

must_process(*obj*) → bool
Must return True if obj must be processed by this hook. "must" means you sure that no other hook should handle this object, for example this hook is for sklearn objects and obj is exactly that.

Parameters **obj** – object to analyze

Returns True or False

process(*obj*, ***kwargs*)
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

ebonite.core.analyzer.**analyzer_class**(*hook_type: type*, *return_type: type*)
Function to create separate hook hierarchies for analyzing different objects

Parameters

- **hook_type** – Subtype of *Hook*
- **return_type** – Type that this hierarchy will use as analysis result

Returns Analyzer type

class ebonite.core.analyzer.**CanIsAMustHookMixin**
Bases: ebonite.core.analyzer.base.Hook

Mixin for cases when can_process equals to must_process

can_process(*obj*) → bool
Returns same as *Hook.must_process()*

class ebonite.core.analyzer.**BaseModuleHookMixin**
Bases: ebonite.core.analyzer.base.CanIsAMustHookMixin, ebonite.core.analyzer.base.Hook

Mixin for cases when hook must process all objects with certain base modules

is_valid_base_module_name(*module_name: str*) → bool
Must return True if module_name is valid for this hook

Parameters **module_name** – module name

Returns True or False

is_valid_base_module(*base_module: module*) → bool
Returns True if module is valid

Parameters **base_module** – module object

Returns True or False

must_process(*obj*)
Returns True if obj has valid base module

class ebonite.core.analyzer.**TypeHookMixin**
Bases: ebonite.core.analyzer.base.CanIsAMustHookMixin

Mixin for cases when hook must process objects of certain types

valid_types = None

must_process(*obj*) → bool
Returns True if obj is instance of one of valid types

Submodules

ebonite.core.analyzer.buildable module

class ebonite.core.analyzer.buildable.**BuildableHook**
Bases: ebonite.core.analyzer.base.Hook, abc.ABC

ebonite.core.analyzer.dataset module

class ebonite.core.analyzer.dataset.**DatasetHook**
Bases: ebonite.core.analyzer.base.Hook

Base hook type for DatasetAnalyzer. Analysis result is an instance of *DatasetType*

process(*obj*, ***kwargs*) → ebonite.core.objects.dataset_type.DatasetType
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

class ebonite.core.analyzer.dataset.**PrimitivesHook**
Bases: *ebonite.core.analyzer.dataset.DatasetHook*

Hook for primitive data, for example when you model outputs just one int

can_process(*obj*)
Must return True if obj can be processed by this hook

Parameters **obj** – object to analyze

Returns True or False

must_process(*obj*)
Must return True if obj must be processed by this hook. "must" means you sure that no other hook should handle this object, for example this hook is for sklearn objects and obj is exactly that.

Parameters **obj** – object to analyze

Returns True or False

process(*obj*, ***kwargs*) → ebonite.core.objects.dataset_type.DatasetType
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

class ebonite.core.analyzer.dataset.**OrderedCollectionHookDelegator**
Bases: *ebonite.core.analyzer.dataset.DatasetHook*

Hook for list/tuple data

can_process(*obj*) → bool
Must return True if obj can be processed by this hook

Parameters **obj** – object to analyze

Returns True or False

must_process(*obj*) → bool
Must return True if obj must be processed by this hook. "must" means you sure that no other hook should handle this object, for example this hook is for sklearn objects and obj is exactly that.

Parameters **obj** – object to analyze

Returns True or False

process(*obj*, ***kwargs*) → ebonite.core.objects.dataset_type.DatasetType
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

class ebonite.core.analyzer.dataset.**DictHookDelegator**
Bases: *ebonite.core.analyzer.dataset.DatasetHook*

Hook for dict data

can_process(*obj*) → bool
Must return True if obj can be processed by this hook

Parameters **obj** – object to analyze

Returns True or False

must_process(*obj*) → bool
Must return True if obj must be processed by this hook. "must" means you sure that no other hook should handle this object, for example this hook is for sklearn objects and obj is exactly that.

Parameters **obj** – object to analyze

Returns True or False

process(*obj*, ***kwargs*) → ebonite.core.objects.dataset_type.DatasetType
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

class ebonite.core.analyzer.dataset.**BytesDatasetHook**
Bases: *ebonite.core.analyzer.dataset.DatasetHook*

Hook for bytes objects

process(*obj*, ***kwargs*) → ebonite.core.objects.dataset_type.DatasetType
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

can_process(*obj*) → bool
Must return True if obj can be processed by this hook

Parameters **obj** – object to analyze

Returns True or False

must_process(*obj*) → bool
Must return True if obj must be processed by this hook. "must" means you sure that no other hook should handle this object, for example this hook is for sklearn objects and obj is exactly that.

Parameters **obj** – object to analyze

Returns True or False

ebonite.core.analyzer.metric module

class ebonite.core.analyzer.metric.**MetricHook**
Bases: ebonite.core.analyzer.base.Hook

Base hook type for DatasetAnalyzer. Analysis result is an instance of *DatasetType*

process(*obj*, ***kwargs*) → ebonite.core.objects.metric.Metric
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

class ebonite.core.analyzer.metric.**LibFunctionMixin**
Bases: *ebonite.core.analyzer.metric.MetricHook*, ebonite.core.analyzer.base.LibHookMixin

invert = False

default_args = {}

get_args(*obj*)

process(*obj*, ***kwargs*) → ebonite.core.objects.metric.Metric
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

class ebonite.core.analyzer.metric.**CallableMetricHook**
Bases: *ebonite.core.analyzer.metric.MetricHook*

process(*obj*, ***kwargs*) → ebonite.core.objects.metric.Metric
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

can_process(*obj*) → bool
Must return True if obj can be processed by this hook

Parameters **obj** – object to analyze

Returns True or False

must_process(*obj*) → bool
Must return True if obj must be processed by this hook. "must" means you sure that no other hook should handle this object, for example this hook is for sklearn objects and obj is exactly that.

Parameters **obj** – object to analyze

Returns True or False

ebonite.core.analyzer.model module

class ebonite.core.analyzer.model.**ModelHook**
Bases: ebonite.core.analyzer.base.Hook

Base hook type for ModelAnalyzer. Analysis result is an instance of *ModelWrapper*

valid_types = None

process(*obj*, ***kwargs*) → ebonite.core.objects.wrapper.ModelWrapper
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

class ebonite.core.analyzer.model.**BindingModelHook**
Bases: *ebonite.core.analyzer.model.ModelHook*

Binding model hook which *process* by first creating corresponding model wrapper (by means of a subclass) and then binding created wrapper to given model object

process(*obj*, ***kwargs*) → ebonite.core.objects.wrapper.ModelWrapper
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

class ebonite.core.analyzer.model.**CallableMethodModelHook**
Bases: *ebonite.core.analyzer.model.BindingModelHook*

Hook for processing functions

can_process(*obj*) → bool
Must return True if obj can be processed by this hook

Parameters **obj** – object to analyze

Returns True or False

must_process(*obj*) → bool
Must return True if obj must be processed by this hook. "must" means you sure that no other hook should handle this object, for example this hook is for sklearn objects and obj is exactly that.

Parameters **obj** – object to analyze

Returns True or False

ebonite.core.analyzer.requirement module

class ebonite.core.analyzer.requirement.**RequirementAnalyzer**
Bases: object

Analyzer for RequirementHook hooks

hooks = []

classmethod analyze(*obj: Union[ebonite.core.objects.requirements.Requirements, ebonite.core.objects.requirements.Requirement, List[ebonite.core.objects.requirements.Requirement], str, List[str]]*) → ebonite.core.objects.requirements.Requirements
Run RequirementHook hooks to analyze obj

Parameters **obj** – objects to analyze

Returns Instance of Requirements

class ebonite.core.analyzer.requirement.**RequirementHook**
Bases: ebonite.core.analyzer.base.Hook

must_process(*obj: ebonite.core.objects.requirements.Requirement*) → bool
Must return True if obj must be processed by this hook. "must" means you sure that no other hook should handle this object, for example this hook is for sklearn objects and obj is exactly that.

Parameters **obj** – object to analyze

Returns True or False

can_process(*obj: ebonite.core.objects.requirements.Requirement*) → bool
Must return True if obj can be processed by this hook

Parameters **obj** – object to analyze

Returns True or False

process(*obj: ebonite.core.objects.requirements.Requirement*, ***kwargs*) → ebonite.core.objects.requirements.Requirements
Analyzes obj and returns result. Result type is determined by specific Hook class sub-hierarchy

Parameters

- **obj** – object to analyze
- **kwargs** – additional information to be used for analysis

Returns analysis result

ebonite.core.objects package

class ebonite.core.objects.**Project**(*name: str*, *id: int = None*, *author: str = None*, *creation_date: datetime.datetime = None*)
Bases: *ebonite.core.objects.core.EboniteObject*

Project is a collection of tasks

Parameters

- **id** – project id
- **name** – project name
- **author** – user that created that project
- **creation_date** – date when this project was created

delete(*cascade: bool = False*)
Deletes project and(if required) all tasks associated with it from metadata repository

Parameters **cascade** – whether should project be deleted with all associated tasks

Returns Nothing

add_task(*task: ebonite.core.objects.core.Task*)
Add task to project and save it to meta repo

Parameters **task** – task to add

add_tasks(*tasks: List[Task]*)
Add multiple tasks and save them to meta repo

Parameters **tasks** – tasks to add

delete_task(*task: ebonite.core.objects.core.Task*, *cascade: bool = False*)
Remove task from this project and delete it from meta repo

Parameters

- **cascade** – whether task should be deleted with all nested objects
- **task** – task to delete

save()
Saves object state to metadata repository

has_children()
Checks if object has existing relationship

class ebonite.core.objects.**Requirements**(*requirements: List[ebonite.core.objects.requirements.Requirement] = None*)
Bases: ebonite.core.objects.base.EboniteParams

A collection of requirements

Parameters **requirements** – list of *Requirement* instances

installable
List of installable requirements

custom
List of custom requirements

of_type(*type_: Type[T]*) → List[T]

Parameters **type** – type of requirements

Returns List of requirements of type *type_*

modules
List of module names

add(*requirement: ebonite.core.objects.requirements.Requirement*)
Adds requirement to this collection

Parameters **requirement** – *Requirement* instance to add

to_pip() → List[str]

Returns list of pip installable packages

class ebonite.core.objects.**Requirement**
Bases: *ebonite.core.objects.requirements.Requirement*, pyjackson.decorators.SubtypeRegisterMixin

Base class for python requirement

type = 'pyjackson.decorators.Requirement'

class ebonite.core.objects.**ArtifactCollection**
Bases: *ebonite.core.objects.artifacts.ArtifactCollection*, pyjackson.decorators.SubtypeRegisterMixin

Base class for artifact collection. Artifact collection is a number of named artifacts, represented by Blob's

Must be pyjackson-able

type = 'pyjackson.decorators.ArtifactCollection'

class ebonite.core.objects.**ModelWrapper**(*io: ebonite.core.objects.wrapper.ModelIO*)
Bases: *ebonite.core.objects.wrapper.ModelWrapper*, pyjackson.decorators.SubtypeRegisterMixin

Base class for model wrapper. Wrapper is an object that can save, load and inference a model

Must be pyjackson-serializable

type = 'pyjackson.decorators.ModelWrapper'

class ebonite.core.objects.**Task**(*name: str*, *id: int = None*, *project_id: int = None*, *author: str = None*, *creation_date: datetime.datetime = None*, *datasets: Dict[str, ebonite.core.objects.dataset_source.DatasetSource] = None*, *metrics: Dict[str, ebonite.core.objects.metric.Metric] = None*, *evaluation_sets: Dict[str, ebonite.core.objects.core.EvaluationSet] = None*)

Bases: *ebonite.core.objects.core.EboniteObject*, *ebonite.core.objects.core.WithDatasetRepository*

Task is a collection of models

Parameters

- **id** – task id
- **name** – task name
- **project_id** – parent project id for this task
- **author** – user that created that task
- **creation_date** – date when this task was created

project

delete(*cascade: bool = False*)

Deletes task from metadata

Parameters **cascade** – whether should task be deleted with all associated objects

Returns Nothing

add_model(*model: ebonite.core.objects.core.Model*)

Add model to task and save it to meta repo

Parameters **model** – model to add

add_models(*models: List[Model]*)

Add multiple models and save them to meta repo

Parameters **models** – models to add

delete_model(*model: ebonite.core.objects.core.Model*, *force=False*)

Remove model from this task and delete it from meta repo

Parameters

- **model** – model to delete
- **force** – whether model artifacts' deletion errors should be ignored, default is false

create_and_push_model(*model_object*, *input_data*, *model_name: str = None*, ***kwargs*) → ebonite.core.objects.core.Model

Create *Model* instance from model object and push it to repository

Parameters

- **model_object** – model object to build Model from
- **input_data** – input data sample to determine structure of inputs and outputs for given model
- **model_name** – name for model
- **kwargs** – other *create()* arguments

Returns created *Model*

push_model(*model: ebonite.core.objects.core.Model*) → ebonite.core.objects.core.Model
Push *Model* instance to task repository

Parameters **model** – *Model* to push

Returns same pushed *Model*

add_pipeline(*pipeline: ebonite.core.objects.core.Pipeline*)
Add model to task and save it to meta repo

Parameters **pipeline** – pipeline to add

add_pipelines(*pipelines: List[Pipeline]*)
Add multiple models and save them to meta repo

Parameters **pipelines** – pipelines to add

delete_pipeline(*pipeline: ebonite.core.objects.core.Pipeline*)
Remove model from this task and delete it from meta repo

Parameters **pipeline** – pipeline to delete

add_image(*image: ebonite.core.objects.core.Image*)
Add image for model and save it to meta repo

Parameters **image** – image to add

add_images(*images: List[Image]*)
Add multiple images for model and save them to meta repo

Parameters **images** – images to add

delete_image(*image: ebonite.core.objects.core.Image*, *meta_only: bool = False*, *cascade: bool = False*)
Remove image from this model and delete it from meta repo

Parameters

- **image** – image to delete
- **meta_only** – should image be deleted only from metadata
- **cascade** – whether image should be deleted with all instances

save()
Saves task to meta repository and pushes unsaved datasets to dataset repository

has_children()
Checks if object has existing relationship

add_evaluation(*name: str, data: Union[str, ebonite.core.objects.dataset_source.AbstractDataset, ebonite.core.objects.dataset_source.DatasetSource, Any], target: Union[str, ebonite.core.objects.dataset_source.AbstractDataset, ebonite.core.objects.dataset_source.DatasetSource, Any], metrics: Union[str, ebonite.core.objects.metric.Metric, Any, List[Union[str, ebonite.core.objects.metric.Metric, Any]]]*)
Adds new evaluation set to this task

Parameters

- **name** – name of the evaluation set
- **data** – input dataset for evaluation
- **target** – ground truth for input data

- **metrics** – one or more metrics to measure

delete_evaluation(*name: str*, *save: bool = True*)
Deletes evaluation set from task

Parameters

- **name** – name of the evaluation to delete
- **save** – also update task metadata in repo

add_dataset(*name, dataset: Union[ebonite.core.objects.dataset_source.DatasetSource, ebonite.core.objects.dataset_source.AbstractDataset, Any]*)
Adds new dataset to this task

Parameters

- **name** – name of the dataset
- **dataset** – Dataset, DatasetSource or raw dataset object

push_datasets()
Pushes all unsaved datasets to dataset repository

delete_dataset(*name: str*, *force: bool = False*, *save: bool = True*)
Deletes dataset from task with artifacts

Parameters

- **name** – name of the dataset to delete
- **force** – wheter to check evalsets that use this dataset and remove them or raise error
- **save** – also update task metadata in repo

add_metric(*name, metric: Union[ebonite.core.objects.metric.Metric, Any]*)
Adds metric to this task

Parameters

- **name** – name of the metric
- **metric** – Metric or raw metric object

delete_metric(*name: str*, *force: bool = False*, *save: bool = True*)
Deletes metric from task

Parameters

- **name** – name of the metric to delete
- **force** – wheter to check evalsets that use this metric and remove them or raise error
- **save** – also update task metadata in repo

evaluate_all(*force=False*, *save_result=True*) → Dict[str, ebonite.core.objects.core.EvaluationResult]
Evaluates all viable pairs of evalsets and models/pipelines

Parameters

- **force** – force reevaluate already evaluated
- **save_result** – save evaluation results to meta

class ebonite.core.objects.**Image**(*name: Optional[str], source: ebonite.core.objects.core.Buildable, id: int = None, params: ebonite.core.objects.core.Image.Params = None, author: str = None, creation_date: datetime.datetime = None, task_id: int = None, environment_id: int = None*)

Bases: ebonite.core.objects.core._WithBuilder

Class that represents metadata for image built from Buildable Actual type of image depends on *.params* field type

Parameters

- **name** – name of the image
- **id** – id of the image
- **author** – author of the image
- **source** – *Buildable* instance this image was built from
- **params** – *Image.Params* instance
- **task_id** – task.id this image belongs to
- **environment_id** – environment.id this image belongs to

Parma creation_date creation date of the image

class Params

Bases: ebonite.core.objects.core.Params, pyjackson.decorators.SubtypeRegisterMixin

Abstract class that represents different types of images

type = 'pyjackson.decorators.Params'

task

delete(*meta_only: bool = False*, *cascade: bool = False*)

Deletes existing image from metadata repository and image provider

Parameters

- **meta_only** – should image be deleted only from metadata
- **cascade** – whether to delete nested RuntimeInstances

bind_meta_repo(*repo: ebonite.repository.metadata.base.MetadataRepository*)

is_built() → bool

Checks if image was built and wasn't removed

build(***kwargs*)

Build this image

Parameters **kwargs** – additional params for builder.build_image (depends on builder implementation)

remove(***kwargs*)

remove this image (from environment, not from ebonite metadata)

save()

Saves object state to metadata repository

has_children()

Checks if object has existing relationship

class ebonite.core.objects.**Model**(*name: str*, *wrapper_meta: Optional[dict] = None*, *artifact: Optional[ebonite.core.objects.artifacts.ArtifactCollection] = None*, *requirements: ebonite.core.objects.requirements.Requirements = None*, *params: Dict[str, Any] = None*, *description: str = None*, *id: int = None*, *task_id: int = None*, *author: str = None*, *creation_date: datetime.datetime = None*, *evaluations: Dict[str, Dict[str, ebonite.core.objects.core.EvaluationResultCollection]] = None*)

Bases: ebonite.core.objects.core._InTask

Model contains metadata for machine learning model

Parameters

- **name** – model name
- **wrapper_meta** – *ModelWrapper* instance for this model
- **artifact** – *ArtifactCollection* instance with model artifacts
- **requirements** – *Requirements* instance with model requirements
- **params** – dict with arbitrary parameters. Must be json-serializable
- **description** – text description of this model
- **id** – model id
- **task_id** – parent task_id
- **author** – user that created that model
- **creation_date** – date when this model was created

PYTHON_VERSION = 'python_version'

load()
Load model artifacts into wrapper

ensure_loaded()
Ensure that wrapper has loaded model object

wrapper

with_wrapper(*wrapper: ebonite.core.objects.wrapper.ModelWrapper*)
Bind wrapper instance to this Model

Parameters **wrapper** – *ModelWrapper* instance

Returns self

wrapper_meta
pyjackson representation of *ModelWrapper* for this model: e.g., this provides possibility to move a model between repositories without its dependencies being installed

Type return

with_wrapper_meta(*wrapper_meta: dict*)
Bind wrapper_meta dict to this Model

Parameters **wrapper_meta** – dict with serialized *ModelWrapper* instance

Returns self

artifact
persisted artifacts if any

Type return

artifact_any
artifacts in any state (persisted or not)

Type return

artifact_req_persisted
Similar to *artifact* but checks that no unpersisted artifacts are left

Returns persisted artifacts if any

attach_artifact(*artifact: ebonite.core.objects.artifacts.ArtifactCollection*)

Parameters **artifact** – artifacts to attach to model in an unpersisted state

persist_artifacts(*persister: Callable[[ArtifactCollection], ArtifactCollection]*)
Model artifacts persisting workflow

Parameters **persister** – external object which stores model artifacts

without_artifacts() → ebonite.core.objects.core.Model

Returns copy of the model with no artifacts attached

classmethod create(*model_object, input_data, model_name: str = None, params: Dict[str, Any] = None, description: str = None, additional_artifacts: ebonite.core.objects.artifacts.ArtifactCollection = None, additional_requirements: Union[ebonite.core.objects.requirements.Requirements, ebonite.core.objects.requirements.Requirement, List[ebonite.core.objects.requirements.Requirement], str, List[str]] = None, custom_wrapper: ebonite.core.objects.wrapper.ModelWrapper = None, custom_artifact: ebonite.core.objects.artifacts.ArtifactCollection = None, custom_requirements: Union[ebonite.core.objects.requirements.Requirements, ebonite.core.objects.requirements.Requirement, List[ebonite.core.objects.requirements.Requirement], str, List[str]] = None*) → ebonite.core.objects.core.Model
Creates Model instance from arbitrary model objects and sample of input data

Parameters

- **model_object** – The model object to analyze.
- **input_data** – Input data sample to determine structure of inputs and outputs for given model object.
- **model_name** – The model name.
- **params** – dict with arbitrary parameters. Must be json-serializable
- **description** – text description of this model
- **additional_artifacts** – Additional artifact.
- **additional_requirements** – Additional requirements.
- **custom_wrapper** – Custom model wrapper.
- **custom_artifact** – Custom artifact collection to replace all other.
- **custom_requirements** – Custom requirements to replace all other.

Returns *Model*

delete(*force: bool = False*)
Deletes model from metadata and artifact repositories

Parameters **force** – whether model artifacts' deletion errors should be ignored, default is false

Returns Nothing

push(*task: ebonite.core.objects.core.Task = None*) → ebonite.core.objects.core.Model
Pushes *Model* instance into metadata and artifact repositories

Parameters **task** – *Task* instance to save model to. Optional if model already has

task :return: same saved *Model* instance

as_pipeline(*method_name=None*) → ebonite.core.objects.core.Pipeline
Create Pipeline that consists of this model's single method

Parameters **method_name** – name of the method. can be omitted if model has only one method

save()
Saves model to metadata repo and pushes unpersisted artifacts

has_children()
Checks if object has existing relationship

evaluate_set(*evalset: Union[str, ebonite.core.objects.core.EvaluationSet], evaluation_name: str = None, method_name: str = None, timestamp=None, save=True, force=False, raise_on_error=False*) → Optional[ebonite.core.objects.core.EvaluationResult]
Evaluates this model

Parameters

- **evalset** – evalset or it's name
- **evaluation_name** – name of this evaluation
- **method_name** – name of wrapper method. If none, all methods with consistent datatypes will be evaluated
- **timestamp** – time of the evaluation (defaults to now)
- **save** – save results to meta
- **force** – force reevalute
- **raise_on_error** – raise error if datatypes are incorrect or just return

evaluate(*input: ebonite.core.objects.dataset_source.DatasetSource, output: ebonite.core.objects.dataset_source.DatasetSource, metrics: Dict[str, ebonite.core.objects.metric.Metric], evaluation_name: str = None, method_name: str = None, timestamp=None, save=True, force=False, raise_on_error=False*) → Union[ebonite.core.objects.core.EvaluationResult, Dict[str, ebonite.core.objects.core.EvaluationResult], None]
Evaluates this model

Parameters

- **input** – input data
- **output** – target
- **metrics** – dict of metrics to evaluate
- **evaluation_name** – name of this evaluation
- **method_name** – name of wrapper method. If none, all methods with consistent datatypes will be evaluated
- **timestamp** – time of the evaluation (defaults to now)

- **save** – save results to meta
- **force** – force reevalute
- **raise_on_error** – raise error if datatypes are incorrect or just return

class ebonite.core.objects.**DatasetType**
Bases: *ebonite.core.objects.dataset_type.DatasetType*, pyjackson.decorators.SubtypeRegisterMixin

Base class for dataset type metadata. Children of this class must be both pyjackson-serializable and be a pyjackson serializer for it's dataset type

type = 'pyjackson.generics.DatasetType'

class ebonite.core.objects.**RuntimeEnvironment**(*name: str*, *id: int = None*, *params: ebonite.core.objects.core.RuntimeEnvironment.Params = None*, *author: str = None*, *creation_date: datetime.datetime = None*)
Bases: *ebonite.core.objects.core.EboniteObject*

Represents and environment where you can build and deploy your services Actual type of environment depends on *.params* field type

Parameters

- **name** – name of the environment
- **id** – id of the environment
- **author** – author of the enviroment
- **creation_date** – creation date of the enviroment
- **params** – *RuntimeEnvironment.Params* instance

class Params
Bases: ebonite.core.objects.core.Params, pyjackson.decorators.SubtypeRegisterMixin

Abstract class that represents different types of environments

type = 'pyjackson.decorators.Params'

delete(*meta_only: bool = False*, *cascade: bool = False*)
Deletes environment from metadata repository and(if required) stops associated instances

Parameters

- **meta_only** – wheter to only delete metadata
- **cascade** – Whether should environment be deleted with all associated instances

Returns Nothing

save()
Saves this env to metadata repository

has_children()
Checks if object has existing relationship

class ebonite.core.objects.**RuntimeInstance**(*name: Optional[str], id: int = None, image_id: int = None, environment_id: int = None, params: ebonite.core.objects.core.RuntimeInstance.Params = None, author: str = None, creation_date: datetime.datetime = None*)

Bases: ebonite.core.objects.core._WithRunner

Class that represents metadata for instance running in environment Actual type of instance depends on *.params* field type

Parameters

- **name** – name of the instance
- **id** – id of the instance
- **author** – author of the instance
- **image_id** – id of base image for htis instance
- **params** – *RuntimeInstance.Params* instance

Parma creation_date creation date of the instance

class Params

Bases: ebonite.core.objects.core.Params, pyjackson.decorators.SubtypeRegisterMixin

Abstract class that represents different types of images

type = 'pyjackson.decorators.Params'

image

delete(*meta_only: bool = False*)

Stops instance of model service and deletes it from repository

Parameters **meta_only** – only remove from metadata, do not stop instance

Returns nothing

run(***runner_kwargs*) → ebonite.core.objects.core.RuntimeInstance

Run this instance

Parameters **runner_kwargs** – additional params for runner.run (depends on runner implementation)

logs(***kwargs*)

Get logs of this instance

Parameters **kwargs** – parameters for runner *logs* method

Yields str logs from running instance

is_running(***kwargs*) → bool

Checks whether instance is running

Parameters **kwargs** – params for runner *is_running* method

Returns "is running" flag

stop(***kwargs*)

Stops the instance

Parameters **kwargs** – params for runner *stop* method

exists(**kwargs) → bool
Checks if instance exists (it may be stopped)

Parameters **kwargs** – params for runner *instance_exists* method

remove(**kwargs)
Removes the instance from environment (not from metadata)

Parameters **kwargs** – params for runner *remove_instance* method

save()
Saves object state to metadata repository

has_children()
Checks if object has existing relationship

class ebonite.core.objects.**ModelIO**
Bases: *ebonite.core.objects.wrapper.ModelIO*, pyjackson.decorators.SubtypeRegisterMixin

Helps model wrapper with IO

Must be pyjackson-serializable

type = 'pyjackson.decorators.ModelIO'

class ebonite.core.objects.**Pipeline**(*name: str, steps: List[ebonite.core.objects.core.PipelineStep], input_data: ebonite.core.objects.dataset_type.DatasetType, output_data: ebonite.core.objects.dataset_type.DatasetType, id: int = None, author: str = None, creation_date: datetime.datetime = None, task_id: int = None, evaluations: Dict[str, ebonite.core.objects.core.EvaluationResultCollection] = None*)
Bases: ebonite.core.objects.core._InTask

Pipeline is a class to represent a sequence of different Model's methods. They can be used to reuse different models (for example, pre-processing functions) in different pipelines. Pipelines must have exact same in and out data types as tasks they are in

Parameters

- **name** – name of the pipeline
- **steps** – sequence of :class:‘.PipelineStep‘s the pipeline consists of
- **input_data** – datatype of input dataset
- **output_data** – datatype of output datset
- **id** – id of the pipeline
- **author** – author of the pipeline
- **creation_date** – date of creation
- **task_id** – task.id of parent task

delete()
Deletes pipeline from metadata

load()

run(*data*)
Applies sequence of pipeline steps to data

Parameters **data** – data to apply pipeline to. must have type *Pipeline.input_data*

Returns processed data of type *Pipeline.output_data*

append(*model: Union[ebonite.core.objects.core.Model, ebonite.core.objects.core._WrapperMethodAccessor], method_name: str = None*)

Appends another Model to the sequence of this pipeline steps

Parameters

- **model** – either Model instance, or model method (as in *model.method* where *method* is method name)
- **method_name** – if Model was provided in *model*, this should be method name.

can be omitted if model have only one method

save()

Saves this pipeline to metadata repository

has_children()

Checks if object has existing relationship

evaluate_set(*evalset: Union[str, ebonite.core.objects.core.EvaluationSet], evaluation_name: str = None, timestamp=None, save=True, force=False, raise_on_error=False*) → Optional[ebonite.core.objects.core.EvaluationResult]

Evaluates this pipeline

Parameters

- **evalset** – evalset or it's name
- **evaluation_name** – name of this evaluation
- **timestamp** – time of the evaluation (defaults to now)
- **save** – save results to meta
- **force** – force reevalute
- **raise_on_error** – raise error if datatypes are incorrect or just return

evaluate(*input: ebonite.core.objects.dataset_source.DatasetSource, output: ebonite.core.objects.dataset_source.DatasetSource, metrics: Dict[str, ebonite.core.objects.metric.Metric], evaluation_name: str = None, timestamp=None, save=True, force=False, raise_on_error=False*) → Optional[ebonite.core.objects.core.EvaluationResult]

Evaluates this pipeline

Parameters

- **input** – input data
- **output** – target
- **metrics** – dict of metrics to evaluate
- **evaluation_name** – name of this evaluation
- **timestamp** – time of the evaluation (defaults to now)
- **save** – save results to meta
- **force** – force reevalute
- **raise_on_error** – raise error if datatypes are incorrect or just return

class ebonite.core.objects.**PipelineStep**(*model_name: str*, *method_name: str*)
Bases: ebonite.core.objects.base.EboniteParams

A class to represent one step of a Pipeline - a Model with one of its' methods name

Parameters

- **model_name** – name of the Model (in the same Task as Pipeline object)
- **method_name** – name of the method in Model's wrapper to use

Submodules

ebonite.core.objects.artifacts module

class ebonite.core.objects.artifacts.**Blob**
Bases: *ebonite.core.objects.artifacts.Blob*, pyjackson.decorators.SubtypeRegisterMixin

This class is a base class for blobs. Blob is a binary payload, which can be accessed either through `bytestream()` context manager, which returns file-like object, or through `materialize()` method, which places a file in local fs

Must be pyjackson-able or marked Unserializable

type = 'pyjackson.decorators.Blob'

class ebonite.core.objects.artifacts.**LocalFileBlob**(*path: str*)
Bases: *ebonite.core.objects.artifacts.Blob*

Blob implementation for local file

Parameters **path** – path to local file

type = 'local_file'

materialize(*path*)
Copies local file to another path

Parameters **path** – target path

bytestream() → Iterable[BinaryIO]
Opens file for reading

Returns file handler

class ebonite.core.objects.artifacts.**MaterializeOnlyBlobMixin**
Bases: *ebonite.core.objects.artifacts.Blob*

Mixin for blobs which always have to be materialized first

bytestream() → Iterable[BinaryIO]
Materializes blob to temporary dir and returns it's file handler

Returns file handler

type = 'ebonite.core.objects.artifacts.MaterializeOnlyBlobMixin'

class ebonite.core.objects.artifacts.**InMemoryBlob**(*payload: bytes*)
Bases: *ebonite.core.objects.artifacts.Blob*, pyjackson.core.Unserializable

Blob implementation for in-memory bytes

Parameters **payload** – bytes

type = 'inmemory'

materialize(*path*)
Writes payload to path

Parameters **path** – target path

bytestream() → Iterable[BinaryIO]
Creates BytesIO object from bytes

Yields file-like object

class ebonite.core.objects.artifacts.**LazyBlob**(*source: Callable[[], Union[str, bytes, IO]], encoding: str = 'utf8'*)
Bases: *ebonite.core.objects.artifacts.Blob*, pyjackson.core.Unserializable

Represents a lazy blob, which is computed only when needed

Parameters

- **source** – function with no arguments, that must return str, bytes or file-like object
- **encoding** – encoding for payload if source returns str of io.StringIO

materialize(*path*)
Writes payload to path

Parameters **path** – target path

bytestream() → Iterable[BinaryIO]
Creates BytesIO object from bytes

Yields file-like object

type = 'ebonite.core.objects.artifacts.LazyBlob'

class ebonite.core.objects.artifacts.**ArtifactCollection**
Bases: *ebonite.core.objects.artifacts.ArtifactCollection*, pyjackson.decorators.SubtypeRegisterMixin

Base class for artifact collection. Artifact collection is a number of named artifacts, represented by Blob's

Must be pyjackson-able

type = 'pyjackson.decorators.ArtifactCollection'

class ebonite.core.objects.artifacts.**Blobs**(*blobs: Dict[str, ebonite.core.objects.artifacts.Blob]*)
Bases: *ebonite.core.objects.artifacts.ArtifactCollection*

Artifact collection represented by a dictionary of blobs

Parameters **blobs** – dict of name -> blob

type = 'blobs'

materialize(*path*)
Materializes artifacts to path

Parameters **path** – target dir

bytes_dict() → Dict[str, bytes]
Implementation must return a dict of artifact name -> artifact payload

Returns dict of artifact names -> artifact payloads

blob_dict() → AbstractContextManager[Dict[str, ebonite.core.objects.artifacts.Blob]]

Yields self.blobs

class ebonite.core.objects.artifacts.**CompositeArtifactCollection**(*artifacts: List[ebonite.core.objects.artifacts.Arti*

Bases: *ebonite.core.objects.artifacts.ArtifactCollection*

Represents a merger of two or more ArtifactCollections

Parameters **artifacts** – ArtifactCollections to merge

type = 'composite'

materialize(*path*)
Materializes every ArtifactCollection to path

Parameters **path** – target dir

bytes_dict() → Dict[str, bytes]
Implementation must return a dict of artifact name -> artifact payload

Returns dict of artifact names -> artifact payloads

blob_dict() → AbstractContextManager[Dict[str, ebonite.core.objects.artifacts.Blob]]
Enters all ArtifactCollections blob_dict context managers and returns their union

Yields name -> blob mapping

ebonite.core.objects.core module

class ebonite.core.objects.core.**ExposedObjectMethod**(*name*, *param_name: str*, *param_type: str*, *param_doc: str = None*)

Bases: *ebonite.client.expose.ExposedMethod*

Decorator for EboniteObject methods that will be exposed to Ebonite class by autogen

Parameters

- **name** – name of the exposed method
- **param_name** – name of the first parameter
- **param_type** – type hint for the first parameter
- **param_doc** – docstring for the first parameter

get_doc()

generate_code()
Generates code for exposed Ebonite method

class ebonite.core.objects.core.**WithMetadataRepository**
Bases: object

Intermediat abstract class for objects that can be binded to meta repository

bind_meta_repo(*repo: ebonite.repository.metadata.base.MetadataRepository*)

unbind_meta_repo()

has_meta_repo

class ebonite.core.objects.core.**WithArtifactRepository**
Bases: object

Intermediat abstract class for objects that can be binded to artifact repository

bind_artifact_repo(*repo: ebonite.repository.artifact.base.ArtifactRepository*)

unbind_artifact_repo()

has_artifact_repo

class ebonite.core.objects.core.**WithDatasetRepository**
Bases: object

Intermediat abstract class for objects that can be binded to dataset repository

bind_dataset_repo(*repo: ebonite.repository.dataset.base.DatasetRepository*)

unbind_dataset_repo()

has_dataset_repo

class ebonite.core.objects.core.**EboniteObject**(*id: int*, *name: str*, *author: str = None*, *creation_date: datetime.datetime = None*)
Bases: pyjackson.core.Comparable, *ebonite.core.objects.core.WithMetadataRepository*, *ebonite.core.objects.core.WithArtifactRepository*

Base class for high level ebonite objects. These objects can be binded to metadata repository and/or to artifact repository

Parameters

- **id** – object id
- **name** – object name
- **author** – user that created that object
- **creation_date** – date when this object was created

id

bind_as(*other: ebonite.core.objects.core.EboniteObject*)

save()
Saves object state to metadata repository

has_children()
Checks if object has existing relationship

class ebonite.core.objects.core.**Project**(*name: str*, *id: int = None*, *author: str = None*, *creation_date: datetime.datetime = None*)
Bases: *ebonite.core.objects.core.EboniteObject*

Project is a collection of tasks

Parameters

- **id** – project id
- **name** – project name
- **author** – user that created that project
- **creation_date** – date when this project was created

delete(*cascade: bool = False*)
Deletes project and(if required) all tasks associated with it from metadata repository

Parameters **cascade** – whether should project be deleted with all associated tasks

Returns Nothing

add_task(*task: ebonite.core.objects.core.Task*)
Add task to project and save it to meta repo

Parameters **task** – task to add

add_tasks(*tasks: List[Task]*)
Add multiple tasks and save them to meta repo

Parameters **tasks** – tasks to add

delete_task(*task: ebonite.core.objects.core.Task*, *cascade: bool = False*)
Remove task from this project and delete it from meta repo

Parameters

- **cascade** – whether task should be deleted with all nested objects
- **task** – task to delete

save()
Saves object state to metadata repository

has_children()
Checks if object has existing relationship

class ebonite.core.objects.core.**EvaluationSet**(*input_dataset: str, output_dataset: str, metrics: List[str]*)
Bases: ebonite.core.objects.base.EboniteParams

Represents a set of objects for evaluation

Parameters

- **input_dataset** – name of the input dataset
- **output_dataset** – name of the output dataset
- **metrics** – list of metric names

get(*task: ebonite.core.objects.core.Task*, *cache=True*) → Tuple[ebonite.core.objects.dataset_source.DatasetSource, ebonite.core.objects.dataset_source.DatasetSource, Dict[str, ebonite.core.objects.metric.Metric]]
Loads actual datasets and metrics from task

Parameters

- **task** – task to load from
- **cache** – wheter to cache datasets

class ebonite.core.objects.core.**Task**(*name: str, id: int = None, project_id: int = None, author: str = None, creation_date: datetime.datetime = None, datasets: Dict[str, ebonite.core.objects.dataset_source.DatasetSource] = None, metrics: Dict[str, ebonite.core.objects.metric.Metric] = None, evaluation_sets: Dict[str, ebonite.core.objects.core.EvaluationSet] = None*)
Bases: *ebonite.core.objects.core.EboniteObject*, *ebonite.core.objects.core.WithDatasetRepository*

Task is a collection of models

Parameters

- **id** – task id

- **name** – task name
- **project_id** – parent project id for this task
- **author** – user that created that task
- **creation_date** – date when this task was created

project

delete(*cascade: bool = False*)
Deletes task from metadata

Parameters **cascade** – whether should task be deleted with all associated objects

Returns Nothing

add_model(*model: ebonite.core.objects.core.Model*)
Add model to task and save it to meta repo

Parameters **model** – model to add

add_models(*models: List[Model]*)
Add multiple models and save them to meta repo

Parameters **models** – models to add

delete_model(*model: ebonite.core.objects.core.Model*, *force=False*)
Remove model from this task and delete it from meta repo

Parameters

- **model** – model to delete
- **force** – whether model artifacts' deletion errors should be ignored, default is false

create_and_push_model(*model_object*, *input_data*, *model_name: str = None*, ***kwargs*) → ebonite.core.objects.core.Model
Create *Model* instance from model object and push it to repository

Parameters

- **model_object** – model object to build Model from
- **input_data** – input data sample to determine structure of inputs and outputs for given model
- **model_name** – name for model
- **kwargs** – other *create()* arguments

Returns created *Model*

push_model(*model: ebonite.core.objects.core.Model*) → ebonite.core.objects.core.Model
Push *Model* instance to task repository

Parameters **model** – *Model* to push

Returns same pushed *Model*

add_pipeline(*pipeline: ebonite.core.objects.core.Pipeline*)
Add model to task and save it to meta repo

Parameters **pipeline** – pipeline to add

add_pipelines(*pipelines: List[Pipeline]*)
Add multiple models and save them to meta repo

Parameters **pipelines** – pipelines to add

delete_pipeline(*pipeline: ebonite.core.objects.core.Pipeline*)
Remove model from this task and delete it from meta repo

Parameters **pipeline** – pipeline to delete

add_image(*image: ebonite.core.objects.core.Image*)
Add image for model and save it to meta repo

Parameters **image** – image to add

add_images(*images: List[Image]*)
Add multiple images for model and save them to meta repo

Parameters **images** – images to add

delete_image(*image: ebonite.core.objects.core.Image*, *meta_only: bool = False*, *cascade: bool = False*)
Remove image from this model and delete it from meta repo

Parameters

- **image** – image to delete
- **meta_only** – should image be deleted only from metadata
- **cascade** – whether image should be deleted with all instances

save()
Saves task to meta repository and pushes unsaved datasets to dataset repository

has_children()
Checks if object has existing relationship

add_evaluation(*name: str, data: Union[str, ebonite.core.objects.dataset_source.AbstractDataset, ebonite.core.objects.dataset_source.DatasetSource, Any], target: Union[str, ebonite.core.objects.dataset_source.AbstractDataset, ebonite.core.objects.dataset_source.DatasetSource, Any], metrics: Union[str, ebonite.core.objects.metric.Metric, Any, List[Union[str, ebonite.core.objects.metric.Metric, Any]]]*)
Adds new evaluation set to this task

Parameters

- **name** – name of the evaluation set
- **data** – input dataset for evaluation
- **target** – ground truth for input data
- **metrics** – one or more metrics to measure

delete_evaluation(*name: str*, *save: bool = True*)
Deletes evaluation set from task

Parameters

- **name** – name of the evaluation to delete
- **save** – also update task metadata in repo

add_dataset(*name, dataset: Union[ebonite.core.objects.dataset_source.DatasetSource, ebonite.core.objects.dataset_source.AbstractDataset, Any]*)
Adds new dataset to this task

Parameters

- **name** – name of the dataset

- **dataset** – Dataset, DatasetSource or raw dataset object

push_datasets()
Pushes all unsaved datasets to dataset repository

delete_dataset(*name: str*, *force: bool = False*, *save: bool = True*)
Deletes dataset from task with artifacts

Parameters

- **name** – name of the dataset to delete
- **force** – wheter to check evalsets that use this dataset and remove them or raise error
- **save** – also update task metadata in repo

add_metric(*name, metric: Union[ebonite.core.objects.metric.Metric, Any]*)
Adds metric to this task

Parameters

- **name** – name of the metric
- **metric** – Metric or raw metric object

delete_metric(*name: str*, *force: bool = False*, *save: bool = True*)
Deletes metric from task

Parameters

- **name** – name of the metric to delete
- **force** – wheter to check evalsets that use this metric and remove them or raise error
- **save** – also update task metadata in repo

evaluate_all(*force=False*, *save_result=True*) → Dict[str, ebonite.core.objects.core.EvaluationResult]
Evaluates all viable pairs of evalsets and models/pipelines

Parameters

- **force** – force reevaluate already evaluated
- **save_result** – save evaluation results to meta

class ebonite.core.objects.core.**EvaluationResult**(*timestamp: float*, *scores: Dict[str, float] = None*)
Bases: ebonite.core.objects.base.EboniteParams

Represents result of evaluation of one evalset on multiple evaluatable objects

Parameters

- **scores** – mapping 'metric' -> 'score'
- **timestamp** – time of evaluation

class ebonite.core.objects.core.**EvaluationResultCollection**(*results: List[ebonite.core.objects.core.EvaluationResul = None*)
Bases: ebonite.core.objects.base.EboniteParams

Collection of evaluation results for single evalset

Parameters **results** – list of results

add(*result: ebonite.core.objects.core.EvaluationResult*)

latest

class ebonite.core.objects.core.**Model**(*name: str*, *wrapper_meta: Optional[dict] = None*, *artifact: Optional[ebonite.core.objects.artifacts.ArtifactCollection] = None*, *requirements: ebonite.core.objects.requirements.Requirements = None*, *params: Dict[str, Any] = None*, *description: str = None*, *id: int = None*, *task_id: int = None*, *author: str = None*, *creation_date: datetime.datetime = None*, *evaluations: Dict[str, Dict[str, ebonite.core.objects.core.EvaluationResultCollection]] = None*)

Bases: ebonite.core.objects.core._InTask

Model contains metadata for machine learning model

Parameters

- **name** – model name
- **wrapper_meta** – *ModelWrapper* instance for this model
- **artifact** – *ArtifactCollection* instance with model artifacts
- **requirements** – *Requirements* instance with model requirements
- **params** – dict with arbitrary parameters. Must be json-serializable
- **description** – text description of this model
- **id** – model id
- **task_id** – parent task_id
- **author** – user that created that model
- **creation_date** – date when this model was created

PYTHON_VERSION = 'python_version'

load()
Load model artifacts into wrapper

ensure_loaded()
Ensure that wrapper has loaded model object

wrapper

with_wrapper(*wrapper: ebonite.core.objects.wrapper.ModelWrapper*)
Bind wrapper instance to this Model

Parameters **wrapper** – *ModelWrapper* instance

Returns self

wrapper_meta
pyjackson representation of *ModelWrapper* for this model: e.g., this provides possibility to move a model between repositories without its dependencies being installed

Type return

with_wrapper_meta(*wrapper_meta: dict*)
Bind wrapper_meta dict to this Model

Parameters **wrapper_meta** – dict with serialized *ModelWrapper* instance

Returns self

artifact
persisted artifacts if any

Type return

artifact_any
artifacts in any state (persisted or not)

Type return

artifact_req_persisted
Similar to *artifact* but checks that no unpersisted artifacts are left

Returns persisted artifacts if any

attach_artifact(*artifact: ebonite.core.objects.artifacts.ArtifactCollection*)

Parameters **artifact** – artifacts to attach to model in an unpersisted state

persist_artifacts(*persister: Callable[[ArtifactCollection], ArtifactCollection]*)
Model artifacts persisting workflow

Parameters **persister** – external object which stores model artifacts

without_artifacts() → ebonite.core.objects.core.Model

Returns copy of the model with no artifacts attached

classmethod create(*model_object, input_data, model_name: str = None, params: Dict[str, Any] = None, description: str = None, additional_artifacts: ebonite.core.objects.artifacts.ArtifactCollection = None, additional_requirements: Union[ebonite.core.objects.requirements.Requirements, ebonite.core.objects.requirements.Requirement, List[ebonite.core.objects.requirements.Requirement], str, List[str]] = None, custom_wrapper: ebonite.core.objects.wrapper.ModelWrapper = None, custom_artifact: ebonite.core.objects.artifacts.ArtifactCollection = None, custom_requirements: Union[ebonite.core.objects.requirements.Requirements, ebonite.core.objects.requirements.Requirement, List[ebonite.core.objects.requirements.Requirement], str, List[str]] = None*) → ebonite.core.objects.core.Model
Creates Model instance from arbitrary model objects and sample of input data

Parameters

- **model_object** – The model object to analyze.
- **input_data** – Input data sample to determine structure of inputs and outputs for given model object.
- **model_name** – The model name.
- **params** – dict with arbitrary parameters. Must be json-serializable
- **description** – text description of this model
- **additional_artifacts** – Additional artifact.
- **additional_requirements** – Additional requirements.
- **custom_wrapper** – Custom model wrapper.
- **custom_artifact** – Custom artifact collection to replace all other.
- **custom_requirements** – Custom requirements to replace all other.

Returns *Model*

delete(*force: bool = False*)
Deletes model from metadata and artifact repositories

Parameters **force** – whether model artifacts' deletion errors should be ignored, default is false

Returns Nothing

push(*task: ebonite.core.objects.core.Task = None*) → ebonite.core.objects.core.Model
Pushes *Model* instance into metadata and artifact repositories

Parameters **task** – *Task* instance to save model to. Optional if model already has

task :return: same saved *Model* instance

as_pipeline(*method_name=None*) → ebonite.core.objects.core.Pipeline
Create Pipeline that consists of this model's single method

Parameters **method_name** – name of the method. can be omitted if model has only one method

save()
Saves model to metadata repo and pushes unpersisted artifacts

has_children()
Checks if object has existing relationship

evaluate_set(*evalset: Union[str, ebonite.core.objects.core.EvaluationSet], evaluation_name: str = None, method_name: str = None, timestamp=None, save=True, force=False, raise_on_error=False*) → Optional[ebonite.core.objects.core.EvaluationResult]
Evaluates this model

Parameters

- **evalset** – evalset or it's name
- **evaluation_name** – name of this evaluation
- **method_name** – name of wrapper method. If none, all methods with consistent datatypes will be evaluated
- **timestamp** – time of the evaluation (defaults to now)
- **save** – save results to meta
- **force** – force reevalute
- **raise_on_error** – raise error if datatypes are incorrect or just return

evaluate(*input: ebonite.core.objects.dataset_source.DatasetSource, output: ebonite.core.objects.dataset_source.DatasetSource, metrics: Dict[str, ebonite.core.objects.metric.Metric], evaluation_name: str = None, method_name: str = None, timestamp=None, save=True, force=False, raise_on_error=False*) → Union[ebonite.core.objects.core.EvaluationResult, Dict[str, ebonite.core.objects.core.EvaluationResult], None]
Evaluates this model

Parameters

- **input** – input data
- **output** – target
- **metrics** – dict of metrics to evaluate
- **evaluation_name** – name of this evaluation

- **method_name** – name of wrapper method. If none, all methods with consistent datatypes will be evaluated
- **timestamp** – time of the evaluation (defaults to now)
- **save** – save results to meta
- **force** – force reevalute
- **raise_on_error** – raise error if datatypes are incorrect or just return

class ebonite.core.objects.core.**PipelineStep**(*model_name: str*, *method_name: str*)

Bases: ebonite.core.objects.base.EboniteParams

A class to represent one step of a Pipeline - a Model with one of its' methods name

Parameters

- **model_name** – name of the Model (in the same Task as Pipeline object)
- **method_name** – name of the method in Model's wrapper to use

class ebonite.core.objects.core.**Pipeline**(*name: str, steps: List[ebonite.core.objects.core.PipelineStep], input_data: ebonite.core.objects.dataset_type.DatasetType, output_data: ebonite.core.objects.dataset_type.DatasetType, id: int = None, author: str = None, creation_date: datetime.datetime = None, task_id: int = None, evaluations: Dict[str, ebonite.core.objects.core.EvaluationResultCollection] = None*)

Bases: ebonite.core.objects.core._InTask

Pipeline is a class to represent a sequence of different Model's methods. They can be used to reuse different models (for example, pre-processing functions) in different pipelines. Pipelines must have exact same in and out data types as tasks they are in

Parameters

- **name** – name of the pipeline
- **steps** – sequence of :class:‘.PipelineStep‘s the pipeline consists of
- **input_data** – datatype of input dataset
- **output_data** – datatype of output datset
- **id** – id of the pipeline
- **author** – author of the pipeline
- **creation_date** – date of creation
- **task_id** – task.id of parent task

delete()

Deletes pipeline from metadata

load()

run(*data*)

Applies sequence of pipeline steps to data

Parameters **data** – data to apply pipeline to. must have type *Pipeline.input_data*

Returns processed data of type *Pipeline.output_data*

append(*model: Union[ebonite.core.objects.core.Model, ebonite.core.objects.core._WrapperMethodAccessor], method_name: str = None*)
Appends another Model to the sequence of this pipeline steps

Parameters

- **model** – either Model instance, or model method (as in *model.method* where *method* is method name)
- **method_name** – if Model was provided in *model*, this should be method name.

can be omitted if model have only one method

save()
Saves this pipeline to metadata repository

has_children()
Checks if object has existing relationship

evaluate_set(*evalset: Union[str, ebonite.core.objects.core.EvaluationSet], evaluation_name: str = None, timestamp=None, save=True, force=False, raise_on_error=False*) → Optional[ebonite.core.objects.core.EvaluationResult]
Evaluates this pipeline

Parameters

- **evalset** – evalset or it's name
- **evaluation_name** – name of this evaluation
- **timestamp** – time of the evaluation (defaults to now)
- **save** – save results to meta
- **force** – force reevalute
- **raise_on_error** – raise error if datatypes are incorrect or just return

evaluate(*input: ebonite.core.objects.dataset_source.DatasetSource, output: ebonite.core.objects.dataset_source.DatasetSource, metrics: Dict[str, ebonite.core.objects.metric.Metric], evaluation_name: str = None, timestamp=None, save=True, force=False, raise_on_error=False*) → Optional[ebonite.core.objects.core.EvaluationResult]
Evaluates this pipeline

Parameters

- **input** – input data
- **output** – target
- **metrics** – dict of metrics to evaluate
- **evaluation_name** – name of this evaluation
- **timestamp** – time of the evaluation (defaults to now)
- **save** – save results to meta
- **force** – force reevalute
- **raise_on_error** – raise error if datatypes are incorrect or just return

class ebonite.core.objects.core.**Buildable**
Bases: *ebonite.core.objects.core.Buildable*, pyjackson.decorators.SubtypeRegisterMixin

An abstract class that represents something that can be built by Builders Have default implementations for Models and Pipelines (and lists of them)

type = 'pyjackson.decorators.Buildable'

class ebonite.core.objects.core.**RuntimeEnvironment**(*name: str*, *id: int = None*, *params: ebonite.core.objects.core.RuntimeEnvironment.Params = None*, *author: str = None*, *creation_date: datetime.datetime = None*)

Bases: *ebonite.core.objects.core.EboniteObject*

Represents and environment where you can build and deploy your services Actual type of environment depends on *.params* field type

Parameters

- **name** – name of the environment
- **id** – id of the environment
- **author** – author of the enviroment
- **creation_date** – creation date of the enviroment
- **params** – *RuntimeEnvironment.Params* instance

class Params

Bases: ebonite.core.objects.core.Params, pyjackson.decorators.SubtypeRegisterMixin

Abstract class that represents different types of environments

type = 'pyjackson.decorators.Params'

delete(*meta_only: bool = False*, *cascade: bool = False*)

Deletes environment from metadata repository and(if required) stops associated instances

Parameters

- **meta_only** – wheter to only delete metadata
- **cascade** – Whether should environment be deleted with all associated instances

Returns Nothing

save()

Saves this env to metadata repository

has_children()

Checks if object has existing relationship

class ebonite.core.objects.core.**Image**(*name: Optional[str]*, *source: ebonite.core.objects.core.Buildable*, *id: int = None*, *params: ebonite.core.objects.core.Image.Params = None*, *author: str = None*, *creation_date: datetime.datetime = None*, *task_id: int = None*, *environment_id: int = None*)

Bases: ebonite.core.objects.core._WithBuilder

Class that represents metadata for image built from Buildable Actual type of image depends on *.params* field type

Parameters

- **name** – name of the image

- **id** – id of the image
- **author** – author of the image
- **source** – *Buildable* instance this image was built from
- **params** – *Image.Params* instance
- **task_id** – task.id this image belongs to
- **environment_id** – environment.id this image belongs to

Parma creation_date creation date of the image

class Params

Bases: ebonite.core.objects.core.Params, pyjackson.decorators.SubtypeRegisterMixin

Abstract class that represents different types of images

type = 'pyjackson.decorators.Params'

task

delete(*meta_only: bool = False*, *cascade: bool = False*)

Deletes existing image from metadata repository and image provider

Parameters

- **meta_only** – should image be deleted only from metadata
- **cascade** – whether to delete nested RuntimeInstances

bind_meta_repo(*repo: ebonite.repository.metadata.base.MetadataRepository*)

is_built() → bool

Checks if image was built and wasn't removed

build(***kwargs*)

Build this image

Parameters **kwargs** – additional params for builder.build_image (depends on builder implementation)

remove(***kwargs*)

remove this image (from environment, not from ebonite metadata)

save()

Saves object state to metadata repository

has_children()

Checks if object has existing relationship

class ebonite.core.objects.core.**RuntimeInstance**(*name: Optional[str], id: int = None, image_id: int = None, environment_id: int = None, params: ebonite.core.objects.core.RuntimeInstance.Params = None, author: str = None, creation_date: datetime.datetime = None*)

Bases: ebonite.core.objects.core._WithRunner

Class that represents metadata for instance running in environment Actual type of instance depends on *.params* field type

Parameters

- **name** – name of the instance
- **id** – id of the instance
- **author** – author of the instance
- **image_id** – id of base image for htis instance
- **params** – *RuntimeInstance.Params* instance

Parma creation_date creation date of the instance

class Params

Bases: ebonite.core.objects.core.Params, pyjackson.decorators.SubtypeRegisterMixin

Abstract class that represents different types of images

type = 'pyjackson.decorators.Params'

image

delete(*meta_only: bool = False*)

Stops instance of model service and deletes it from repository

Parameters **meta_only** – only remove from metadata, do not stop instance

Returns nothing

run(***runner_kwargs*) → ebonite.core.objects.core.RuntimeInstance

Run this instance

Parameters **runner_kwargs** – additional params for runner.run (depends on runner implementation)

logs(***kwargs*)

Get logs of this instance

Parameters **kwargs** – parameters for runner *logs* method

Yields str logs from running instance

is_running(***kwargs*) → bool

Checks whether instance is running

Parameters **kwargs** – params for runner *is_running* method

Returns "is running" flag

stop(***kwargs*)

Stops the instance

Parameters **kwargs** – params for runner *stop* method

exists(***kwargs*) → bool

Checks if instance exists (it may be stopped)

Parameters **kwargs** – params for runner *instance_exists* method

remove(***kwargs*)

Removes the instance from environment (not from metadata)

Parameters **kwargs** – params for runner *remove_instance* method

save()

Saves object state to metadata repository

has_children()
Checks if object has existing relationship

ebonite.core.objects.dataset_source module

class ebonite.core.objects.dataset_source.**AbstractDataset**(*dataset_type: ebonite.core.objects.dataset_type.DatasetType*)
Bases: pyjackson.core.Unserializable

ABC for Dataset objects

Parameters **dataset_type** – DatasetType instance for the data in the Dataset

iterate() → collections.abc.Iterable
Abstract method to iterate through data

get()
Abstract method to get data object

get_writer()
Returns writer for this dataset. Defaults to dataset_type.get_writer()

get_reader()
Returns reader for this dataset. Defaults to dataset_type.get_reader()

class ebonite.core.objects.dataset_source.**Dataset**(*data: Any, dataset_type: ebonite.core.objects.dataset_type.DatasetType*)
Bases: *ebonite.core.objects.dataset_source.AbstractDataset*

Wrapper for dataset objects

Parameters

- **data** – raw dataset
- **dataset_type** – DatasetType of the raw data

iterate() → collections.abc.Iterable
Abstract method to iterate through data

get()
Abstract method to get data object

classmethod from_object(*data*)
Creates Dataset instance from raw data object

to_inmemory_source() → ebonite.core.objects.dataset_source.InMemoryDatasetSource
Returns *InMemoryDatasetSource* with this dataset

class ebonite.core.objects.dataset_source.**DatasetSource**(*dataset_type: ebonite.core.objects.dataset_type.DatasetType*)
Bases: *ebonite.core.objects.dataset_source.DatasetSource*, pyjackson.decorators.SubtypeRegisterMixin

Class that represents a source that can produce a Dataset

Parameters **dataset_type** – DatasetType of contained dataset

type = 'pyjackson.decorators.DatasetSource'

class ebonite.core.objects.dataset_source.**CachedDatasetSource**(*source: ebonite.core.objects.dataset_source.Datas*
Bases: *ebonite.core.objects.dataset_source.DatasetSource*

Wrapper that will cache the result of underlying source on the first read

Parameters **source** – underlying DatasetSource

read() → ebonite.core.objects.dataset_source.Dataset
Abstract method that must return produced Dataset instance

cache()
Returns *CachedDatasetSource* that will cache data on the first read

type = 'ebonite.core.objects.dataset_source.CachedDatasetSource'

class ebonite.core.objects.dataset_source.**InMemoryDatasetSource**(*dataset: ebonite.core.objects.dataset_source.Da*

Bases: *ebonite.core.objects.dataset_source.CachedDatasetSource*, pyjackson.core.Unserializable

DatasetSource that holds existing dataset inmemory

Parameters **dataset** – Dataset instance to hold

type = 'ebonite.core.objects.dataset_source.InMemoryDatasetSource'

ebonite.core.objects.dataset_type module

class ebonite.core.objects.dataset_type.**DatasetType**
Bases: *ebonite.core.objects.dataset_type.DatasetType*, pyjackson.decorators.SubtypeRegisterMixin

Base class for dataset type metadata. Children of this class must be both pyjackson-serializable and be a pyjackson serializer for it's dataset type

type = 'pyjackson.generics.DatasetType'

class ebonite.core.objects.dataset_type.**LibDatasetTypeMixin**
Bases: *ebonite.core.objects.dataset_type.DatasetType*

DatasetType mixin which provides requirements list consisting of PIP packages represented by module objects in *libraries* field.

libraries = None

requirements

type = 'ebonite.core.objects.dataset_type.LibDatasetTypeMixin'

class ebonite.core.objects.dataset_type.**PrimitiveDatasetType**(*ptype: str*)
Bases: *ebonite.core.objects.dataset_type.DatasetType*

DatasetType for int, str, bool, complex and float types

type = 'primitive'

classmethod from_object(*obj*)

to_type

get_spec() → List[pyjackson.core.Field]

deserialize(*obj*)

serialize(*instance*)

requirements

get_writer()

class ebonite.core.objects.dataset_type.**ListDatasetType**(*dtype: ebonite.core.objects.dataset_type.DatasetType*, *size: int*)

Bases: *ebonite.core.objects.dataset_type.DatasetType*, *ebonite.core.objects.typing.SizedTypedListType*

DatasetType for list type

real_type = None

type = 'list'

deserialize(*obj*)

serialize(*instance: list*)

requirements

get_writer()

class ebonite.core.objects.dataset_type.**TupleLikeListDatasetType**(*items: List[ebonite.core.objects.dataset_type*

Bases: ebonite.core.objects.dataset_type._TupleLikeDatasetType

DatasetType for tuple-like list type

actual_type
alias of builtins.list

type = 'tuple_like_list'

class ebonite.core.objects.dataset_type.**TupleDatasetType**(*items: List[ebonite.core.objects.dataset_type.DatasetTyp*

Bases: ebonite.core.objects.dataset_type._TupleLikeDatasetType

DatasetType for tuple type

actual_type
alias of builtins.tuple

type = 'tuple'

class ebonite.core.objects.dataset_type.**DictDatasetType**(*item_types: Dict[str, ebonite.core.objects.dataset_type.DatasetType]*)

Bases: *ebonite.core.objects.dataset_type.DatasetType*

DatasetType for dict type

real_type = None

type = 'dict'

get_spec() → List[pyjackson.core.Field]

deserialize(*obj*)

serialize(*instance: dict*)

requirements

get_writer()

class ebonite.core.objects.dataset_type.**BytesDatasetType**

Bases: *ebonite.core.objects.dataset_type.DatasetType*

DatasetType for bytes objects

type = 'bytes'

real_type = None

get_spec() → List[pyjackson.core.Field]

deserialize(*obj*) → object

serialize(*instance: object*) → dict

requirements

get_writer()

ebonite.core.objects.metric module

class ebonite.core.objects.metric.**Metric**

Bases: *ebonite.core.objects.metric.Metric*, pyjackson.decorators.SubtypeRegisterMixin

type = 'pyjackson.decorators.Metric'

class ebonite.core.objects.metric.**LibFunctionMetric**(*function: str*, *args: Dict[str, Any] = None*, *invert_input: bool = False*)

Bases: *ebonite.core.objects.metric.Metric*

evaluate(*truth*, *prediction*)

type = 'ebonite.core.objects.metric.LibFunctionMetric'

class ebonite.core.objects.metric.**CallableMetricWrapper**(*artifacts: Dict[str, str]*, *requirements: ebonite.core.objects.requirements.Requirements*)

Bases: object

bind(*callable*)

static compress(*s: bytes*) → str

Helper method to compress source code

Parameters **s** – source code

Returns base64 encoded string of zipped source

static decompress(*s: str*) → bytes

Helper method to decompress source code

Parameters **s** – compressed source code

Returns decompressed source code

classmethod from_callable(*callable*)

load()

class ebonite.core.objects.metric.**CallableMetric**(*wrapper: ebonite.core.objects.metric.CallableMetricWrapper*)

Bases: *ebonite.core.objects.metric.Metric*

evaluate(*truth*, *prediction*)

type = 'ebonite.core.objects.metric.CallableMetric'

ebonite.core.objects.requirements module

ebonite.core.objects.requirements.**read**(*path*, *bin=False*)

class ebonite.core.objects.requirements.**Requirement**
Bases: *ebonite.core.objects.requirements.Requirement*, pyjackson.decorators.SubtypeRegisterMixin

Base class for python requirement

type = 'pyjackson.decorators.Requirement'

class ebonite.core.objects.requirements.**PythonRequirement**
Bases: *ebonite.core.objects.requirements.Requirement*

module = None

type = 'ebonite.core.objects.requirements.PythonRequirement'

class ebonite.core.objects.requirements.**InstallableRequirement**(*module: str*, *version: str = None*, *package_name: str = None*)
Bases: *ebonite.core.objects.requirements.PythonRequirement*

This class represents pip-installable python library

Parameters

- **module** – name of python module
- **version** – version of python package
- **package_name** – Optional. pip package name for this module, if it is different from module name

type = 'installable'

package
Pip package name

to_str()
pip installable representation of this module

classmethod from_module(*mod: module*, *package_name: str = None*) → ebonite.core.objects.requirements.InstallableRequirement
Factory method to create *InstallableRequirement* from module object

Parameters

- **mod** – module object
- **package_name** – PIP package name if it is not equal to module name

Returns *InstallableRequirement*

classmethod from_str(*name*)
Factory method for creating *InstallableRequirement* from string

Parameters **name** – string representation

Returns *InstallableRequirement*

class ebonite.core.objects.requirements.**CustomRequirement**(*name: str*, *source64zip: str*, *is_package: bool*)

Bases: *ebonite.core.objects.requirements.PythonRequirement*

This class represents local python code that you need as a requirement for your code

Parameters

- **name** – filename of this code
- **source64zip** – zipped and base64-encoded source
- **is_package** – whether this code should be in %name%/__init__.py

type = 'custom'

static from_module(*mod: module*) → ebonite.core.objects.requirements.CustomRequirement

Factory method to create *CustomRequirement* from module object

Parameters **mod** – module object

Returns *CustomRequirement*

static compress(*s: str*) → str

Helper method to compress source code

Parameters **s** – source code

Returns base64 encoded string of zipped source

static compress_package(*s: Dict[str, bytes]*) → str

static decompress(*s: str*) → str

Helper method to decompress source code

Parameters **s** – compressed source code

Returns decompressed source code

static decompress_package(*s: str*) → Dict[str, bytes]

module

Module name for this requirement

source

Source code of this requirement

sources

to_sources_dict()

Mapping path -> source code for this requirement

Returns dict path -> source

class ebonite.core.objects.requirements.**FileRequirement**(*name: str*, *source64zip: str*)

Bases: *ebonite.core.objects.requirements.CustomRequirement*

to_sources_dict()

Mapping path -> source code for this requirement

Returns dict path -> source

classmethod from_path(*path: str*)

type = 'ebonite.core.objects.requirements.FileRequirement'

class ebonite.core.objects.requirements.**UnixPackageRequirement**(*package_name: str*)

Bases: *ebonite.core.objects.requirements.Requirement*

type = 'ebonite.core.objects.requirements.UnixPackageRequirement'

class ebonite.core.objects.requirements.**Requirements**(*requirements: List[ebonite.core.objects.requirements.Requirement] = None*)

Bases: ebonite.core.objects.base.EboniteParams

A collection of requirements

Parameters **requirements** – list of *Requirement* instances

installable
List of installable requirements

custom
List of custom requirements

of_type(*type_: Type[T]*) → List[T]

Parameters **type** – type of requirements

Returns List of requirements of type *type_*

modules
List of module names

add(*requirement: ebonite.core.objects.requirements.Requirement*)
Adds requirement to this collection

Parameters **requirement** – *Requirement* instance to add

to_pip() → List[str]

Returns list of pip installable packages

ebonite.core.objects.requirements.**resolve_requirements**(*other: Union[ebonite.core.objects.requirements.Requirement, ebonite.core.objects.requirements.Requirement, List[ebonite.core.objects.requirements.Requirement], str, List[str]]*) → ebonite.core.objects.requirements.Requirements

Helper method to create *Requirements* from any supported source. Supported formats: *Requirements*, *Requirement*, list of *Requirement*, string representation or list of string representations

Parameters **other** – requirement in supported format

Returns *Requirements* instance

ebonite.core.objects.typing module

class ebonite.core.objects.typing.**TypeWithSpec**
Bases: pyjackson.generics.Serializer

Abstract base class for types providing its OpenAPI schema definition

get_spec() → List[pyjackson.core.Field]

is_list()

list_size()

class ebonite.core.objects.typing.**ListTypeWithSpec**
Bases: *ebonite.core.objects.typing.TypeWithSpec*

Abstract base class for *list*-like types providing its OpenAPI schema definition

is_list()

list_size()

class ebonite.core.objects.typing.**SizedTypedListType**(*size: Optional[int]*, *dtype: type*)
Bases: *ebonite.core.objects.typing.ListTypeWithSpec*

Subclass of *ListTypeWithSpec* which specifies size of internal *list*

get_spec() → List[pyjackson.core.Field]

list_size()

deserialize(*obj*)

serialize(*instance*)

ebonite.core.objects.wrapper module

class ebonite.core.objects.wrapper.**ModelIO**
Bases: *ebonite.core.objects.wrapper.ModelIO*, pyjackson.decorators.SubtypeRegisterMixin

Helps model wrapper with IO

Must be pyjackson-serializable

type = 'pyjackson.decorators.ModelIO'

class ebonite.core.objects.wrapper.**ModelWrapper**(*io: ebonite.core.objects.wrapper.ModelIO*)
Bases: *ebonite.core.objects.wrapper.ModelWrapper*, pyjackson.decorators.SubtypeRegisterMixin

Base class for model wrapper. Wrapper is an object that can save, load and inference a model

Must be pyjackson-serializable

type = 'pyjackson.decorators.ModelWrapper'

class ebonite.core.objects.wrapper.**LibModelWrapperMixin**(*io: ebonite.core.objects.wrapper.ModelIO*)
Bases: *ebonite.core.objects.wrapper.ModelWrapper*

ModelWrapper mixin which provides model object requirements list consisting of PIP packages represented by module objects in *libraries* field.

libraries = None

type = 'ebonite.core.objects.wrapper.LibModelWrapperMixin'

class ebonite.core.objects.wrapper.**WrapperArtifactCollection**(*wrapper: ebonite.core.objects.wrapper.ModelWrapper*)
Bases: *ebonite.core.objects.artifacts.ArtifactCollection*, pyjackson.core.Unserializable

This is a proxy ArtifactCollection for not persisted artifacts. Internally uses `dump()` to create model artifacts

Parameters **wrapper** – *ModelWrapper* instance

type = '_wrapper'

materialize(*path*)
Calls dump() to materialize model in path

Parameters **path** – path to materialize model

bytes_dict() → Dict[str, bytes]
Calls dump() to get model artifacts bytes dict :return: dict artifact name -> bytes

blob_dict() → Iterable[Dict[str, ebonite.core.objects.artifacts.Blob]]
Calls dump() to get model artifacts blob dict

Returns dict artifact name -> *Blob*

class ebonite.core.objects.wrapper.**PickleModelIO**
Bases: *ebonite.core.objects.wrapper.ModelIO*

ModelIO for pickle-able models

When model is dumped, recursively checks objects if they can be dumped with ModelIO instead of pickling

So, if you use function that internally calls tensorflow model, this tensorflow model will be dumped with tensorflow code and not pickled

model_filename = 'model.pkl'

io_ext = '.io'

dump(*model*) → ebonite.core.objects.artifacts.ArtifactCollection
Dumps model artifacts as *ArtifactCollection*

Returns context manager with *ArtifactCollection*

load(*path*)
Loads artifacts into model field

Parameters **path** – path to load from

type = 'ebonite.core.objects.wrapper.PickleModelIO'

class ebonite.core.objects.wrapper.**CallableMethodModelWrapper**
Bases: *ebonite.core.objects.wrapper.ModelWrapper*

ModelWrapper implementation for functions

type = 'callable_method'

Submodules

ebonite.core.errors module

exception ebonite.core.errors.**EboniteError**
Bases: Exception

General Ebonite error

exception ebonite.core.errors.**MetadataError**
Bases: *ebonite.core.errors.EboniteError*

General Ebonite Metadata Error

exception ebonite.core.errors.**ExistingProjectError**(*project: Union[ebonite.core.objects.core.Project, int, str]*)
Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**NonExistingProjectError**(*project: Union[ebonite.core.objects.core.Project, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ExistingTaskError**(*task: Union[ebonite.core.objects.core.Task, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**NonExistingTaskError**(*task: Union[ebonite.core.objects.core.Task, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**TaskWithoutIdError**(*task: Union[ebonite.core.objects.core.Task, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ExistingModelError**(*model: Union[ebonite.core.objects.core.Model, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**NonExistingModelError**(*model: Union[ebonite.core.objects.core.Model, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ExistingPipelineError**(*pipeline: Union[ebonite.core.objects.core.Pipeline, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**NonExistingPipelineError**(*pipeline: Union[ebonite.core.objects.core.Pipeline, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ExistingImageError**(*image: Union[ebonite.core.objects.core.Image, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**NonExistingImageError**(*image: Union[ebonite.core.objects.core.Image, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ExistingEnvironmentError**(*environment: Union[ebonite.core.objects.core.RuntimeEnvironmen int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**NonExistingEnvironmentError**(*environment: Union[ebonite.core.objects.core.RuntimeEnviron int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ExistingInstanceError**(*instance: Union[ebonite.core.objects.core.RuntimeInstance, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**NonExistingInstanceError**(*instance: Union[ebonite.core.objects.core.RuntimeInstance, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**TaskNotInProjectError**(*task: ebonite.core.objects.core.Task*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ModelNotInTaskError**(*model: ebonite.core.objects.core.Model*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**PipelineNotInTaskError**(*pipeline: ebonite.core.objects.core.Pipeline*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ImageNotInTaskError**(*image: ebonite.core.objects.core.Image*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**InstanceNotInImageError**(*instance: ebonite.core.objects.core.RuntimeInstance*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**InstanceNotInEnvironmentError**(*instance: ebonite.core.objects.core.RuntimeInstance*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ModelWithoutIdError**(*model: Union[ebonite.core.objects.core.Model, int, str]*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**UnboundObjectError**

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ProjectWithTasksError**(*project: ebonite.core.objects.core.Project*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**TaskWithFKError**(*task: ebonite.core.objects.core.Task*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**ImageWithInstancesError**(*image: ebonite.core.objects.core.Image*)

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**EnvironmentWithInstancesError**(*environment: ebonite.core.objects.core.RuntimeEnvironmen*

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**UnknownMetadataError**

Bases: *ebonite.core.errors.MetadataError*

exception ebonite.core.errors.**DatasetError**

Bases: *ebonite.core.errors.EboniteError*

Base class for exceptions in `DatasetRpository`

exception ebonite.core.errors.**NoSuchDataset**(*dataset_id*, *repo*, *e=None*)

Bases: *ebonite.core.errors.DatasetError*

exception ebonite.core.errors.**DatasetExistsError**(*dataset_id*, *repo*, *e=None*)

Bases: *ebonite.core.errors.DatasetError*

exception ebonite.core.errors.**ArtifactError**
Bases: *ebonite.core.errors.EboniteError*

Base class for exceptions in ArtifactRepository

exception ebonite.core.errors.**NoSuchArtifactError**(*artifact_id*, *repo*)
Bases: *ebonite.core.errors.ArtifactError*

Exception which is thrown if artifact is not found in the repository

exception ebonite.core.errors.**ArtifactExistsError**(*artifact_id*, *repo*)
Bases: *ebonite.core.errors.ArtifactError*

Exception which is thrown if artifact already exists in the repository

4.1.4 ebonite.ext package

class ebonite.ext.**ExtensionLoader**
Bases: object

Class that tracks and loads extensions.

builtin_extensions = {'ebonite.ext.aiohttp': <Extension ebonite.ext.aiohttp>, 'ebonit

loaded_extensions = {}

classmethod load_all(*try_lazy=True*)
Load all (builtin and additional) extensions

Parameters **try_lazy** – if *False*, use force load for all builtin extensions

classmethod load(*extension: Union[str, ebonite.ext.ext_loader.Extension]*)
Load single extension

Parameters **extension** – str of *Extension* instance to load

Subpackages

ebonite.ext.aiohttp package

Submodules

ebonite.ext.aiohttp.server module

ebonite.ext.catboost package

Submodules

ebonite.ext.catboost.model module

ebonite.ext.docker package

class ebonite.ext.docker.**DockerRegistry**
Bases: ebonite.ext.docker.base.DockerRegistry, pyjackson.decorators.SubtypeRegisterMixin

Registry for docker images. This is the default implementation that represents registry of the docker daemon

type = 'pyjackson.decorators.DockerRegistry'

class ebonite.ext.docker.**DockerContainer**(*name: str*, *port_mapping: Dict[int, int] = None*, *params: Dict[str, object] = None*, *container_id: str = None*)

Bases: ebonite.core.objects.core.Params

RuntimeInstance.Params implementation for docker containers

Parameters

- **name** – name of the container
- **port_mapping** – port mapping in this container
- **params** – other parameters for docker run cmd
- **container_id** – internal docker id for this container

type = 'ebonite.ext.docker.base.DockerContainer'

class ebonite.ext.docker.**DockerEnv**(*registry: ebonite.ext.docker.base.DockerRegistry = None*, *daemon: ebonite.ext.docker.base.DockerDaemon = None*)

Bases: ebonite.core.objects.core.Params

RuntimeEnvironment.Params implementation for docker environment

Parameters

- **registry** – default registry to push images to
- **daemon** – DockerDaemon instance

get_runner()

Returns docker runner

get_builder()

Returns docker builder instance

type = 'ebonite.ext.docker.base.DockerEnv'

class ebonite.ext.docker.**DockerImage**(*name: str*, *tag: str = 'latest'*, *repository: str = None*, *registry: ebonite.ext.docker.base.DockerRegistry = None*, *image_id: str = None*)

Bases: ebonite.core.objects.core.Params

Image.Params implementation for docker images full uri for image looks like registry.host/repository/name:tag

Parameters

- **name** – name of the image
- **tag** – tag of the image
- **repository** – repository of the image
- **registry** – *DockerRegistry* instance with this image
- **image_id** – docker internal id of this image

fullname

uri

exists(*client: docker.client.DockerClient*)

Checks if this image exists in it's registry

delete(*client: docker.client.DockerClient*, *force=False*, ***kwargs*)
Deletes image from registry

type = 'ebonite.ext.docker.base.DockerImage'

class ebonite.ext.docker.**RemoteRegistry**(*host: str = None*)
Bases: ebonite.ext.docker.base.DockerRegistry

DockerRegistry implementation for official Docker Registry (as in https://docs.docker.com/registry/)

Parameters **host** – adress of the registry

login(*client*)
Logs in to Docker registry

Corresponding credentials should be specified as environment variables per registry: e.g., if registry host is "168.32.25.1:5000" then "168_32_25_1_5000_USERNAME" and "168_32_25_1_5000_PASSWORD" variables should be specified

Parameters **client** – Docker client instance

Returns nothing

get_host() → str
Returns registry host or emty string for local

push(*client*, *tag*)
Pushes image to registry

Parameters

- **client** – DockerClient to use
- **tag** – name of the tag to push

uri(*image: str*)
Cretate an uri for image in this registry

Parameters **image** – image name

image_exists(*client*, *image: ebonite.ext.docker.base.DockerImage*)
Check if image exists in this registry

Parameters

- **client** – DockerClient to use
- **image** – *DockerImage* to check

delete_image(*client*, *image: ebonite.ext.docker.base.DockerImage*, *force=False*, ***kwargs*)
Deleta image from this registry

Parameters

- **client** – DockerClient to use
- **image** – *DockerImage* to delete
- **force** – force delete

type = 'ebonite.ext.docker.base.RemoteRegistry'

class ebonite.ext.docker.**DockerRunner**
Bases: ebonite.build.runner.base.RunnerBase

RunnerBase implementation for docker containers

instance_exists(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*) → bool
Checks if instance exists in environment

Parameters

- **instance** – instance params to check
- **env** – environment to check in

Returns boolean flag

remove_instance(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*)
Removes instance

Parameters

- **instance** – instance params to remove
- **env** – environment to remove from

instance_type() → Type[ebonite.ext.docker.base.DockerContainer]

Returns subtype of *RuntimeInstance.Params* supported by this runner

create_instance(*name: str*, *port_mapping: Dict[int, int] = None*, ***kwargs*) → ebonite.ext.docker.base.DockerContainer
Creates new runtime instance on given name and args

Parameters **name** – name of instance to use

Returns created *RuntimeInstance.Params* subclass instance

run(*instance: ebonite.ext.docker.base.DockerContainer*, *image: ebonite.ext.docker.base.DockerImage*, *env: ebonite.ext.docker.base.DockerEnv*, *rm=True*, *detach=True*, ***kwargs*)
Runs given image on given environment with params given by instance

Parameters

- **instance** – instance params to use for running
- **image** – image to base instance on
- **env** – environment to run on

logs(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*) → Generator[str, None, None]
Exposes logs produced by given instance while running on given environment

Parameters

- **instance** – instance to expose logs for
- **env** – environment to expose logs from

Returns generator of log strings or string with logs

is_running(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*) → bool
Checks that given instance is running on given environment

Parameters

- **instance** – instance to check running of
- **env** – environment to check running on

Returns "is running" flag

stop(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*)
Stops running of given instance on given environment

Parameters

- **instance** – instance to stop running of
- **env** – environment to stop running on

class ebonite.ext.docker.**RunnerBase**
Bases: object

instance_type() → Type[ebonite.core.objects.core.RuntimeInstance.Params]

Returns subtype of *RuntimeInstance.Params* supported by this runner

create_instance(*name: str*, ***kwargs*) → ebonite.core.objects.core.RuntimeInstance.Params
Creates new runtime instance on given name and args

Parameters **name** – name of instance to use

Returns created *RuntimeInstance.Params* subclass instance

run(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *image: ebonite.core.objects.core.Image.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Runs given image on given environment with params given by instance

Parameters

- **instance** – instance params to use for running
- **image** – image to base instance on
- **env** – environment to run on

is_running(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*) → bool
Checks that given instance is running on given environment

Parameters

- **instance** – instance to check running of
- **env** – environment to check running on

Returns "is running" flag

stop(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Stops running of given instance on given environment

Parameters

- **instance** – instance to stop running of
- **env** – environment to stop running on

logs(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*) → Generator[str, None, None]
Exposes logs produced by given instance while running on given environment

Parameters

- **instance** – instance to expose logs for

- **env** – environment to expose logs from

Returns generator of log strings or string with logs

instance_exists(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*) → bool
Checks if instance exists in environment

Parameters

- **instance** – instance params to check
- **env** – environment to check in

Returns boolean flag

remove_instance(*instance: ebonite.core.objects.core.RuntimeInstance.Params*, *env: ebonite.core.objects.core.RuntimeEnvironment.Params*, ***kwargs*)
Removes instance

Parameters

- **instance** – instance params to remove
- **env** – environment to remove from

class ebonite.ext.docker.**DockerBuilder**
Bases: ebonite.build.builder.base.BuilderBase

Builder implementation to build docker images

create_image(*name: str*, *environment: ebonite.ext.docker.base.DockerEnv*, *tag: str = 'latest'*, *repository: str = None*, ***kwargs*) → ebonite.ext.docker.base.DockerImage
Abstract method to create image

build_image(*buildable: ebonite.core.objects.core.Buildable*, *image: ebonite.ext.docker.base.DockerImage*, *environment: ebonite.ext.docker.base.DockerEnv*, *force_overwrite=False*, ***kwargs*)
Abstract method to build image

delete_image(*image: ebonite.ext.docker.base.DockerImage*, *environment: ebonite.ext.docker.base.DockerEnv*, *force=False*, ***kwargs*)
Abstract method to delete image

image_exists(*image: ebonite.ext.docker.base.DockerImage*, *environment: ebonite.ext.docker.base.DockerEnv*, ***kwargs*) → bool
Abstract method to check if image exists

class ebonite.ext.docker.**DockerIORegistry**
Bases: ebonite.ext.docker.base.DockerRegistry

The class represents docker.io registry.

get_host() → str
Returns registry host or emty string for local

push(*client*, *tag*)
Pushes image to registry

Parameters

- **client** – DockerClient to use
- **tag** – name of the tag to push

image_exists(*client*, *image: ebonite.ext.docker.base.DockerImage*)
Check if image exists in this registry

Parameters

- **client** – DockerClient to use
- **image** – *DockerImage* to check

delete_image(*client*, *image: ebonite.ext.docker.base.DockerImage*, *force=False*, ***kwargs*)
Deleta image from this registry

Parameters

- **client** – DockerClient to use
- **image** – *DockerImage* to delete
- **force** – force delete

type = 'ebonite.ext.docker.base.DockerIORegistry'

ebonite.ext.docker.**build_docker_image**(*name: str*, *obj*, *server: ebonite.runtime.server.base.Server = None*, *env: ebonite.ext.docker.base.DockerEnv = None*, *tag: str = 'latest'*, *repository: str = None*, *force_overwrite: bool = False*, ***kwargs*) → ebonite.core.objects.core.Image
Build docker image from object

Parameters

- **name** – name of the resultimg image
- **obj** – obj to build image. must be convertible to Buildable: Model, Pipeline, list of one of those, etc.
- **server** – server to build image with
- **env** – DockerEnv to build in. Default - local docker daemon
- **tag** – image tag
- **repository** – image repository
- **force_overwrite** – wheter to force overwrite existing image

Parma kwargs additional arguments for DockerBuilder.build_image

ebonite.ext.docker.**run_docker_instance**(*image: ebonite.core.objects.core.Image*, *name: str = None*, *env: ebonite.ext.docker.base.DockerEnv = None*, *port_mapping: Dict[int, int] = None*, *instance_kwargs: Dict[str, Any] = None*, *rm: bool = False*, *detach: bool = True*, ***kwargs*) → ebonite.core.objects.core.RuntimeInstance
Create and run docker container

Parameters

- **image** – image to build from
- **name** – name of the container. defaults to image name
- **env** – DockerEnv to run in. Default - local docker daemon
- **port_mapping** – port mapping for container
- **instance_kwargs** – additional DockerInstance args
- **rm** – wheter to remove container on exit

- **detach** – wheter to detach from container after run
- **kwargs** – additional args for DockerRunner.run

Submodules

ebonite.ext.docker.build_context module

class ebonite.ext.docker.build_context.**DockerBuildArgs**(*base_image: Union[str, Callable[[str], str]] = None, python_version: str = None, templates_dir: Union[str, List[str]] = None, run_cmd: Union[bool, str] = None, package_install_cmd: str = None, prebuild_hook: Callable[[str], Any] = None*)

Bases: object

Container for DockerBuild arguments

Parameters

- **base_image** – base image for the built image in form of a string or function from python version, default: python:{python_version}
- **python_version** – Python version to use, default: version of running interpreter
- **templates_dir** – directory or list of directories for Dockerfile templates, default: ./docker_templates - *pre_install.j2* - Dockerfile commands to run before pip - *post_install.j2* - Dockerfile commands to run after pip - *post_copy.j2* - Dockerfile commands to run after pip and Ebonite distribution copy
- **run_cmd** – command to run in container, default: sh run.sh
- **package_install_cmd** – command to install packages. Default is apt-get, change it for other package manager
- **prebuild_hook** – callable to call before build, accepts python version. Used for pre-building server images

prebuild_hook

templates_dir

package_install_cmd

run_cmd

python_version

base_image

update(*other: ebonite.ext.docker.build_context.DockerBuildArgs*)

class ebonite.ext.docker.build_context.**DockerBuildContext**(*provider: ebonite.build.provider.base.PythonProvider, params: ebonite.ext.docker.base.DockerImage, force_overwrite=False, **kwargs*)

Bases: ebonite.build.builder.base.PythonBuildContext

PythonBuilder implementation for building docker containers

Parameters

- **provider** – PythonProvider instance
- **params** – params for docker image to be built
- **force_overwrite** – if false, raise error if image already exists
- **kwargs** – for possible keys, look at *DockerBuildArgs*

build(*env: ebonite.ext.docker.base.DockerEnv*) → docker.models.images.Image

ebonite.ext.docker.builder module

class ebonite.ext.docker.builder.**DockerBuilder**
Bases: ebonite.build.builder.base.BuilderBase

Builder implementation to build docker images

create_image(*name: str*, *environment: ebonite.ext.docker.base.DockerEnv*, *tag: str = 'latest'*, *repository: str = None*, ***kwargs*) → ebonite.ext.docker.base.DockerImage
Abstract method to create image

build_image(*buildable: ebonite.core.objects.core.Buildable*, *image: ebonite.ext.docker.base.DockerImage*, *environment: ebonite.ext.docker.base.DockerEnv*, *force_overwrite=False*, ***kwargs*)
Abstract method to build image

delete_image(*image: ebonite.ext.docker.base.DockerImage*, *environment: ebonite.ext.docker.base.DockerEnv*, *force=False*, ***kwargs*)
Abstract method to delete image

image_exists(*image: ebonite.ext.docker.base.DockerImage*, *environment: ebonite.ext.docker.base.DockerEnv*, ***kwargs*) → bool
Abstract method to check if image exists

ebonite.ext.docker.prebuild module

ebonite.ext.docker.prebuild.**prebuild_image**(*prebuild_path*, *name_template*, *python_version*, ***, *push=False*)

ebonite.ext.docker.prebuild.**prebuild_missing_images**(*prebuild_path*, *name_template*)

ebonite.ext.docker.runner module

exception ebonite.ext.docker.runner.**DockerRunnerException**
Bases: Exception

class ebonite.ext.docker.runner.**DockerRunner**
Bases: ebonite.build.runner.base.RunnerBase

RunnerBase implementation for docker containers

instance_exists(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*) → bool
Checks if instance exists in environment

Parameters

- **instance** – instance params to check
- **env** – environment to check in

Returns boolean flag

remove_instance(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*)
Removes instance

Parameters

- **instance** – instance params to remove
- **env** – environment to remove from

instance_type() → Type[ebonite.ext.docker.base.DockerContainer]

Returns subtype of *RuntimeInstance.Params* supported by this runner

create_instance(*name: str*, *port_mapping: Dict[int, int] = None*, ***kwargs*) → ebonite.ext.docker.base.DockerContainer
Creates new runtime instance on given name and args

Parameters **name** – name of instance to use

Returns created *RuntimeInstance.Params* subclass instance

run(*instance: ebonite.ext.docker.base.DockerContainer*, *image: ebonite.ext.docker.base.DockerImage*, *env: ebonite.ext.docker.base.DockerEnv*, *rm=True*, *detach=True*, ***kwargs*)
Runs given image on given environment with params given by instance

Parameters

- **instance** – instance params to use for running
- **image** – image to base instance on
- **env** – environment to run on

logs(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*) → Generator[str, None, None]
Exposes logs produced by given instance while running on given environment

Parameters

- **instance** – instance to expose logs for
- **env** – environment to expose logs from

Returns generator of log strings or string with logs

is_running(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*) → bool
Checks that given instance is running on given environment

Parameters

- **instance** – instance to check running of
- **env** – environment to check running on

Returns "is running" flag

stop(*instance: ebonite.ext.docker.base.DockerContainer*, *env: ebonite.ext.docker.base.DockerEnv*, ***kwargs*)
Stops running of given instance on given environment

Parameters

- **instance** – instance to stop running of
- **env** – environment to stop running on

ebonite.ext.docker.utils module

ebonite.ext.docker.utils.**is_docker_running**() → bool
Check if docker binary and docker daemon are available

Returns true or false

ebonite.ext.docker.utils.**create_docker_client**(*docker_host: str = ''*, *check=True*) → docker.client.DockerClient
Context manager for DockerClient creation

Parameters

- **docker_host** – DOCKER_HOST arg for DockerClient
- **check** – check if docker is available

Returns DockerClient instance

ebonite.ext.docker.utils.**image_exists_at_dockerhub**(*tag*)

ebonite.ext.docker.utils.**repository_tags_at_dockerhub**(*repo*)

ebonite.ext.flask package

class ebonite.ext.flask.**FlaskServer**
Bases: ebonite.runtime.server.base.BaseHTTPServer

Flask- and Flasgger-based *BaseHTTPServer* implementation

additional_sources = ['/home/docs/checkouts/readthedocs.org/user_builds/ebonite/checko

additional_options = {'docker': {'base_image': <function FlaskServer.<lambda>>, 'prel

run(*interface: ebonite.runtime.interface.base.Interface*)
Starts flask service

Parameters **interface** – runtime interface to expose via HTTP

Submodules

ebonite.ext.flask.client module

class ebonite.ext.flask.client.**HTTPClient**(*host=None*, *port=None*)
Bases: ebonite.runtime.client.base.BaseClient

Simple implementation of HTTP-based Ebonite runtime client.

Interface definition is acquired via HTTP GET call to */interface.json*, method calls are performed via HTTP POST calls to */<name>*.

Parameters

- **host** – host of server to connect to, if no host given connects to host *localhost*

- **port** – port of server to connect to, if no port given connects to port 9000

ebonite.ext.flask.server module

ebonite.ext.flask.server.**create_executor_function**(*interface: ebonite.runtime.interface.base.Interface*, *method: str*)

Creates a view function for specific interface method

Parameters

- **interface** – *Interface* instance
- **method** – method name

Returns callable view function

ebonite.ext.flask.server.**create_interface_routes**(*app*, *interface: ebonite.runtime.interface.base.Interface*)

ebonite.ext.flask.server.**create_schema_route**(*app*, *interface: ebonite.runtime.interface.base.Interface*)

ebonite.ext.flask.server.**prebuild_hook**(*python_version*)

class ebonite.ext.flask.server.**FlaskServer**

Bases: ebonite.runtime.server.base.BaseHTTPServer

Flask- and Flasgger-based *BaseHTTPServer* implementation

additional_sources = ['/home/docs/checkouts/readthedocs.org/user_builds/ebonite/checko

additional_options = {'docker': {'base_image': <function FlaskServer.<lambda>>, 'prel

run(*interface: ebonite.runtime.interface.base.Interface*)

Starts flask service

Parameters **interface** – runtime interface to expose via HTTP

ebonite.ext.flask.server.**main**()

ebonite.ext.imageio package

ebonite.ext.lightgbm package

Submodules

ebonite.ext.lightgbm.dataset module

ebonite.ext.lightgbm.model module

ebonite.ext.lightgbm.requirement module

ebonite.ext.numpy package

Submodules

ebonite.ext.numpy.dataset module

ebonite.ext.numpy.dataset_source module

ebonite.ext.pandas package

Submodules

ebonite.ext.pandas.dataset module

ebonite.ext.pandas.dataset_source module

ebonite.ext.s3 package

Submodules

ebonite.ext.s3.artifact module

ebonite.ext.sklearn package

Submodules

ebonite.ext.sklearn.metric module

class ebonite.ext.sklearn.metric.**SklearnMetricHook**

Bases: *ebonite.core.analyzer.metric.LibFunctionMixin*

base_module_name = 'sklearn'

ebonite.ext.sklearn.model module

ebonite.ext.sqlalchemy package

Submodules

ebonite.ext.sqlalchemy.models module

ebonite.ext.sqlalchemy.repository module

ebonite.ext.tensorflow package

Submodules

ebonite.ext.tensorflow.dataset module

ebonite.ext.tensorflow.model module

ebonite.ext.tensorflow_v2 package

Submodules

ebonite.ext.tensorflow_v2.dataset module

ebonite.ext.tensorflow_v2.model module

ebonite.ext.torch package

Submodules

ebonite.ext.torch.dataset module

ebonite.ext.torch.model module

ebonite.ext.xgboost package

Submodules

ebonite.ext.xgboost.dataset module

ebonite.ext.xgboost.model module

ebonite.ext.xgboost.requirement module

Submodules

ebonite.ext.ext_loader module

class ebonite.ext.ext_loader.**Extension**(*module*, *reqs: List[str]*, *force=True*, *validator=None*)

Bases: `object`

Extension descriptor

Parameters

- **module** – main extension module
- **reqs** – list of extension dependencies
- **force** – if True, disable lazy loading for this extension
- **validator** – boolean predicate which should evaluate to True for this extension to be loaded

class ebonite.ext.ext_loader.**ExtensionDict**(**extensions*)

Bases: `dict`

`_Extension` container

ebonite.ext.ext_loader.**is_tf_v1**()

ebonite.ext.ext_loader.**is_tf_v2**()

class ebonite.ext.ext_loader.**ExtensionLoader**
Bases: object

Class that tracks and loads extensions.

builtin_extensions = {'ebonite.ext.aiohttp': <Extension ebonite.ext.aiohttp>, 'ebonit

loaded_extensions = {}

classmethod load_all(*try_lazy=True*)
Load all (builtin and additional) extensions

Parameters **try_lazy** – if *False*, use force load for all builtin extensions

classmethod load(*extension: Union[str, ebonite.ext.ext_loader.Extension]*)
Load single extension

Parameters **extension** – str of *Extension* instance to load

ebonite.ext.ext_loader.**load_extensions**(**exts*)
Load extensions

Parameters **exts** – list of extension main modules

4.1.5 ebonite.repository package

class ebonite.repository.**ArtifactRepository**
Bases: ebonite.repository.artifact.base.ArtifactRepository, pyjackson.decorators.SubtypeRegisterMixin

Base abstract class for persistent repositories of artifacts

type = 'pyjackson.decorators.ArtifactRepository'

class ebonite.repository.**MetadataRepository**
Bases: ebonite.repository.metadata.base.MetadataRepository, pyjackson.decorators.SubtypeRegisterMixin

Abstract base class for persistent repositories of metadata (*core.Project*, *core.Task*, etc)

type = 'pyjackson.decorators.MetadataRepository'

class ebonite.repository.**DatasetRepository**
Bases: object

Base class for persisting datasets

save(*dataset_id: str*, *dataset: ebonite.core.objects.dataset_source.Dataset*) → ebonite.core.objects.dataset_source.DatasetSource
Method to save dataset to this repository

Parameters

- **dataset_id** – string identifier
- **dataset** – dataset to save

Returns DatasetSource that produces same Dataset

delete(*dataset_id: str*)
Method to delete dataset from this repository

Parameters **dataset_id** – dataset identifier

Subpackages

ebonite.repository.artifact package

class ebonite.repository.artifact.**ArtifactRepository**
Bases: ebonite.repository.artifact.base.ArtifactRepository, pyjackson.decorators.SubtypeRegisterMixin

Base abstract class for persistent repositories of artifacts

type = 'pyjackson.decorators.ArtifactRepository'

class ebonite.repository.artifact.**RepoArtifactBlob**(*repository: ebonite.repository.artifact.base.ArtifactRepository*)
Bases: *ebonite.core.objects.artifacts.Blob*

type = 'ebonite.repository.artifact.base.RepoArtifactBlob'

Submodules

ebonite.repository.artifact.inmemory module

class ebonite.repository.artifact.inmemory.**InMemoryArtifactRepository**
Bases: ebonite.repository.artifact.base.ArtifactRepository

ArtifactRepository implementation which stores artifacts in-memory

type = 'inmemory'

get_artifact(*artifact_type*, *artifact_id: str*) → ebonite.core.objects.artifacts.ArtifactCollection

push_artifact(*artifact_type, artifact_id: str, blobs: Dict[str, ebonite.core.objects.artifacts.Blob]*) → ebonite.core.objects.artifacts.ArtifactCollection

delete_artifact(*artifact_type*, *artifact_id: str*)

ebonite.repository.artifact.local module

class ebonite.repository.artifact.local.**LocalArtifactRepository**(*path: str = None*)
Bases: ebonite.repository.artifact.base.ArtifactRepository

ArtifactRepository implementation which stores artifacts in a local file system as directory

Param path: path to directory where artifacts are to be stored, if *None* "local_storage" directory in Ebonite distribution is used

type = 'local'

get_artifact(*artifact_type*, *artifact_id: str*) → ebonite.core.objects.artifacts.ArtifactCollection

push_artifact(*artifact_type, artifact_id: str, blobs: Dict[str, ebonite.core.objects.artifacts.Blob]*) → ebonite.core.objects.artifacts.ArtifactCollection

delete_artifact(*artifact_type*, *artifact_id: str*)

ebonite.repository.dataset package

class ebonite.repository.dataset.**DatasetRepository**
Bases: object

Base class for persisting datasets

save(*dataset_id: str*, *dataset: ebonite.core.objects.dataset_source.Dataset*) → ebonite.core.objects.dataset_source.DatasetSource
Method to save dataset to this repository

Parameters

- **dataset_id** – string identifier
- **dataset** – dataset to save

Returns DatasetSource that produces same Dataset

delete(*dataset_id: str*)
Method to delete dataset from this repository

Parameters **dataset_id** – dataset identifier

Submodules

ebonite.repository.dataset.artifact module

class ebonite.repository.dataset.artifact.**DatasetReader**
Bases: *ebonite.repository.dataset.artifact.DatasetReader*, pyjackson.decorators.SubtypeRegisterMixin

ABC for reading Dataset from files (artifacts) to use with ArtifactDatasetSource

type = 'pyjackson.decorators.DatasetReader'

class ebonite.repository.dataset.artifact.**DatasetWriter**
Bases: *ebonite.repository.dataset.artifact.DatasetWriter*, pyjackson.decorators.SubtypeRegisterMixin

ABC for writing Dataset to files (artifacts) to use with ArtifactDatasetSource

type = 'pyjackson.decorators.DatasetWriter'

class ebonite.repository.dataset.artifact.**OneFileDatasetReader**(*dataset_type: ebonite.core.objects.dataset_type.Datase*
Bases: *ebonite.repository.dataset.artifact.DatasetReader*

read(*artifacts: ebonite.core.objects.artifacts.ArtifactCollection*) → ebonite.core.objects.dataset_source.Dataset
Method to read Dataset from artifacts

Parameters **artifacts** – artifacts to read

convert(*payload: bytes*)

type = 'ebonite.repository.dataset.artifact.OneFileDatasetReader'

class ebonite.repository.dataset.artifact.**OneFileDatasetWriter**
Bases: *ebonite.repository.dataset.artifact.DatasetWriter*

FILENAME = 'data'

convert(*instance*) → bytes

write(*dataset: ebonite.core.objects.dataset_source.Dataset*) → Tuple[ebonite.repository.dataset.artifact.DatasetReader, ebonite.core.objects.artifacts.ArtifactCollection]
Method to write dataset to artifacts

Parameters **dataset** – dataset to write

Returns tuple of DatasetReader and ArtifactCollection.

DatasetReader must produce the same dataset if used with same artifacts

type = 'ebonite.repository.dataset.artifact.OneFileDatasetWriter'

class ebonite.repository.dataset.artifact.**PrimitiveDatasetReader**(*dataset_type: ebonite.core.objects.dataset_type.Dat*

Bases: *ebonite.repository.dataset.artifact.OneFileDatasetReader*

convert(*payload: bytes*)

type = 'ebonite.repository.dataset.artifact.PrimitiveDatasetReader'

class ebonite.repository.dataset.artifact.**PrimitiveDatasetWriter**

Bases: *ebonite.repository.dataset.artifact.OneFileDatasetWriter*

convert(*instance*) → bytes

type = 'ebonite.repository.dataset.artifact.PrimitiveDatasetWriter'

class ebonite.repository.dataset.artifact.**PickleReader**(*dataset_type: ebonite.core.objects.dataset_type.DatasetType*)

Bases: *ebonite.repository.dataset.artifact.OneFileDatasetReader*

convert(*payload: bytes*)

type = 'ebonite.repository.dataset.artifact.PickleReader'

class ebonite.repository.dataset.artifact.**PickleWriter**

Bases: *ebonite.repository.dataset.artifact.OneFileDatasetWriter*

convert(*instance*) → bytes

type = 'ebonite.repository.dataset.artifact.PickleWriter'

class ebonite.repository.dataset.artifact.**ArtifactDatasetRepository**(*repo: ebonite.repository.artifact.base.A*

Bases: ebonite.repository.dataset.base.DatasetRepository

DatasetRpository implementation that saves datasets as artifacts to ArtifactRepository

Parameters **repo** – underlying ArtifactRepository

ARTIFACT_TYPE = 'datasets'

save(*dataset_id: str*, *dataset: ebonite.core.objects.dataset_source.Dataset*) → ebonite.core.objects.dataset_source.DatasetSource
Method to save dataset to this repository

Parameters

- **dataset_id** – string identifier
- **dataset** – dataset to save

Returns DatasetSource that produces same Dataset

delete(*dataset_id: str*)
Method to delete dataset from this repository

Parameters **dataset_id** – dataset identifier

class ebonite.repository.dataset.artifact.**ArtifactDatasetSource**(*reader: ebonite.repository.dataset.artifact.Data artifacts: ebonite.core.objects.artifacts.ArtifactC dataset_type: ebonite.core.objects.dataset_type.Datas*)

Bases: *ebonite.core.objects.dataset_source.DatasetSource*

DatasetSource for reading datasets from ArtifactDatasetRepository

Parameters

- **reader** – DatasetReader for this dataset
- **artifacts** – ArtifactCollection with actual files
- **dataset_type** – DatasetType of contained dataset

read() → ebonite.core.objects.dataset_source.Dataset
Abstract method that must return produced Dataset instance

type = 'ebonite.repository.dataset.artifact.ArtifactDatasetSource'

ebonite.repository.metadata package

class ebonite.repository.metadata.**MetadataRepository**
Bases: ebonite.repository.metadata.base.MetadataRepository, pyjackson.decorators.SubtypeRegisterMixin

Abstract base class for persistent repositories of metadata (*core.Project*, *core.Task*, etc)

type = 'pyjackson.decorators.MetadataRepository'

Submodules

ebonite.repository.metadata.local module

class ebonite.repository.metadata.local.**LocalMetadataRepository**(*path=None*)
Bases: ebonite.repository.metadata.base.MetadataRepository

MetadataRepository implementation which stores metadata in a local filesystem as JSON file.

Warning: file storage is completely overwritten on each update, thus this repository is not suitable for high-performance scenarios.

Parameters **path** – path to json with the metadata, if *None* metadata is stored in-memory.

type = 'local'

load()

save()

get_projects() → List[ebonite.core.objects.core.Project]
Gets all projects in the repository

Returns all projects in the repository

get_project_by_name(*name: str*) → ebonite.core.objects.core.Project
Finds project in the repository by name

Parameters **name** – name of the project to return

Returns found project if exists or *None*

get_project_by_id(*id*) → ebonite.core.objects.core.Project
Finds project in the repository by identifier

Parameters **id** – project id

Returns found project if exists or *None*

create_project(*project: ebonite.core.objects.core.Project*) → ebonite.core.objects.core.Project
Creates the project and all its tasks.

Parameters **project** – project to create

Returns created project

Exception *errors.ExistingProjectError* if given project has the same name as existing one.

update_project(*project: ebonite.core.objects.core.Project*) → ebonite.core.objects.core.Project
Updates the project and all its tasks.

Parameters **project** – project to update

Returns updated project

Exception *errors.NonExistingProjectError* if given project doesn't exist in the repository

delete_project(*project: ebonite.core.objects.core.Project*)
Deletes the project and all tasks.

Parameters **project** – project to delete

Returns nothing

Exception *errors.NonExistingProjectError* if given project doesn't exist in the repository

get_tasks(*project: Union[int, str, core.Project]*) → List[ebonite.core.objects.core.Task]
Gets a list of tasks for given project

Parameters **project** – project to search for tasks in

Returns project tasks

get_task_by_name(*project: Union[int, str, core.Project], task_name: str*) → Optional[ebonite.core.objects.core.Task]
Finds task with given name in given project

Parameters

- **project** – project to search for task in
- **task_name** – expected name of task

Returns task if exists or *None*

get_task_by_id(*id*) → ebonite.core.objects.core.Task
Finds task with given id

Parameters **id** – id of task to search for

Returns task if exists or *None*

create_task(*task: ebonite.core.objects.core.Task*) → ebonite.core.objects.core.Task
Creates task in a repository

Parameters **task** – task to create

Returns created task

Exception *errors.ExistingTaskError* if given task has the same name and project as existing one

update_task(*task: ebonite.core.objects.core.Task*) → ebonite.core.objects.core.Task
Updates task in a repository.

Parameters **task** – task to update

Returns updated task

Exception *errors.NonExistingTaskError* if given tasks doesn't exist in the repository

delete_task(*task: ebonite.core.objects.core.Task*)
Deletes the task and all its models.

Parameters **task** – task to delete

Returns nothing

Exception *errors.NonExistingTaskError* if given tasks doesn't exist in the repository

get_models(*task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → List[ebonite.core.objects.core.Model]
Gets a list of models in given project and task

Parameters

- **task** – task to search for models in
- **project** – project to search for models in

Returns found models

get_model_by_name(*model_name: str, task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → Optional[ebonite.core.objects.core.Model]
Finds model by name in given task and project.

Parameters

- **model_name** – expected model name
- **task** – task to search for model in
- **project** – project to search for model in

Returns found model if exists or *None*

get_model_by_id(*id*) → ebonite.core.objects.core.Model
Finds model by identifier.

Parameters **id** – expected model id

Returns found model if exists or *None*

create_model(*model: ebonite.core.objects.core.Model*) → ebonite.core.objects.core.Model
Creates model in the repository

Parameters **model** – model to create

Returns created model

Exception *errors.ExistingModelError* if given model has the same name and task as existing one

update_model(*model: ebonite.core.objects.core.Model*) → ebonite.core.objects.core.Model
Updates model in the repository

Parameters **model** – model to update

Returns updated model

Exception *errors.NonExistingModelError* if given model doesn't exist in the repository

delete_model(*model: ebonite.core.objects.core.Model*)
Deletes model from the repository

Parameters **model** – model to delete

Returns nothing

Exception *errors.NonExistingModelError* if given model doesn't exist in the repository

get_pipelines(*task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → List[ebonite.core.objects.core.Pipeline]
Gets a list of pipelines in given project and task

Parameters

- **task** – task to search for models in
- **project** – project to search for models in

Returns found pipelines

get_pipeline_by_name(*pipeline_name: str, task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → Optional[ebonite.core.objects.core.Pipeline]
Finds model by name in given task and project.

Parameters

- **pipeline_name** – expected pipeline name
- **task** – task to search for pipeline in
- **project** – project to search for pipeline in

Returns found pipeline if exists or *None*

get_pipeline_by_id(*id*) → ebonite.core.objects.core.Pipeline
Finds model by identifier.

Parameters **id** – expected model id

Returns found model if exists or *None*

create_pipeline(*pipeline: ebonite.core.objects.core.Pipeline*) → ebonite.core.objects.core.Pipeline
Creates model in the repository

Parameters **pipeline** – pipeline to create

Returns created pipeline

Exception *errors.ExistingPipelineError* if given model has the same name and task as existing one

update_pipeline(*pipeline: ebonite.core.objects.core.Pipeline*) → ebonite.core.objects.core.Pipeline
Updates model in the repository

Parameters **pipeline** – pipeline to update

Returns updated model

Exception *errors.NonExistingPipelineError* if given pipeline doesn't exist in the repository

delete_pipeline(*pipeline: ebonite.core.objects.core.Pipeline*)
Deletes model from the repository

Parameters **pipeline** – pipeline to delete

Returns nothing

Exception *errors.NonExistingPipelineError* if given pipeline doesn't exist in the repository

get_images(*task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → List[ebonite.core.objects.core.Image]
Gets a list of images in given model, task and project

Parameters

- **task** – task to search for images in
- **project** – project to search for images in

Returns found images

get_image_by_name(*image_name, task: Union[int, str, core.Task], project: Union[int, str, core.Project] = None*) → Optional[ebonite.core.objects.core.Image]
Finds image by name in given model, task and project.

Parameters

- **image_name** – expected image name
- **task** – task to search for image in
- **project** – project to search for image in

Returns found image if exists or *None*

get_image_by_id(*id: int*) → Optional[ebonite.core.objects.core.Image]
Finds image by identifier.

Parameters **id** – expected image id

Returns found image if exists or *None*

create_image(*image: ebonite.core.objects.core.Image*) → ebonite.core.objects.core.Image
Creates image in the repository

Parameters **image** – image to create

Returns created image

Exception *errors.ExistingImageError* if given image has the same name and model as existing one

update_image(*image: ebonite.core.objects.core.Image*) → ebonite.core.objects.core.Image
Updates image in the repository

Parameters **image** – image to update

Returns updated image

Exception *errors.NonExistingImageError* if given image doesn't exist in the repository

delete_image(*image: ebonite.core.objects.core.Image*)
Deletes image from the repository

Parameters **image** – image to delete

Returns nothing

Exception *errors.NonExistingImageError* if given image doesn't exist in the repository

get_environments() → List[ebonite.core.objects.core.RuntimeEnvironment]
Gets a list of runtime environments

Returns found runtime environments

get_environment_by_name(*name*) → Optional[ebonite.core.objects.core.RuntimeEnvironment]
Finds runtime environment by name.

Parameters **name** – expected runtime environment name

Returns found runtime environment if exists or *None*

get_environment_by_id(*id: int*) → Optional[ebonite.core.objects.core.RuntimeEnvironment]
Finds runtime environment by identifier.

Parameters **id** – expected runtime environment id

Returns found runtime environment if exists or *None*

create_environment(*environment: ebonite.core.objects.core.RuntimeEnvironment*) → ebonite.core.objects.core.RuntimeEnvironment
Creates runtime environment in the repository

Parameters **environment** – runtime environment to create

Returns created runtime environment

Exception *errors.ExistingEnvironmentError* if given runtime environment has the same name as existing

update_environment(*environment: ebonite.core.objects.core.RuntimeEnvironment*) → ebonite.core.objects.core.RuntimeEnvironment
Updates runtime environment in the repository

Parameters **environment** – runtime environment to update

Returns updated runtime environment

Exception *errors.NonExistingEnvironmentError* if given runtime environment doesn't exist in the

repository

delete_environment(*environment: ebonite.core.objects.core.RuntimeEnvironment*)
Deletes runtime environment from the repository

Parameters **environment** – runtime environment to delete

Returns nothing

Exception *errors.NonExistingEnvironmentError* if given runtime environment doesn't exist in the

repository

get_instances(*image: Union[int, ebonite.core.objects.core.Image] = None, environment: Union[int, ebonite.core.objects.core.RuntimeEnvironment] = None*) → List[ebonite.core.objects.core.RuntimeInstance]
Gets a list of instances in given image or environment

Parameters

- **image** – image (or id) to search for instances in
- **environment** – environment (or id) to search for instances in

Returns found instances

get_instance_by_name(*instance_name, image: Union[int, ebonite.core.objects.core.Image], environment: Union[int, ebonite.core.objects.core.RuntimeEnvironment]*) → Optional[ebonite.core.objects.core.RuntimeInstance]
Finds instance by name in given image and environment.

Parameters

- **instance_name** – expected instance name
- **image** – image (or id) to search for instance in
- **environment** – environment (or id) to search for instance in

Returns found instance if exists or *None*

get_instance_by_id(*id: int*) → Optional[ebonite.core.objects.core.RuntimeInstance]
Finds instance by identifier.

Parameters **id** – expected instance id

Returns found instance if exists or *None*

create_instance(*instance: ebonite.core.objects.core.RuntimeInstance*) → ebonite.core.objects.core.RuntimeInstance
Creates instance in the repository

Parameters **instance** – instance to create

Returns created instance

Exception *errors.ExistingInstanceError* if given instance has the same name, image and environment as existing one

update_instance(*instance: ebonite.core.objects.core.RuntimeInstance*) → ebonite.core.objects.core.RuntimeInstance
Updates instance in the repository

Parameters **instance** – instance to update

Returns updated instance

Exception *errors.NonExistingInstanceError* if given instance doesn't exist in the repository

delete_instance(*instance: ebonite.core.objects.core.RuntimeInstance*)
Deletes instance from the repository

Parameters **instance** – instance to delete

Returns nothing

Exception *errors.NonExistingInstanceError* if given instance doesn't exist in the repository

4.1.6 ebonite.runtime package

class ebonite.runtime.**Interface**
Bases: object

Collection of executable methods with explicitly defined signatures

exposed = {}

executors = {}

execute(*method: str, args: Dict[str, object]*)
Executes given method with given arguments

Parameters

- **method** – method name to execute
- **args** – arguments to pass into method

Returns method result

exposed_methods()
Lists signatures of methods exposed by interface

Returns list of signatures

get_method(*method_name: str*) → callable
Returns callable exposed method object with given name

Parameters **method_name** – method name

exposed_method_signature(*method_name: str*) → pyjackson.core.Signature
Gets signature of given method

Parameters **method_name** – name of method to get signature for

Returns signature

exposed_method_docs(*method_name: str*) → str
Gets docstring for given method

Parameters **method_name** – name of the method

Returns docstring

exposed_method_args(*method_name: str*) → List[pyjackson.core.Field]
Gets argument types of given method

Parameters **method_name** – name of method to get argument types for

Returns list of argument types

exposed_method_returns(*method_name: str*) → pyjackson.core.Field
Gets return type of given method

Parameters **method_name** – name of method to get return type for

Returns return type

class ebonite.runtime.**InterfaceLoader**
Bases: ebonite.runtime.utils.RegType

Base class for loaders of *Interface*

load() → ebonite.runtime.interface.base.Interface

static get(*class_path*) → ebonite.runtime.interface.base.InterfaceLoader

ebonite.runtime.**run_model_server**(*model: ebonite.core.objects.core.Model*, *server: ebonite.runtime.server.base.Server = None*)
start_runtime() wrapper helper which starts Ebonite runtime for given model and (optional) server

Parameters

- **model** – model to start Ebonite runtime for
- **server** – server to use for Ebonite runtime, default is a flask-based server

Returns nothing

Subpackages

ebonite.runtime.client package

class ebonite.runtime.client.**BaseClient**
Bases: object

Base class for clients of Ebonite runtime.

User method calls are transparently proxied to Interface deployed on Server. PyJackson is always used for serialization of inputs and deserialization of outputs.

ebonite.runtime.interface package

exception ebonite.runtime.interface.**ExecutionError**
Bases: Exception

Exception which is raised when interface method is executed with arguments incompatible to its signature

class ebonite.runtime.interface.**Interface**
Bases: object

Collection of executable methods with explicitly defined signatures

exposed = {}

executors = {}

execute(*method: str, args: Dict[str, object]*)
Executes given method with given arguments

Parameters

- **method** – method name to execute
- **args** – arguments to pass into method

Returns method result

exposed_methods()
Lists signatures of methods exposed by interface

Returns list of signatures

get_method(*method_name: str*) → callable
Returns callable exposed method object with given name

Parameters **method_name** – method name

exposed_method_signature(*method_name: str*) → pyjackson.core.Signature
Gets signature of given method

Parameters **method_name** – name of method to get signature for

Returns signature

exposed_method_docs(*method_name: str*) → str
Gets docstring for given method

Parameters **method_name** – name of the method

Returns docstring

exposed_method_args(*method_name: str*) → List[pyjackson.core.Field]
Gets argument types of given method

Parameters **method_name** – name of method to get argument types for

Returns list of argument types

exposed_method_returns(*method_name: str*) → pyjackson.core.Field
Gets return type of given method

Parameters **method_name** – name of method to get return type for

Returns return type

class ebonite.runtime.interface.**InterfaceLoader**
Bases: ebonite.runtime.utils.RegType

Base class for loaders of *Interface*

load() → ebonite.runtime.interface.base.Interface

static get(*class_path*) → ebonite.runtime.interface.base.InterfaceLoader

ebonite.runtime.interface.**expose**(*class_method*)
Decorator which exposes given method into interface

Parameters **class_method** – method to expose

Returns given method with modifications

Submodules

ebonite.runtime.interface.ml_model module

ebonite.runtime.interface.ml_model.**model_interface**(*model_meta: ebonite.core.objects.core.Model*)
Creates an interface from given model with methods exposed by wrapper Methods signature is determined via metadata associated with given model.

Parameters **model_meta** – model to create interface for

Returns instance of *Interface* implementation

class ebonite.runtime.interface.ml_model.**ModelLoader**
Bases: ebonite.runtime.interface.base.InterfaceLoader

Implementation of *InterfaceLoader* which loads a model via PyJackson and wraps it into an interface

load() → ebonite.runtime.interface.base.Interface

class ebonite.runtime.interface.ml_model.**MultiModelLoader**
Bases: ebonite.runtime.interface.base.InterfaceLoader

Implementation of *InterfaceLoader* which loads a collection of models via PyJackson and wraps them into a single interface

load() → ebonite.runtime.interface.base.Interface

ebonite.runtime.interface.pipeline module

class ebonite.runtime.interface.pipeline.**PipelineMeta**(*pipeline: ebonite.core.objects.core.Pipeline, models: Dict[str, ebonite.core.objects.core.Model]*)

Bases: object

ebonite.runtime.interface.pipeline.**pipeline_interface**(*pipeline_meta: ebonite.core.objects.core.Pipeline*)

Creates an interface from given pipeline with *run* method Method signature is determined via metadata associated with given pipeline.

Parameters **pipeline_meta** – pipeline to create interface for

Returns instance of *Interface* implementation

class ebonite.runtime.interface.pipeline.**PipelineLoader**

Bases: ebonite.runtime.interface.base.InterfaceLoader

Implementation of *InterfaceLoader* which loads a pipeline via PyJackson and wraps it into an interface

load() → ebonite.runtime.interface.base.Interface

ebonite.runtime.interface.utils module

ebonite.runtime.interface.utils.**merge**(*ifaces: Dict[str, ebonite.runtime.interface.base.Interface]*) → ebonite.runtime.interface.base.Interface

Helper to produce composite interface from a number of interfaces. Exposes all methods of all given interfaces via given prefixes.

Parameters **ifaces** – dict with (prefix, interface) mappings

Returns composite interface

ebonite.runtime.openapi package

Submodules

ebonite.runtime.openapi.spec module

ebonite.runtime.openapi.spec.**make_object**(*properties: List[pyjackson.core.Field] = None, arbitrary_properties_type: Type[CT_co] = None, has_default=False, default=None*)

Converts object type described as list of fields to OpenAPI schema definition

Parameters

- **properties** – fields of object
- **arbitrary_properties_type** – (optional) required type for properties which are not specified in *properties*

- **has_default** – specifies whether given type has default value
- **default** – specifies default value for given type

Returns dict with OpenAPI schema definition

ebonite.runtime.openapi.spec.**make_array**(*item_type: Type[CT_co], minimum_size=None, maximum_size=None, has_default=False, default=None*)

Converts array type described as type of its items and range of possible sizes to OpenAPI schema definition

Parameters

- **item_type** – type of items in array
- **minimum_size** – minimal possible size of array
- **maximum_size** – maximal possible size of array
- **has_default** – specifies whether given type has default value
- **default** – specifies default value for given type

Returns dict with OpenAPI schema definition

ebonite.runtime.openapi.spec.**type_to_schema**(*field_type*, *has_default=False*, *default=None*)

Facade method converting arbitrary type to OpenAPI schema definitions. Has special support for builtins, collections and instances of *TypeWithSpec* subclasses.

Parameters

- **field_type** – type to generate schema for
- **has_default** – specifies whether given type has default value
- **default** – specifies default value for given type

Returns dict with OpenAPI schema definition

ebonite.runtime.openapi.spec.**create_spec**(*method_name: str*, *signature: pyjackson.core.Signature*, *name: str*, *docs: str*)

Generates OpenAPI schema definition for given method

Parameters

- **method_name** – name of method
- **signature** – types of arguments and type of return value
- **name** – name of the interface
- **docs** – docs for method

Returns dict with OpenAPi schema definition

ebonite.runtime.server package

class ebonite.runtime.server.**BaseHTTPServer**

Bases: ebonite.runtime.server.base.Server

HTTP-based Ebonite runtime server.

Interface definition is exposed for clients via HTTP GET call to */interface.json*, method calls - via HTTP POST calls to */<name>*, server health check - via HTTP GET call to */health*.

Host to which server binds is configured via *EBONITE_HOST* environment variable: default is *0.0.0.0* which means any local or remote, for rejecting remote connections use *localhost* instead.

Port to which server binds to is configured via *EBONITE_PORT* environment variable: default is 9000.

class ebonite.runtime.server.**HTTPServerConfig**
Bases: *ebonite.config.Config*

exception ebonite.runtime.server.**MalformedHTTPRequestException**(*message: str*)
Bases: Exception

code()

response_body()

class ebonite.runtime.server.**Server**
Bases: ebonite.runtime.utils.RegType

Base class for Ebonite servers

additional_sources = []

additional_binaries = []

additional_envs = {}

additional_options = {}

static get(*class_path*) → ebonite.runtime.server.base.Server
Gets a fresh instance of given server implementation

Parameters **class_path** – full name of server implementation

Returns server object

run(*executor: ebonite.runtime.interface.base.Interface*)
Main server method which "executes" given interface. Should be implemented by subclasses.

Parameters **executor** – interface to "execute"

Returns nothing

start(*loader: ebonite.runtime.interface.base.InterfaceLoader*)
Starts server "execution" for given loader: loads an interface and "executes" it

Parameters **loader** – loader to take interface from

Returns nothing

type = 'ebonite.runtime.server.base.Server'

Submodules

ebonite.runtime.command_line module

ebonite.runtime.command_line.**start_runtime**(*loader=None*, *server=None*)
Starts Ebonite runtime for given (optional) loader and (optional) server

Parameters

- **loader** – loader of model to start Ebonite runtime for, if not given class specified in config.Runtime.LOADER is used
- **server** – server to use for Ebonite runtime, default is a flask-based server, if not given class specified in config.Runtime.SERVER is used

Returns nothing

ebonite.runtime.utils module

ebonite.runtime.utils.**registering_type**(*type_name*)
Helper for base classes which maintains registry of all their subclasses

Parameters **type_name** – name for base class to use

Returns class with subclasses registry built in

4.1.7 ebonite.utils package

Submodules

ebonite.utils.abc_utils module

ebonite.utils.abc_utils.**is_abstract_method**(*cls_or_method*, *method_name=None*)
Checks that given method is abstract (has no body and should be implemented by subclass)

Parameters

- **cls_or_method** – either a class in which method *method_name* is found or method itself
- **method_name** – unused if *cls_or_method* is a method or name of method to look in *cls_or_method* class for

Returns boolean flag

ebonite.utils.classproperty module

class ebonite.utils.classproperty.**ClassPropertyDescriptor**(*f_get*, *f_set=None*)
Bases: object

Wrapper which provides access to methods through property syntax

ebonite.utils.classproperty.**classproperty**(*func*)
Decorator for properties of classes, similar to stdlib's *property* which is limited to properties of objects

Parameters **func** – function to decorate

Returns wrapper which provides access to methods through property syntax

ebonite.utils.fs module

ebonite.utils.fs.**get_lib_path**(**filename*)

ebonite.utils.fs.**current_module_path**(**path*)

ebonite.utils.fs.**switch_curdir**(*path*)
Context manager to temproary switch current dir

ebonite.utils.importing module

ebonite.utils.importing.**import_module**(*name*, *package=None*)
Import a module.

The 'package' argument is required when performing a relative import. It specifies the package to use as the anchor point from which to resolve the relative import to an absolute import.

ebonite.utils.importing.**import_string**(*dotted_path*)
Import a dotted module path and return the attribute/class designated by the last name in the path. Raise ImportError if the import failed.

ebonite.utils.importing.**module_importable**(*module_name*)

ebonite.utils.importing.**module_imported**(*module_name*)
Checks if module already imported

Parameters **module_name** – module name to check

Returns *True* or *False*

ebonite.utils.index_dict module

class ebonite.utils.index_dict.**IndexDict**(*key_field*, *index_field*, **args*, ***kwargs*)
Bases: dict, typing.Generic

add(*value: T*)

get_index(*key*, *default=Ellipsis*) → T

reindex()

clear() → None. Remove all items from D.

class ebonite.utils.index_dict.**IndexDictAccessor**(*data: ebonite.utils.index_dict.IndexDict[~T][T]*)
Bases: typing.Generic

contains(*item*)

values()

keys()

items()

get(*key*, *default=Ellipsis*) → T

ebonite.utils.log module

ebonite.utils.module module

ebonite.utils.module.**analyze_module_imports**(*module_path*)

ebonite.utils.module.**check_pypi_module**(*module_name*, *module_version=None*, *raise_on_error=False*, *warn_on_error=True*)
Checks that module with given name and (optionally) version exists in PyPi repository.

Parameters

- **module_name** – name of module to look for in PyPi

- **module_version** – (optional) version of module to look for in PyPi
- **raise_on_error** – raise *ValueError* if module is not found in PyPi instead of returning *False*
- **warn_on_error** – print a warning if module is not found in PyPi

Returns *True* if module found in PyPi, *False* otherwise

ebonite.utils.module.**get_object_base_module**(*obj: object*) → module

Determines base module of module given object comes from.

```
>>> import numpy
>>> get_object_base_module(numpy.random.Generator)
<module 'numpy' from '...'>
```

Essentially this function is a combination of *get_object_module()* and *get_base_module()*.

Parameters **obj** – object to determine base module for

Returns Python module object for base module

ebonite.utils.module.**get_base_module**(*mod: module*)

Determines base module for given module.

```
>>> import numpy
>>> get_base_module(numpy.random)
<module 'numpy' from '...'>
```

Parameters **mod** – Python module object to determine base module for

Returns Python module object for base module

ebonite.utils.module.**get_object_module**(*obj: object*) → module

Determines module given object comes from

```
>>> import numpy
>>> get_object_module(numpy.ndarray)
<module 'numpy' from '...'>
```

Parameters **obj** – obj to determine module it comes from

Returns Python module object for object module

class ebonite.utils.module.**ISortModuleFinder**

Bases: object

Determines type of module: standard library (*ISortModuleFinder.is_stdlib()*) or third party (*ISortModuleFinder.is_thirdparty()*). This class uses *isort* library heuristics with some modifications.

instance = None

classmethod init()

classmethod is_stdlib(*module: str*)

classmethod is_thirdparty(*module: str*)

ebonite.utils.module.**is_private_module**(*mod: module*)

Determines that given module object represents private module.

Parameters **mod** – module object to use

Returns boolean flag

ebonite.utils.module.**is_pseudo_module**(*mod: module*)
Determines that given module object represents pseudo (aka Python "magic") module.

Parameters **mod** – module object to use

Returns boolean flag

ebonite.utils.module.**is_extension_module**(*mod: module*)
Determines that given module object represents native code extension module.

Parameters **mod** – module object to use

Returns boolean flag

ebonite.utils.module.**is_installable_module**(*mod: module*)
Determines that given module object represents PyPi-installable (aka third party) module.

Parameters **mod** – module object to use

Returns boolean flag

ebonite.utils.module.**is_builtin_module**(*mod: module*)
Determines that given module object represents standard library (aka builtin) module.

Parameters **mod** – module object to use

Returns boolean flag

ebonite.utils.module.**is_ebonite_module**(*mod: module*)
Determines that given module object is ebonite module

Parameters **mod** – module object to use

Returns boolean flag

ebonite.utils.module.**is_local_module**(*mod: module*)
Determines that given module object represents local module. Local module is a module (Python file) which is not from standard library and not installed via pip.

Parameters **mod** – module object to use

Returns boolean flag

ebonite.utils.module.**is_from_installable_module**(*obj: object*)
Determines that given object comes from PyPi-installable (aka third party) module.

Parameters **obj** – object to check

Returns boolean flag

ebonite.utils.module.**get_module_version**(*mod: module*)
Determines version of given module object.

Parameters **mod** – module object to use

Returns version as *str* or *None* if version could not be determined

ebonite.utils.module.**get_python_version**()

Returns Current python version in 'major.minor.micro' format

ebonite.utils.module.**get_package_name**(*mod: module*) → str
Determines PyPi package name for given module object

Parameters **mod** – module object to use

Returns name as *str*

ebonite.utils.module.**get_module_repr**(*mod: module*, *validate_pypi=False*) → str
Builds PyPi *requirements.txt*-compatible representation of given module object

Parameters

- **mod** – module object to use
- **validate_pypi** – if *True* (default is *False*) perform representation validation in PyPi repository

Returns representation as *str*

ebonite.utils.module.**get_module_as_requirement**(*mod: module*, *validate_pypi=False*) → ebonite.core.objects.requirements.InstallableRequirement
Builds Ebonite representation of given module object

Parameters

- **mod** – module object to use
- **validate_pypi** – if *True* (default is *False*) perform representation validation in PyPi repository

Returns representation as *InstallableRequirement*

ebonite.utils.module.**get_local_module_reqs**(*mod*)

ebonite.utils.module.**add_closure_inspection**(*f*)

ebonite.utils.module.**get_object_requirements**(*obj*) → ebonite.core.objects.requirements.Requirements
Analyzes packages required for given object to perform its function. This function uses *pickle*/*dill* libraries serialization hooks internally. Thus result of this function depend on given object being serializable by *pickle*/*dill* libraries: all nodes in objects graph which can't be serialized are skipped and their dependencies are lost.

Parameters **obj** – obj to analyze

Returns *Requirements* object containing all required packages

ebonite.utils.pickling module

class ebonite.utils.pickling.**EbonitePickler**(**args*, ***kwds*)
Bases: dill._dill.Pickler

Base class for *pickle* serializers in Ebonite. Based on *dill* library.

class ebonite.utils.pickling.**EboniteUnpickler**(**args*, ***kwds*)
Bases: dill._dill.Unpickler

Base class for *pickle* deserializers in Ebonite. Based on *dill* library.

4.2 Submodules

4.2.1 ebonite.config module

class ebonite.config.**ConfigEnv**
Bases: object

register = True

on_top = True

get(*key*, *namespace=None*)

class ebonite.config.**Param**(*key*, *namespace=None*, *default=NO_VALUE*, *alternate_keys=NO_VALUE*, *doc=''*, *parser: Callable = <class 'str'>*, *raise_error=True*, *raw_value=False*)

Bases: object

class ebonite.config.**Config**

Bases: object

class ebonite.config.**Core**

Bases: *ebonite.config.Config*

class ebonite.config.**Logging**

Bases: *ebonite.config.Config*

class ebonite.config.**Runtime**

Bases: *ebonite.config.Config*

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

ebonite could always use more documentation, whether as part of the official ebonite docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/zyfra/ebonite/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *ebonite* for local development:

1. Fork ebonite (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:zyfra/ebonite.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Github Actions - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Mikhail Sveshnikov - https://github.com/mike0sv
- Timur Iakobidze - https://github.com/TimurPlusPlus
- Ivan Andrianov - https://github.com/i-a-andrianov
- Mikhail Trofimov - https://github.com/geffy

CHAPTER 7

Changelog

7.1 Current release candidate

7.2 0.6.2 (2020-06-18)

- Minor bugfixes

7.3 0.6.1 (2020-06-15)

- Deleted accidental debug 'print' call :/

7.4 0.6.0 (2020-06-12)

- Prebuilt flask server images for faster image build
- More and better methods in Ebonite client
- Pipelines - chain Models methods into one Model-like objects
- Refactioring of image and instance API
- Rework of pandas DatasetType: now with column types, even non-primitive (e.g. datetimes)
- Helper functions for stanalone docker build/run
- Minor bugfixes and features

7.5 0.5.2 (2020-05-16)

- Fixed dependency inspection to include wrapper dependencies
- Fixed s3 repo to fail with subdirectories
- More flexible way to add parameters for instance running (e.g. docker run arguments)
- Added new type of Requirement to represent unix packages - for example, libgomp for xgboost
- Minor tweaks

7.6 0.5.1 (2020-04-16)

- Minor fixes and examples update

7.7 0.5.0 (2020-04-10)

- Built Docker images and running Docker containers along with their metadata are now persisted in metadata repository
- Added possibility to track running status of Docker container via Ebonite client
- Implemented support for pushing built images to remote Docker registry
- Improved testing of metadata repositories and Ebonite client and fixed discovered bugs in them
- Fixed bug with failed transactions not being rolled back
- Fixed bug with serialization of complex models some component of which could not be pickled
- Decomposed model IO from model wrappers
- bytes are now used for binary datasets instead of file-like objects
- Eliminated build_model_flask_docker in favor of Server-driven abstraction
- Sped up PickleModelIO by avoiding ModelAnalyzer calls for non-model objects
- Sped up Model.create by calling model methods with given input data just once
- Dataset types and model wrappers expose their runtime requirements

7.8 0.4.0 (2020-02-17)

- Implemented asyncio-based server via aiohttp library
- Implemented support for Tensorflow 2.x models
- Changed default type of base python docker image to “slim”
- Added ‘description’ and ‘params’ fields to Model. ‘description’ is a text field and ‘params’ is a dict with arbitrary keys
- Fixed bug with building docker image with different python version that the Model was created with

7.9 0.3.5 (2020-01-31)

- Fixed critical bug with wrapper_meta

7.10 0.3.4 (2020-01-31)

- Fixed bug with deleting models from tasks
- Support working with model meta without requiring installation of all model dependencies
- Added region argument for s3 repository
- Support for delete_model in Ebonite client
- Support for force flag in delete_model which deletes model even if artifacts could not be deleted

7.11 0.3.3 (2020-01-10)

- Eliminated tensorflow warnings. Added more tests for providers/loaders. Fixed bugs in multi-model provider/builder.
- Improved documentation
- Eliminate useless "which docker" check which fails on Windows hosts
- Perform redirect from / to Swagger API docs in Flask server
- Support for predict_proba method in ML model
- Do not fix first dimension size for numpy arrays and torch tensors
- Support for Pytorch JIT (TorchScript) models
- Bump tensorflow from 1.14.0 to 1.15.0
- Added more tests

7.12 0.3.2 (2019-12-04)

- Multi-model interface bug fixes

7.13 0.3.1 (2019-12-04)

- Minor bug fixes

7.14 0.3.0 (2019-11-27)

- Added support for LightGBM models
- Added support for XGBoost models
- Added support for PyTorch models

- Added support for CatBoost models
- Added uwsgi server for flask containers

7.15 0.2.1 (2019-11-19)

- Minor bug fixes

7.16 0.2.0 (2019-11-14)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index

A

B

C

D

E

N

O

P

R

S

T

U

V

W