pythx Documentation

Release 1.7.3

Dominik Muhs

May 12, 2022

Contents:

CHAPTER 1

PythX

PythX is a library for the MythX smart contract security analysis platform.

Table of Contents

1.1 What is MythX?

MythX is a security analysis API that allows anyone to create purpose-built security tools for smart contract developers. Tools built on MythX integrate seamlessly into the development environments and continuous integration pipelines used throughout the Ethereum ecosystem.

1.2 Installation

PythX runs on Python 3.6+ and PyPy3.

To get started, simply run

```
$ pip3 install pythx
```

Alternatively, clone the repository and run

```
$ pip3 install .
```

Or directly through Python's `setuptools`:

```
$ python3 setup.py install
```

1.3 Example

PythX aims to provide an easy-to-use interface to the official MythX API. Its goal is to turbocharge tool development and make it easy to deal with even complex use cases.

```
from pythx import Client


c = Client(api_key="...")

# submit bytecode, source files, their AST and more!
resp = c.analyze(bytecode="0xfe")

# wait for the analysis to finish
while not c.analysis_ready(resp.uuid):
    time.sleep(1)

# have all your security report data at your fingertips
for issue in c.report(resp.uuid):
    print(issue.swc_title or "Undefined", "-", issue.description_short)

# Output:
# Assert Violation - A reachable exception has been detected.
```

1.4 The PythX CLI has now become the MythX CLI!

Originally, the PythX CLI was a proof of concept to display to interested developers what can be done using the library. The interest in the CLI grew so large that a lot of developers contacted me and asked for support and new features.

This is the PSA that **I will no longer maintain the PythX CLI**. But wait! There's more!

Because a PoC is not exactly what you would call future-proof and maintainable software, I have decided to do a complete revamp. It is called *mythx-cli* and incorporates all feature requests I have gotten so far. Check it out here and let me know what you think!

Enjoy! :)

CHAPTER 2

Advanced Installation

2.1 Stable release

To install pythx, run this command in your terminal:

```
$ pip3 install pythx
```

This is the preferred method to install pythx, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for pythx can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/dmuhs/pythx
```

Or download the tarball:

```
$ curl  -OL https://github.com/dmuhs/pythx/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

pythx

3.1 pythx package

3.1.1 Subpackages

pythx.api package

Submodules

pythx.api.client module

This module contains the main API Client implementation.

class pythx.api.client.**Client**(*username: str = None*, *password: str = None*, *api_key: str = None*, *refresh_token: str = None*, *handler: pythx.api.handler.APIHandler = None*, *no_cache: bool = False*, *middlewares: List[pythx.middleware.base.BaseMiddleware] = None*, *api_url: str = None*)

Bases: object

The main class for API interaction.

The client makes sure that you are authenticated at all times. For authentication data it required either the account's Ethereum address *and* password, or a valid combination of access *and* refresh token. If any token expires, the client will automatically try to refresh the access token, or log the user in again. After that, the original request is executed.

Furthermore, the client class supports various actions for high-level usage to easily submit new analysis jobs, check their status, get notified whether they are ready, and fetch analysis job report data.

A user can inject custom middlewares. There are two required internal ones:

1. `ClientToolNameMiddleware` Fills in the `clientToolName` field for new analysis submissions
2. `AnalysisCacheMiddleware` Sets the `noCacheLookup` field in new analysis submissions

These middlewares can also be overwritten by the user (even though using the Client parameters is recommended). If any of these middleware instances are missing in the user-defined list, e.g. because they simply add their own ones, the Client constructor will automatically add them with their default or parameter-defined values (if given).

add_group_to_project(*group_id: str*, *project_id: str*) → mythx_models.response.group_operation.GroupOperationResponse

Adds group to the project.

This will add the group to the project in the MythX platform.

Parameters

- **group_id** – The target group ID
- **project_id** – The target project ID

Returns `GroupOperationResponse`

analysis_list(*date_from: datetime.datetime = None*, *date_to: datetime.datetime = None*, *offset: int = None*, *created_by: str = None*, *group_name: str = None*, *group_id: str = None*, *main_source: str = None*) → mythx_models.response.analysis_list.AnalysisListResponse

Get a list of the user's analyses jobs.

Parameters

- **date_from** – Start of the date range (optional)
- **date_to** – End of the date range (optional)
- **offset** – The number of results to skip (used for pagination)
- **created_by** – Filter analysis results based on the creator
- **group_name** – Filter analysis results based on the group name
- **group_id** – Filter analysis results based on their group ID
- **main_source** – Filter analysis results based on their main source name

Returns `AnalysisListResponse`

analysis_ready(*uuid: str*) → bool

Return a boolean whether the analysis job with the given UUID has finished processing.

Parameters **uuid** – The analysis job UUID

Returns bool indicating whether the analysis has finished

analysis_status(*uuid: str*) → mythx_models.response.analysis_status.AnalysisStatusResponse

Get the status of an analysis job based on its UUID.

Parameters **uuid** – The job's UUID

Returns `AnalysisStatusResponse`

analyze(*bytecode: str = None*, *main_source: str = None*, *sources: Dict[str, Dict[str, str]] = None*, *contract_name: str = None*, *source_map: str = None*, *deployed_bytecode: str = None*, *deployed_source_map: str = None*, *source_list: List[str] = None*, *solc_version: str = None*, *analysis_mode: str = 'quick'*, *payload: mythx_models.request.analysis_submission.AnalysisSubmissionRequest = None*) → mythx_models.response.analysis_submission.AnalysisSubmissionResponse

Submit a new analysis job.

At least the smart contracts bytecode, or it's source code must be given. The more information the MythX API gets, the more precise and verbose the results will be.

Parameters

- **contract_name** – The main Solidity contract's name
- **bytecode** – The EVM creation bytecode obtained
- **source_map** – The source map for the EVM creation bytecode
- **deployed_bytecode** – The deployed EVM bytecode
- **deployed_source_map** – The deployed bytecode's source map
- **main_source** – The main source file to start analysis from
- **sources** – A dictionary holding the source file data
- **source_list** – A list of source files (ordered by the source map locs)
- **solc_version** – The solc version used for compilation
- **analysis_mode** – The analysis mode
- **payload** – Directly inject an `AnalysisSubmissionRequest` model

Returns `AnalysisSubmissionResponse`

assert_authentication() → None
Make sure the user is authenticated.

If necessary, this method will refresh the access token, or perform another login to get a fresh combination of tokens if both are expired.

Returns None

create_group(*group_name: str = ''*) → mythx_models.response.group_creation.GroupCreationResponse
Create a new group.

Parameters **group_name** – The name of the group (max. 256 characters, optional)

Returns `GroupCreationResponse`

create_project(*name: str = ''*, *description: str = ''*, *groups: List[str] = None*) → mythx_models.response.project_creation.ProjectCreationResponse
Create a new project.

Parameters

- **name** – The project name
- **description** – The project description
- **groups** – List of group IDs belonging to the project (optional)

Returns `ProjectCreationResponse`

delete_project(*project_id: str = ''*) → mythx_models.response.project_deletion.ProjectDeletionResponse
Delete an existing project.

Parameters **project_id** – The project's ID

Returns `ProjectDeletionResponse`

group_list(*offset: int = None*, *created_by: str = ''*, *group_name: str = ''*, *date_from: datetime.datetime = None*, *date_to: datetime.datetime = None*) → mythx_models.response.group_list.GroupListResponse
Get a list of the currently defined MythX analysis groups.

Parameters

- **offset** – The number of results to skip (used for pagination)
- **created_by** – Filter the list results by the creator's user ID
- **group_name** – Filter the list results by the group's name
- **date_from** – Only display results after the given date
- **date_to** – Only display results until the given date

Returns GroupListResponse

group_status(*group_id: str*) → mythx_models.response.group_status.GroupStatusResponse
Get the status of an analysis group by its ID.

Parameters **group_id** – The group ID to fetch the status for

Returns GroupStatusResponse

login() → mythx_models.response.auth_login.AuthLoginResponse
Perform a login request on the API and return the response.

Returns AuthLoginResponse

logout() → mythx_models.response.auth_logout.AuthLogoutResponse
Perform a logout request on the API and return the response.

Returns AuthLogoutResponse

project_list(*offset: int = None*, *limit: int = None*, *name: str = ''*) → mythx_models.response.project_list.ProjectListResponse
List the existing projects on the platform

Parameters

- **offset** – The number of projects to skip (optional)
- **limit** – The number of projects to return (optional)
- **name** – The name to filter projects by (optional)

Returns

project_status(*project_id: str = ''*) → mythx_models.response.project_status.ProjectStatusResponse
Get detailed information for a project.

Parameters **project_id** – The project's ID

Returns ProjectStatusResponse

refresh() → mythx_models.response.auth_refresh.AuthRefreshResponse
Perform a JWT refresh on the API and return the response.

Returns AuthRefreshResponse

report(*uuid: str*) → mythx_models.response.detected_issues.DetectedIssuesResponse
Get the report holding found issues for an analysis job based on its UUID.

Parameters **uuid** – The analysis job UUID

Returns DetectedIssuesResponse

request_by_uuid(*uuid: str*) → mythx_models.response.analysis_input.AnalysisInputResponse
Get the input request based on the analysis job's UUID.

Parameters **uuid** – The analysis job UUID

Returns `AnalysisInputResponse`

seal_group(*group_id: str*) → mythx_models.response.group_operation.GroupOperationResponse
Seal the group.

This closes an open group for the submission of any further analyses.

Parameters **group_id** – The target group ID

Returns `GroupOperationResponse`

update_project(*project_id: str = ''*, *name: str = ''*, *description: str = ''*) → mythx_models.response.project_update.ProjectUpdateResponse
Update an existing project.

A new name, a new description, or both should be given.

Parameters

- **project_id** – The ID of the project to update
- **name** – The new project name (optional)
- **description** – The new project description (optional)

Returns `ProjectUpdateResponse`

version() → mythx_models.response.version.VersionResponse
Call the APIs version endpoint to get its backend version numbers.

Returns `VersionResponse`

pythx.api.handler module

This module contains the API request handler implementation.

class pythx.api.handler.**APIHandler**(*middlewares: List[pythx.middleware.base.BaseMiddleware] = None*, *api_url: str = None*)
Bases: `object`

Handle the low-level API interaction.

The API handler takes care of serializing API requests, sending them to the configured endpoint, parsing the response into its respective domain model, as well as registering and executing request/response middlewares.

assemble_request(*req: [<class 'mythx_models.request.analysis_input.AnalysisInputRequest'>, <class 'mythx_models.request.analysis_list.AnalysisListRequest'>, <class 'mythx_models.request.analysis_status.AnalysisStatusRequest'>, <class 'mythx_models.request.analysis_submission.AnalysisSubmissionRequest'>, <class 'mythx_models.request.auth_login.AuthLoginRequest'>, <class 'mythx_models.request.auth_logout.AuthLogoutRequest'>, <class 'mythx_models.request.auth_refresh.AuthRefreshRequest'>, <class 'mythx_models.request.detected_issues.DetectedIssuesRequest'>, <class 'mythx_models.request.group_creation.GroupCreationRequest'>, <class 'mythx_models.request.group_list.GroupListRequest'>, <class 'mythx_models.request.group_operation.GroupOperationRequest'>, <class 'mythx_models.request.group_status.GroupStatusRequest'>, <class 'mythx_models.request.project_creation.ProjectCreationRequest'>, <class 'mythx_models.request.project_deletion.ProjectDeleteRequest'>, <class 'mythx_models.request.project_list.ProjectListRequest'>, <class 'mythx_models.request.project_status.ProjectStatusRequest'>, <class 'mythx_models.request.project_update.ProjectUpdateRequest'>, <class 'mythx_models.request.version.VersionRequest'>]*) → Dict[KT, VT]

Assemble a request that is later sent to the API.

This method generates an intermediate data dictionary format holding all the relevant request data needed by the API. This encompasses the HTTP verb, the request payload content (if there is any), the request's URL parameters, additional headers, as well as the API endpoint the request should be sent to.

Each of these data points is encoded in the domain model as a property. The endpoint URL is constructed from the domain model's path (e.g. `/v1/auth/login`) and the API base path: `https://api.mythx.io`, which is contained in the library configuration module.

Before the serialized request is returned, all registered middlewares are applied to it.

Parameters **req** – The request domain model

Returns The serialized request with all middlewares applied

execute_request_middlewares(*req: Dict[KT, VT]*) → Dict[KT, VT]

Sequentially execute the registered request middlewares.

Each middleware gets the request's data dictionary as generated by the APIHandler.assemble_request method. On top of the request any manipulations can be made.

It is worth mentioning here that this is a simple loop iterating over the middleware list, calling each middleware's `process_request` method. It is expected that each registered middleware exposes this method and returns a data dictionary in the same format as the one passed in. It also means that the order in which middlewares are registered can matter, even though it is recommended that middlewares are kept associative in nature.

Parameters **req** – The request's data dictionary

Returns The updated data dict - ready to be sent to the API

execute_response_middlewares(*resp: Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]*) → Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]

Sequentially execute the registered response middlewares.

Each middleware gets the serialized response domain model. On top of the request any manipulations can be made. Furthermode, each domain model's helper methods can be used.

It is worth mentioning here that this is a simple loop iterating over the middleware list, calling each middleware's `process_response` method. It is expected that each registered middleware exposes this method and returns a domain model of the same type as the one passed in. It also means that the order in which middlewares are registered can matter, even though it is recommended that middlewares are kept associative in nature.

Parameters **resp** – The response domain model

Returns The updated response domain model - ready to be passed on to the user

parse_response(*resp: dict, model_cls: Type[Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]]*) → Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]

Parse the API response into its respective domain model variant.

This method takes the raw HTTP response and a class it should deserialize the responsse data into. As each domain model implements the `from_json` method, we simply call it on the raw input data and return the resulting model.

If a deserialization or validation error is raised, it is not caught and directly passed on to the user.

Parameters

- **resp** – The raw HTTP response JSON payload
- **model_cls** – The domain model class the data should be deserialized into

Returns The domain model holding the response data

static send_request(*request_data: Dict[KT, VT], auth_header: Dict[str, str] = None*) → Dict[KT, VT]

Send a request to the API.

This method takes a data dictionary holding the request's method (HTTP verb), any additional headers, the URL to send the request to, its payload, and any URL parameters it requires. This dictionary is generated

by the APIHandler.assemble_request method.

An example for getting the detected issues for an analysis job's UUID:

```
{
    "method": "GET",
    "headers": {},
    "url": "https://api.mythx.io/v1/analyses/<uuid>/issues",
    "payload": "",
    "params": {}
}
```

If the action requires authentication, the auth headers are passed in a separate, optional parameter. It holds the user's JWT access token.

If the request fails (returns a non 200 status code), a `MythXAPIError` is raised.

Parameters

- **request_data** – The request data dictionary
- **auth_header** – The authorization header carrying the access token

Returns The raw response payload string

`pythx.api.handler.`**`print_request`**(*req: requests.models.PreparedRequest*) → str

Generate a pretty-printed HTTP request string.

Parameters **req** – The prepared requests HTTP request

Returns Pretty HTTP request string

`pythx.api.handler.`**`print_response`**(*res: requests.models.Response*) → str

Generate a pretty-printed HTTP response string.

Parameters **res** – The received requests HTTP response

Returns Pretty HTTP response string

Module contents

This package contans the API request handler and Client implementations.

pythx.middleware package

Submodules

pythx.middleware.base module

This module contains the abstract base middleware class.

class `pythx.middleware.base.`**BaseMiddleware**

Bases: `abc.ABC`

Abstract middleware class that can be used by developers to build their own.

A middleware is expected to expose two methods: `process_request` and `process_response`. Each is expected to return and updated version of their input. The return type must be the same as the input type.

As middlewares are processed sequentially, it is recommended that they are kept associative, meaning that the order in which middlewares are executed does not matter. In practice, this means that a middleware should not depend on the content of other middlewares, or return data that could break other middlewares that are executed after.

process_request(*req: [<class 'mythx_models.request.analysis_input.AnalysisInputRequest'>, <class 'mythx_models.request.analysis_list.AnalysisListRequest'>, <class 'mythx_models.request.analysis_status.AnalysisStatusRequest'>, <class 'mythx_models.request.analysis_submission.AnalysisSubmissionRequest'>, <class 'mythx_models.request.auth_login.AuthLoginRequest'>, <class 'mythx_models.request.auth_logout.AuthLogoutRequest'>, <class 'mythx_models.request.auth_refresh.AuthRefreshRequest'>, <class 'mythx_models.request.detected_issues.DetectedIssuesRequest'>, <class 'mythx_models.request.group_creation.GroupCreationRequest'>, <class 'mythx_models.request.group_list.GroupListRequest'>, <class 'mythx_models.request.group_operation.GroupOperationRequest'>, <class 'mythx_models.request.group_status.GroupStatusRequest'>, <class 'mythx_models.request.project_creation.ProjectCreationRequest'>, <class 'mythx_models.request.project_deletion.ProjectDeleteRequest'>, <class 'mythx_models.request.project_list.ProjectListRequest'>, <class 'mythx_models.request.project_status.ProjectStatusRequest'>, <class 'mythx_models.request.project_update.ProjectUpdateRequest'>, <class 'mythx_models.request.version.VersionRequest'>]*) → Dict[KT, VT]

Abstract method for a request processor.

The implementation is expected to return an updated version of the request data dictionary.

Parameters **req** – The request's data dictionary

process_response(*resp: Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]*) → Type[Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]]

Abstract method for a response processor.

The implementation is expected to return an updated version of the response domain model.

Parameters **resp** – The response domain model

pythx.middleware.toolname module

This module contains a middleware to fill the `clientToolName` field.

class pythx.middleware.toolname.**ClientToolNameMiddleware**(*name: str = 'pythx'*)

Bases: *pythx.middleware.base.BaseMiddleware*

This middleware fills the `clientToolName` field when submitting a new analysis job.

This means that only `process_request` carries business logic, while `process_response` returns the input response object right away without touching it.

process_request(*req: [<class 'mythx_models.request.analysis_input.AnalysisInputRequest'>, <class 'mythx_models.request.analysis_list.AnalysisListRequest'>, <class 'mythx_models.request.analysis_status.AnalysisStatusRequest'>, <class 'mythx_models.request.analysis_submission.AnalysisSubmissionRequest'>, <class 'mythx_models.request.auth_login.AuthLoginRequest'>, <class 'mythx_models.request.auth_logout.AuthLogoutRequest'>, <class 'mythx_models.request.auth_refresh.AuthRefreshRequest'>, <class 'mythx_models.request.detected_issues.DetectedIssuesRequest'>, <class 'mythx_models.request.group_creation.GroupCreationRequest'>, <class 'mythx_models.request.group_list.GroupListRequest'>, <class 'mythx_models.request.group_operation.GroupOperationRequest'>, <class 'mythx_models.request.group_status.GroupStatusRequest'>, <class 'mythx_models.request.project_creation.ProjectCreationRequest'>, <class 'mythx_models.request.project_deletion.ProjectDeleteRequest'>, <class 'mythx_models.request.project_list.ProjectListRequest'>, <class 'mythx_models.request.project_status.ProjectStatusRequest'>, <class 'mythx_models.request.project_update.ProjectUpdateRequest'>, <class 'mythx_models.request.version.VersionRequest'>]*) → [<class 'mythx_models.request.analysis_input.AnalysisInputRequest'>, <class 'mythx_models.request.analysis_list.AnalysisListRequest'>, <class 'mythx_models.request.analysis_status.AnalysisStatusRequest'>, <class 'mythx_models.request.analysis_submission.AnalysisSubmissionRequest'>, <class 'mythx_models.request.auth_login.AuthLoginRequest'>, <class 'mythx_models.request.auth_logout.AuthLogoutRequest'>, <class 'mythx_models.request.auth_refresh.AuthRefreshRequest'>, <class 'mythx_models.request.detected_issues.DetectedIssuesRequest'>, <class 'mythx_models.request.group_creation.GroupCreationRequest'>, <class 'mythx_models.request.group_list.GroupListRequest'>, <class 'mythx_models.request.group_operation.GroupOperationRequest'>, <class 'mythx_models.request.group_status.GroupStatusRequest'>, <class 'mythx_models.request.project_creation.ProjectCreationRequest'>, <class 'mythx_models.request.project_deletion.ProjectDeleteRequest'>, <class 'mythx_models.request.project_list.ProjectListRequest'>, <class 'mythx_models.request.project_status.ProjectStatusRequest'>, <class 'mythx_models.request.project_update.ProjectUpdateRequest'>, <class 'mythx_models.request.version.VersionRequest'>]

Add the `clientToolName` field if the request we are making is the submission of a new analysis job.

Because we execute the middleware on the request data dictionary, we cannot simply match the domain model type here. However, based on the endpoint and the request method we can determine that a new job has been submitted. In any other case, we return the request right away without touching it.

Parameters **req** – The request's data dictionary

Returns The request's data dictionary, optionally with the `clientToolName` field filled in

process_response(*resp: Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]*) → Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]

This method is irrelevant for adding our tool name data, so we don't do anything here.

We still have to define it, though. Otherwise when calling the abstract base class' `process_response` method, we will encounter an exception.

Parameters **resp** – The response domain model

Returns The very same response domain model

pythx.middleware.analysiscache module

This module contains a middleware to fill the `noCacheLookup` field.

class pythx.middleware.analysiscache.**AnalysisCacheMiddleware**(*no_cache: bool = False*)

Bases: *pythx.middleware.base.BaseMiddleware*

This middleware fills the `noCacheLookup` field when submitting a new analysis job.

This means that only `process_request` carries business logic, while `process_response` returns the

input response object right away without touching it.

process_request(*req: [<class 'mythx_models.request.analysis_input.AnalysisInputRequest'>, <class 'mythx_models.request.analysis_list.AnalysisListRequest'>, <class 'mythx_models.request.analysis_status.AnalysisStatusRequest'>, <class 'mythx_models.request.analysis_submission.AnalysisSubmissionRequest'>, <class 'mythx_models.request.auth_login.AuthLoginRequest'>, <class 'mythx_models.request.auth_logout.AuthLogoutRequest'>, <class 'mythx_models.request.auth_refresh.AuthRefreshRequest'>, <class 'mythx_models.request.detected_issues.DetectedIssuesRequest'>, <class 'mythx_models.request.group_creation.GroupCreationRequest'>, <class 'mythx_models.request.group_list.GroupListRequest'>, <class 'mythx_models.request.group_operation.GroupOperationRequest'>, <class 'mythx_models.request.group_status.GroupStatusRequest'>, <class 'mythx_models.request.project_creation.ProjectCreationRequest'>, <class 'mythx_models.request.project_deletion.ProjectDeleteRequest'>, <class 'mythx_models.request.project_list.ProjectListRequest'>, <class 'mythx_models.request.project_status.ProjectStatusRequest'>, <class 'mythx_models.request.project_update.ProjectUpdateRequest'>, <class 'mythx_models.request.version.VersionRequest'>]*) → [<class 'mythx_models.request.analysis_input.AnalysisInputRequest'>, <class 'mythx_models.request.analysis_list.AnalysisListRequest'>, <class 'mythx_models.request.analysis_status.AnalysisStatusRequest'>, <class 'mythx_models.request.analysis_submission.AnalysisSubmissionRequest'>, <class 'mythx_models.request.auth_login.AuthLoginRequest'>, <class 'mythx_models.request.auth_logout.AuthLogoutRequest'>, <class 'mythx_models.request.auth_refresh.AuthRefreshRequest'>, <class 'mythx_models.request.detected_issues.DetectedIssuesRequest'>, <class 'mythx_models.request.group_creation.GroupCreationRequest'>, <class 'mythx_models.request.group_list.GroupListRequest'>, <class 'mythx_models.request.group_operation.GroupOperationRequest'>, <class 'mythx_models.request.group_status.GroupStatusRequest'>, <class 'mythx_models.request.project_creation.ProjectCreationRequest'>, <class 'mythx_models.request.project_deletion.ProjectDeleteRequest'>, <class 'mythx_models.request.project_list.ProjectListRequest'>, <class 'mythx_models.request.project_status.ProjectStatusRequest'>, <class 'mythx_models.request.project_update.ProjectUpdateRequest'>, <class 'mythx_models.request.version.VersionRequest'>]

Add the `noCacheLookup` field if the request we are making is the submission of a new analysis job.

Because we execute the middleware on the request data dictionary, we cannot simply match the domain model type here. However, based on the endpoint and the request method we can determine that a new job has been submitted. In any other case, we return the request right away without touching it.

Parameters **req** – The request's data dictionary

Returns The request's data dictionary, with the `noCacheLookup` field filled in

process_response(*resp: Type[Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]]*) → Type[Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]]

This method is irrelevant for adding our tool name data, so we don't do anything here.

We still have to define it, though. Otherwise when calling the abstract base class' `process_response` method, we will encounter an exception.

Parameters **resp** – The response domain model

Returns The very same response domain model

pythx.middleware.group_data module

This module contains a middleware to fill the `groupId`/`groupName` field.

class pythx.middleware.group_data.**GroupDataMiddleware**(*group_id: str = None, group_name: str = None*)

Bases: *pythx.middleware.base.BaseMiddleware*

This middleware fills the `groupId` and `groupName` fields when submitting a new analysis job.

This means that only `process_request` carries business logic, while `process_response` returns the

input response object right away without touching it.

process_request(*req: [<class 'mythx_models.request.analysis_input.AnalysisInputRequest'>, <class 'mythx_models.request.analysis_list.AnalysisListRequest'>, <class 'mythx_models.request.analysis_status.AnalysisStatusRequest'>, <class 'mythx_models.request.analysis_submission.AnalysisSubmissionRequest'>, <class 'mythx_models.request.auth_login.AuthLoginRequest'>, <class 'mythx_models.request.auth_logout.AuthLogoutRequest'>, <class 'mythx_models.request.auth_refresh.AuthRefreshRequest'>, <class 'mythx_models.request.detected_issues.DetectedIssuesRequest'>, <class 'mythx_models.request.group_creation.GroupCreationRequest'>, <class 'mythx_models.request.group_list.GroupListRequest'>, <class 'mythx_models.request.group_operation.GroupOperationRequest'>, <class 'mythx_models.request.group_status.GroupStatusRequest'>, <class 'mythx_models.request.project_creation.ProjectCreationRequest'>, <class 'mythx_models.request.project_deletion.ProjectDeleteRequest'>, <class 'mythx_models.request.project_list.ProjectListRequest'>, <class 'mythx_models.request.project_status.ProjectStatusRequest'>, <class 'mythx_models.request.project_update.ProjectUpdateRequest'>, <class 'mythx_models.request.version.VersionRequest'>]*) → Dict[KT, VT]

Add the `groupId` and/or `groupName` field if the request we are making is the submission of a new analysis job.

Because we execute the middleware on the request data dictionary, we cannot simply match the domain model type here. However, based on the endpoint and the request method we can determine that a new job has been submitted. In any other case, we return the request right away without touching it.

Parameters **req** – The request's data dictionary

Returns The request's data dictionary, optionally with the group data field(s) filled in

process_response(*resp: Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]*) → Union[mythx_models.response.analysis_input.AnalysisInputResponse, mythx_models.response.analysis_list.AnalysisListResponse, mythx_models.response.analysis_status.AnalysisStatusResponse, mythx_models.response.analysis_submission.AnalysisSubmissionResponse, mythx_models.response.auth_login.AuthLoginResponse, mythx_models.response.auth_logout.AuthLogoutResponse, mythx_models.response.auth_refresh.AuthRefreshResponse, mythx_models.response.detected_issues.DetectedIssuesResponse, mythx_models.response.group_creation.GroupCreationResponse, mythx_models.response.group_list.GroupListResponse, mythx_models.response.group_operation.GroupOperationResponse, mythx_models.response.group_status.GroupStatusResponse, mythx_models.response.project_creation.ProjectCreationResponse, mythx_models.response.project_deletion.ProjectDeletionResponse, mythx_models.response.project_list.ProjectListResponse, mythx_models.response.project_status.ProjectStatusResponse, mythx_models.response.project_update.ProjectUpdateResponse, mythx_models.response.version.VersionResponse]

This method is irrelevant for adding our group data, so we don't do anything here.

We still have to define it, though. Otherwise when calling the abstract base class' `process_response` method, we will encounter an exception.

Parameters **resp** – The response domain model

Returns The very same response domain model

Module contents

This module contains pre-defined middlewares.

This also encompasses an abstract base middleware that developers can easily use to build their own and register it later in the APIHandler class.

3.1.2 Module contents

Top-level package for pythx.

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/dmuhs/pythx/issues.

If you are 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.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

PythX could always use more documentation, whether as part of the official PythX docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dmuhs/pythx/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 contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *pythx* for local development.

1. Fork the *pythx* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pythx.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pythx
$ cd pythx/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests:

```
$ flake8 pythx tests or make lint
$ python3 setup.py test or make test
```

   To get flake8, just pip install them into your virtualenv.

6. 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
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python, 3.5 and 3.6, and 3.7. Check https://travis-ci.org/dmuhs/pythx/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

History

5.1 1.6.1 [2020-06-16]

- Update `mythx-models` dependency to omit stale warning

5.2 1.6.0 [2020-06-16]

- Disable model json schema validation and tests
- Various dependency upgrades

5.3 1.5.7 [2020-04-27]

- Add property checking middleware and tests
- Various dependency upgrades

5.4 1.5.6 [2020-04-21]

- Various critical dependency upgrades
- Fix bug where tests failed due to a data mismatch

5.5 1.5.5 [2020-02-24]

- Allow direct injection of domain model into client analyze method

5.6 1.5.4 [2020-02-20]

- Fix various dependency conflicts
- Add better documentation to readme file
- Update middleware type hints and documentation
- Update API module type hints and documentation
- Various dependency upgrades

5.7 1.5.3 [2020-02-10]

- Handle ambiguous `MYTHX_API_URL` declarations
- Various dependency upgrades

5.8 1.5.2 [2020-01-30]

- Add additional analysis/group list filter parameters
- Various dependency upgrades

5.9 1.5.1 [2020-01-29]

- Remove trial user support as it has been dropped from the API

5.10 1.5.0 [2020-01-29]

- Add pypy3 tests in Travis CI
- Add additional query param support to analysis list handler
- Add various test suite improvements
- Drop support for Python 3.5 and add support for Python 3.8
- Remove stale parameter config options
- Remove stale configuration object support
- Improve PyPI trove classifiers
- Various dependency upgrades

5.11 1.4.1 [2019-11-19]

- Extend status code range for failure check in API handler

5.12 1.4.0 [2019-11-19]

- Fix bug where empty query parameters were sent to the API
- Introduce group ID/name middleware
- Add group status support in client
- Add group creation/sealing support in client
- Add group list support in client
- Fix dependency conflict between `mythx-cli` and `mythx-models`
- Various dependency upgrades

5.13 1.3.2 [2019-10-04]

- Update `mythx-models` to 1.4.1

5.14 1.3.1 [2019-10-04]

- Update pytest from 5.1.2 to 5.2.0
- Update `mythx-models` to 1.4.0

5.15 1.3.0 [2019-09-20]

- Remove the PythX CLI PoC
- Add PSA about deprecation and link to new `mythx-cli` repository

5.16 1.2.6 [2019-09-19]

- Update twine from 1.14.0 to 1.15.0
- Bump `mythx-models` to 1.3.5

5.17 1.2.5 [2019-09-15]

- Update twine from 1.13.0 to 1.14.0
- Clean up dependencies
- Bump `mythx-models` to 1.3.3

5.18 1.2.4 [2019-09-06]

- Bump `mythx-models` to 1.3.2
- Add support to fetch analysis result input by UUID

5.19 1.2.3 [2019-09-05]

- Add an auth check override to handle situations where only the access token is given

5.20 1.2.2 [2019-08-30]

- Update `mythx-models` to 1.3.1

5.21 1.2.1 [2019-08-29]

- Update `mythx-models` to 1.3.0

5.22 1.2.0 [2019-08-26]

- Add mythx-models integration

5.23 1.1.8 [2019-06-05]

- Add debug flag to CLI
- Add support for the `clientToolName` response field
- Add support for the new source list format validation
- Update the bumpversion expression to support black formatting

5.24 1.1.7 [2019-04-20]

- Add main docstring description

5.25 1.1.6 [2019-04-19]

- Add `mainSource` support to CLI
- Fix bug where submission object was malformed (“AST” -> “ast”)
- Upgrade pytest dependency

5.26 1.1.5 [2019-04-16]

- Add middleware to disable analysis cache
- Add CLI support to analyze compiled Truffle projects
- Fix bug where reports were not completely shown
- Update the authentication data format
- Add support for the mainSource field
- Add shortcut to inject middlewares in Client

5.27 1.1.4 [2019-03-28]

- Fix issue in schema detection
- Upgrade Sphinx dependency

5.28 1.1.3 [2019-03-25]

- Initial release!
- 100% branch coverage achieved
- 100% doc coverage achieved
- Examples provided in repo readme
- Automatic PyPI deployment on version tag change

CHAPTER 6

Credits

6.1 Development Lead

- Dominik Muhs <dominik.muhs@consensys.net>

6.2 Contributors

- Remi Pan
- Aleksandr Sobolev

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

R

S

U

V