pythx Documentation

Release 0.1.0

Dominik Muhs

Mar 25, 2019

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 3.5+.

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


# login as free trial user
c = Client(eth_address="0x0000000000000000000000000000000000000000", password="trial")

# 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 report:
    print(issue.swc_title or "Undefined", "-", issue.description_short)

# Output:
# Assert Violation - A reachable exception has been detected.
# Undefined - MythX API trial mode.
```

1.4 The PythX CLI

The PythX CLI aims to be a simple example implementation to show developers on a practical example how PythX can be used in action. It provides a simple (and pretty!) interface to list analyses, submit new ones, check the status of a job, and get report data on the found issues.

```
$ pythx
Usage: pythx [OPTIONS] COMMAND [ARGS]...

Options:
--help  Show this message and exit.

Commands:
check    Submit a new analysis job based on source code, byte code, or...
```

(continues on next page)

(continued from previous page)

```
login    Login to your MythX account
logout   Log out of your MythX account
openapi  Get the OpenAPI spec in HTML or YAML format
ps       Get a greppable overview of submitted analyses
refresh  Refresh your MythX API token
report   Check the detected issues of a finished analysis job
status   Get the status of an analysis by its UUID
top      Display the most recent analysis jobs and their status
version  Print version information of PythX and the API
```

By default, PythX comes with a pre-enabled trial user. To get started right away, simply login with the default values:

```
$ pythx login
Please enter your Ethereum address [0x0000000000000000000000000000000000000000]:
Please enter your MythX password [trial]:
Successfully logged in as 0x0000000000000000000000000000000000000000
```

If you already have an account on MythX, simply login with your Ethereum address and the API password you have set on the website.

Submit an Solidity source file for analysis:

```
$ pythx check -sf test.sol
Analysis submitted as job df137587-7fc1-466a-a4b2-d63392099682
```

Check the status of your analysis job:

```
$ pythx status df137587-7fc1-466a-a4b2-d63392099682

| uuid           | df137587-7fc1-466a-a4b2-d63392099682 |
|----------------|--------------------------------------|
| apiVersion     | v1.4.3                               |
| mythrilVersion | 0.20.0                               |
| maestroVersion | 1.2.3                                |
| harveyVersion  | 0.0.13                               |
| maruVersion    | 0.3.4                                |
| queueTime      | 0                                    |
| runTime        | 0                                    |
| status         | Finished                             |
| submittedAt    | 2019-03-05T10:24:05.071Z             |
| submittedBy    | 12345678901234567890123              |
```

Get the analysis report. Pinpointing the line and column locations is still a bit buggy, sorry. :)

```
$ pythx report df137587-7fc1-466a-a4b2-d63392099682
Report for Unknown
```

(continues on next page)

(continued from previous page)

```
│    Line │    Column │ SWC Title          │ Severity     │ Short Description                    ␣
↪   │

│       0 │         0 │ Reentrancy         │ High         │ persistent state read after␣
↪call   │
├──────────────────────────────────────────────────────────────────────────────────────
│       0 │         0 │ Reentrancy         │ High         │ persistent state write after␣
↪call │
├──────────────────────────────────────────────────────────────────────────────────────
│       0 │         0 │ Assert Violation │ Medium       │ assertion violation                  ␣
↪   │
```

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

class pythx.api.client.**Client**(*eth_address: str = None*, *password: str = None*, *access_token: str = None*, *refresh_token: str = None*, *handler: pythx.api.handler.APIHandler = None*, *staging: bool = False*)

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.

analysis_list(*date_from: datetime.datetime = None*, *date_to: datetime.datetime = None*, *offset: int = None*) → pythx.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)

Returns AnalysisListResponse

analysis_ready(*uuid: str*) → bool
Return a boolean whether the analysis job with the given UUID has finished processing.

Parameters **uuid** –

Returns bool

analyze(*contract_name: str = None*, *bytecode: str = None*, *source_map: str = None*, *deployed_bytecode: str = None*, *deployed_source_map: str = None*, *sources: Dict[str, Dict[str, str]] = None*, *source_list: List[str] = None*, *solc_version: str = None*, *analysis_mode: str = 'quick'*) → pythx.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** –
- **bytecode** –
- **source_map** –
- **deployed_bytecode** –
- **deployed_source_map** –
- **sources** –
- **source_list** –
- **solc_version** –
- **analysis_mode** –

Returns AnalysisSubmissionResponse

assert_authentication()
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

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

Returns AuthLoginResponse

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

Returns AuthLogoutResponse

openapi(*mode='yaml'*) → pythx.models.response.oas.OASResponse
Return the OpenAPI specification either in HTML or YAML.

Parameters **mode** – "yaml" or "html"

Returns OASResponse

refresh(*assert_authentication=True*) → pythx.models.response.auth_refresh.AuthRefreshResponse
Perform a JWT refresh on the API and return the response.

Parameters **assert_authentication** –

Returns AuthRefreshResponse

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

Parameters **uuid** –

Returns DetectedIssuesResponse

status(*uuid: str*) → pythx.models.response.analysis_status.AnalysisStatusResponse
Get the status of an analysis job based on its UUID.

Parameters **uuid** – The job's UUID

Returns AnalysisStatusResponse

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

Returns VersionResponse

pythx.api.handler module

class pythx.api.handler.**APIHandler**(*middlewares: List[pythx.middleware.base.BaseMiddleware] = None*, *staging: bool = False*)
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*)
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 (e.g. https://staging.api.mythx.io for the staging deployment), 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*)
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*)
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: str*, *model*)
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** – 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*)
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/6b9e4a52-f061-4960-8246-
↪e1560627336a/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 PythXAPIError 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*) → 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*)
Generate a pretty-printed HTTP response string.

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

Returns Pretty HTTP response string

Module contents

pythx.cli package

Submodules

pythx.cli.main module

Console script for PythX.

This script aims to be an example of how PythX can be used as a developer-friendly library around the MythX smart contract security analysis API.

pythx.cli.main.**compile_from_source**(*source_path: str*, *solc_path: str = None*)
A simple wrapper around solc to compile Solidity source code.

Parameters

- **source_path** – The source file's path
- **solc_path** – The path to the solc compiler

Returns The parsed solc compiler JSON output

pythx.cli.main.**get_source_location_by_offset**(*filename*, *offset*)
Retrieve the Solidity source file's location based on the source map offset.

Parameters

- **filename** – The Solidity file to analyze
- **offset** – The source map's offset

Returns The line and column number

pythx.cli.main.**parse_config**(*config_path*, *tokens_required=False*)
Recover the user configuration file.

This file holds their most recent access and refresh JWT tokens. It allows PythX to restore the user's login state across executions.

Parameters

- **config_path** – The configuration file's path

- **tokens_required** – Raise an error if the tokens are required but missing

Returns

pythx.cli.main.**ps_core**(*config*, *staging*, *number*)
A helper method to retrieve data from the analysis list endpoint.

This functionality is used in the `pythx ps`, as well as the `pythx top` subcommands.

Parameters

- **config** – The configuration file's path
- **staging** – Boolean to denote whether to use the MythX staging deployment
- **number** – The number of analyses to retrieve

Returns The API response as AnalysisList domain model

pythx.cli.main.**recover_client**(*config_path*, *staging=False*, *exit_on_missing=False*)
A simple helper method to recover a client instance based on a user config.

Parameters

- **config_path** – The configuration file's path
- **staging** – A boolean to denote whether to use staging or not
- **exit_on_missing** – Return if the file is missing

Returns

pythx.cli.main.**update_config**(*config_path*, *client*)
Update the user configuration file with the latest login data.

The stored data encompasses the API password, the user's Ethereum address, and the latest access and refresh tokens.

Parameters

- **config_path** – The configuration file's path
- **client** – The client instance to get the latest information from

Module contents

This module contains a simple CLI example built on PythX.

pythx.conf package

Submodules

pythx.conf.base module

This module contains the base PythX config instantiated in the module's __init__ file.

class pythx.conf.base.**PythXConfig**(**args*, ***kwargs*)
Bases: dict

The global configuration object.

The PythX global configuration is accessible by all PythX modules and allows each component to exhibit the same behaviour while keeping each module isolated. By default, the config object contains the relevant endpoints for the MythX API production and staging deployments to make it easier for developers to switch between the two.

The config object is a subclass of a standard Python dictionary, so it can be used just like that by developers.

Module contents

The PythX config module exposes the global configuration.

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*)
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*)
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='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*)
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*)
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

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.

pythx.models package

Subpackages

pythx.models.request package

Submodules

pythx.models.request.analysis_list module

This module contains the AnalysisListRequest domain model.

class pythx.models.request.analysis_list.**AnalysisListRequest**(*offset: int*, *date_from: datetime.datetime*, *date_to: datetime.datetime*)
Bases: *pythx.models.request.base.BaseRequest*

Perform an API request that lists the logged in user's past analyses.

endpoint
The API's analysis list endpoint.

Returns A string denoting the list endpoint without the host prefix

classmethod from_dict(*d: Dict[str, Any]*)
Create the request domain model from a dict.

This also validates the dict's schema and raises a `RequestValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

headers
Additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method
The HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters
Additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload
The request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

to_dict()
Serialize the request model to a Python dict.

Returns A dict holding the request model data

pythx.models.request.analysis_status module

This module contains the AnalysisStatusRequest domain model.

class pythx.models.request.analysis_status.**AnalysisStatusRequest**(*uuid: str*)
Bases: *pythx.models.request.base.BaseRequest*

Perform an API request that gets the status of a previously submitted analysis job.

endpoint
The API's analysis status endpoint.

Returns A string denoting the status endpoint without the host prefix

classmethod from_dict(*d*)
Create the request domain model from a dict.

This also validates the dict's schema and raises a `RequestValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

headers
Additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method
The HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters
Additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload
The request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

to_dict()
Serialize the request model to a Python dict.

Returns A dict holding the request model data

pythx.models.request.analysis_submission module

This module contains the AnalysisSubmissionRequest domain model.

class pythx.models.request.analysis_submission.**AnalysisSubmissionRequest**(*contract_name: str = None, bytecode: str = None, source_map: str = None, deployed_bytecode: str = None, deployed_source_map: str = None, sources: Dict[str, Dict[str, str]] = None, source_list: List[str] = None, solc_version: str = None, analysis_mode: str = 'quick'*)

Bases: *pythx.models.request.base.BaseRequest*

Perform an API analysis job submission as a logged in user.

endpoint
The API's analysis submission endpoint.

Returns A string denoting the submission endpoint without the host prefix

classmethod from_dict(*d: Dict[KT, VT]*)
Create the request domain model from a dict.

This also validates the dict's schema and raises a `RequestValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

headers
Additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method
The HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters
Additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload
The request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'anyOf': [{'required

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the request model to a Python dict.

Returns A dict holding the request model data

pythx.models.request.auth_login module

This module contains the AuthLoginRequest domain model.

class pythx.models.request.auth_login.**AuthLoginRequest**(*eth_address: str*, *password: str*)
Bases: *pythx.models.request.base.BaseRequest*

Perform an API request that performs a login action with Ethereum address and password.

endpoint
The API's login endpoint.

Returns A string denoting the login endpoint without the host prefix

classmethod from_dict(*d: Dict[str, str]*)
Create the request domain model from a dict.

This also validates the dict's schema and raises a `RequestValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

headers
Additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method
The HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters
Additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload
The request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'ethA

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the request model to a Python dict.

Returns A dict holding the request model data

pythx.models.request.auth_logout module

This module contains the AuthLogoutRequest domain model.

class pythx.models.request.auth_logout.**AuthLogoutRequest**(*global_: bool = False*)
Bases: *pythx.models.request.base.BaseRequest*

Perform an API request that logs out the current user.

endpoint
The API's logout endpoint.

Returns A string denoting the logout endpoint without the host prefix

classmethod from_dict(*d: Dict[KT, VT]*)
Create the request domain model from a dict.

This also validates the dict's schema and raises a `RequestValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

headers
Additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method
The HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters
Additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload
The request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'globa

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the request model to a Python dict.

Returns A dict holding the request model data

pythx.models.request.auth_refresh module

This module contains the AuthRefreshRequest domain model.

class pythx.models.request.auth_refresh.**AuthRefreshRequest**(*access_token: str*, *refresh_token: str*)
Bases: *pythx.models.request.base.BaseRequest*

Perform an API request that refreshes the logged-in user's access token.

endpoint
The API's auth refresh endpoint.

Returns A string denoting the refresh endpoint without the host prefix

classmethod from_dict(*d: Dict[KT, VT]*)
Create the request domain model from a dict.

This also validates the dict's schema and raises a `RequestValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

headers
Additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method
The HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters
Additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload
The request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'acce

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the request model to a Python dict.

Returns A dict holding the request model data

pythx.models.request.base module

This module contains the base request domain model.

class pythx.models.request.base.**BaseRequest**
Bases: abc.ABC

An abstract object describing requests to the MythX API.

classmethod from_dict(*d: dict*)
An abstract method to construct the given domain model from a Python dict instance.

Parameters **d** – The dict instance to deserialize

classmethod from_json(*json_str: str*)
Deserialize a given JSON string to the given domain model.

Internally, this method uses the `from_dict` method.

Parameters **json_str** – The JSON string to deserialize

Returns The concrete deserialized domain model instance

headers
An abstract property returning additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method
An abstract property returning the HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters
An abstract property returning additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload
An abstract property returning the request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

schema = None

to_dict()
An abstract method to serialize the current domain model instance to a Python dict.

Returns A Python dict instance holding the serialized domain model data

to_json()
Serialize the current domain model instance to a JSON string.

Internally, this method uses the `to_dict` method.

Returns The serialized domain model JSON string

classmethod validate(*candidate*)
Validate the object's data format.

This is done using a schema contained at the class level. If no schema is given, it is assumed that the request does not contain any meaningful data (e.g. a simple GET request without any parameters) and no validation is done.

If the schema validation fails, a `RequestValidationError` is raised.

If this method is called on a concrete object that does not contain a schema, `validate` will return right away and log a warning as this behaviour might not have been intended by a developer.

Parameters **candidate** – The candidate dict to check the schema against

Returns None

pythx.models.request.detected_issues module

This module contains the DetectedIssuesRequest domain model.

class pythx.models.request.detected_issues.**DetectedIssuesRequest**(*uuid: str*)

Bases: *pythx.models.request.analysis_status.AnalysisStatusRequest*

Perform an API request that lists the detected issues of a finished analysis job.

endpoint

The API's analysis issue report endpoint.

Returns A string denoting the issue report endpoint without the host prefix

classmethod from_dict(*d: Dict[KT, VT]*)

Create the request domain model from a dict.

This also validates the dict's schema and raises a `RequestValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

headers

Additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method

The HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters

Additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload

The request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'uuid

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()

Serialize the request model to a Python dict.

Returns A dict holding the request model data

pythx.models.request.oas module

This module contains the OASRequest domain model.

class pythx.models.request.oas.**OASRequest**(*mode='yaml'*)
Bases: *pythx.models.request.base.BaseRequest*

Perform an API request that gets the OpenAPI spec.

endpoint
The API's OpenAPI spec endpoint.

Returns A string denoting the OpenAPI endpoint without the host prefix

classmethod from_dict(*d*)
Create the request domain model from a dict.

This also validates the dict's schema and raises a `RequestValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

headers
Additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method
The HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters
Additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload
The request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

to_dict()
Serialize the request model to a Python dict.

Returns A dict holding the request model data

pythx.models.request.version module

This module contains the VersionRequest domain model.

class pythx.models.request.version.**VersionRequest**
Bases: *pythx.models.request.base.BaseRequest*

Perform an API request that fetches API version information.

endpoint
The API's version endpoint.

Returns A string denoting the version endpoint without the host prefix

classmethod from_dict(*d*)
Create the request domain model from a dict.

This also validates the dict's schema and raises a `RequestValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

headers
Additional request headers.

Returns A dict (str -> str) instance mapping header name to header content

method
The HTTP method to perform.

Returns The uppercase HTTP method, e.g. "POST"

parameters
Additional URL parameters

Returns A dict (str -> str) instance mapping parameter name to parameter content

payload
The request's payload data.

Returns A Python dict to be serialized into JSON format and submitted to the endpoint.

to_dict()
Serialize the request model to a Python dict.

Returns A dict holding the request model data

Module contents

This module contains the PythX request domain models

pythx.models.response package

Submodules

pythx.models.response.analysis module

This module contains domain models regarding analysis jobs

class pythx.models.response.analysis.**Analysis**(*uuid: str*, *api_version: str*, *mythril_version: str*, *maestro_version: str*, *harvey_version: str*, *maru_version: str*, *queue_time: int*, *status: pythx.models.response.analysis.AnalysisStatus*, *submitted_at: str*, *submitted_by: str*, *run_time: int = 0*, *error: str = None*)
Bases: *pythx.models.response.base.BaseResponse*

An object describing an analysis job.

Such a model was built, because many other API responses deliver the same data when it comes to analysis jobs. This makes the code more DRY, validation easier, and allows for recursive SerDe (e.g. mapping `from_dict` to a deserialized JSON list of job objects.

classmethod from_dict(*d*)
Create the response domain model from a dict.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

to_dict()
Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

class pythx.models.response.analysis.**AnalysisStatus**
Bases: `str`, `enum.Enum`

An Enum describing the status an analysis job can be in.

ERROR = 'Error'

FINISHED = 'Finished'

IN_PROGRESS = 'In Progress'

QUEUED = 'Queued'

pythx.models.response.analysis_list module

class pythx.models.response.analysis_list.**AnalysisListResponse**(*analyses: List[pythx.models.response.analysis.Ana*
total: int)
Bases: *pythx.models.response.base.BaseResponse*

The API response domain model for a list of analyses.

classmethod from_dict(*d: dict*)
Create the response domain model from a dict.

This also validates the dict's schema and raises a `ResponseValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'analy

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

classmethod validate(*candidate*)
Validate the response data structure and add an explicit type check.

Parameters **candidate** – The Python dict to validate

pythx.models.response.analysis_status module

class pythx.models.response.analysis_status.**AnalysisStatusResponse**(*analysis: pythx.models.response.analysis.An*

Bases: *pythx.models.response.analysis_submission.AnalysisSubmissionResponse*

The API response domain model for the status of a single analysis.

pythx.models.response.analysis_submission module

class pythx.models.response.analysis_submission.**AnalysisSubmissionResponse**(*analysis: pythx.models.response.*

Bases: *pythx.models.response.base.BaseResponse*

The API response domain model for a successful analysis job submision.

classmethod from_dict(*d*)
Create the response domain model from a dict.

This also validates the dict's schema and raises a `ResponseValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'apiV

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

pythx.models.response.auth_login module

class pythx.models.response.auth_login.**AuthLoginResponse**(*access_token: str, refresh_token: str*)

Bases: *pythx.models.response.base.BaseResponse*

The API response domain model for a login action.

classmethod from_dict(*d: Dict[KT, VT]*)
Create the response domain model from a dict.

This also validates the dict's schema and raises a `ResponseValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'acce

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

pythx.models.response.auth_logout module

class pythx.models.response.auth_logout.**AuthLogoutResponse**
Bases: *pythx.models.response.base.BaseResponse*

The API response domain model for a successful logout action.

classmethod from_dict(*d: Dict[KT, VT]*)
Create the response domain model from a dict.

This also validates the dict's schema and raises a `ResponseValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

to_dict()
Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

pythx.models.response.auth_refresh module

class pythx.models.response.auth_refresh.**AuthRefreshResponse**(*access_token: str*, *refresh_token: str*)
Bases: *pythx.models.response.base.BaseResponse*

The API response domain model for a successful authentication refresh.

classmethod from_dict(*d: Dict[KT, VT]*)
Create the response domain model from a dict.

This also validates the dict's schema and raises a `ResponseValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'acce

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

pythx.models.response.base module

This module contains the base response domain model.

class pythx.models.response.base.**BaseResponse**
Bases: `abc.ABC`

An abstract object describing responses from the MythX API.

classmethod from_dict(*d: dict*)
An abstract method to construct the given domain model from a Python dict instance.

Parameters **d** – The dict instance to deserialize

classmethod from_json(*json_str: str*)
Deserialize a given JSON string to the given domain model.

Internally, this method uses the `from_dict` method.

Parameters **json_str** – The JSON string to deserialize

Returns The concrete deserialized domain model instance

schema = None

to_dict()
An abstract method to serialize the current domain model instance to a Python dict.

Returns A Python dict instance holding the serialized domain model data

to_json()
Serialize the current domain model instance to a JSON string.

Internally, this method uses the `to_dict` method.

Returns The serialized domain model JSON string

classmethod validate(*candidate*)
Validate the object's data format.

This is done using a schema contained at the class level. If no schema is given, it is assumed that the request does not contain any meaningful data (e.g. an empty logout response) and no validation is done.

If the schema validation fails, a `RequestValidationError` is raised.

If this method is called on a concrete object that does not contain a schema, `validate` will return right away and log a warning as this behaviour might not have been intended by a developer.

Parameters **candidate** – The candidate dict to check the schema against

Returns None

pythx.models.response.detected_issues module

class pythx.models.response.detected_issues.**DetectedIssuesResponse**(*issues: List[pythx.models.response.issue.Is source_type: pythx.models.response.issue.Source source_format: pythx.models.response.issue.Source source_list: List[str], meta_data: Dict[str, Any]*)

Bases: *pythx.models.response.base.BaseResponse*

The API response domain model for a report of the detected issues.

classmethod from_dict(*d*)
Create the response domain model from a dict.

This also validates the dict's schema and raises a `ResponseValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'items': {'propertie

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()

Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

pythx.models.response.issue module

This module contains domain models regrading found issues.

class pythx.models.response.issue.**Issue**(*swc_id: str, swc_title: str, description_short: str, description_long: str, severity: pythx.models.response.issue.Severity, locations: List[pythx.models.response.issue.SourceLocation], extra: Dict[str, Any]*)

Bases: `object`

The API response domain model for a single issue object.

classmethod from_dict(*d*)

Create the response domain model from a dict.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

classmethod from_json(*json_data: str*)

Parameters **json_data** –

Returns

to_dict()

Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

to_json()

Serialize the model to JSON format.

Internally, this method is using the `to_dict` method.

Returns A JSON string holding the model's data

class pythx.models.response.issue.**Severity**

Bases: `str`, `enum.Enum`

An Enum holding the possible severities an issue can have.

HIGH = 'High'

LOW = 'Low'

MEDIUM = 'Medium'

NONE = 'None'

class pythx.models.response.issue.**SourceFormat**

Bases: `str`, `enum.Enum`

An Enum holding the possible source format values.

EVM_BYZANTIUM_BYTECODE = 'evm-byzantium-bytecode'

EWASM_RAW = 'ewasm-raw'

SOLC_AST_COMPACT_JSON = 'solc-ast-compact-json'

SOLC_AST_LEGACY_JSON = 'solc-ast-legacy-json'

TEXT = 'text'

class pythx.models.response.issue.**SourceLocation**(*source_map: str, source_type: pythx.models.response.issue.SourceType, source_format: pythx.models.response.issue.SourceFormat, source_list: List[str]*)

Bases: object

The domain model for a source location in a detected issue.

classmethod from_dict(*d*)

Create the response domain model from a dict.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from d filled in

to_dict()

Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

class pythx.models.response.issue.**SourceType**

Bases: str, enum.Enum

An Enum holding the possible source type values.

ETHEREUM_ADDRESS = 'ethereum-address'

RAW_BYTECODE = 'raw-bytecode'

SOLIDITY_CONTRACT = 'solidity-contract'

SOLIDITY_FILE = 'solidity-file'

pythx.models.response.oas module

class pythx.models.response.oas.**OASResponse**(*data: str*)

Bases: *pythx.models.response.base.BaseResponse*

The API response domain model for an OpenAPI request.

classmethod from_dict(*d: Dict[KT, VT]*)

Create the response domain model from a dict.

This also validates the dict's schema and raises a ResponseValidationError if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from d filled in

classmethod from_json(*json_str: str*)

Parameters **json_str** –

Returns

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'data

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

pythx.models.response.version module

class pythx.models.response.version.**VersionResponse**(*api_version: str*, *maru_version: str*, *mythril_version: str*, *maestro_version: str*, *harvey_version: str*, *hashed_version: str*)
Bases: *pythx.models.response.base.BaseResponse*

The API response domain model for an API version request.

classmethod from_dict(*d: Dict[KT, VT]*)
Create the response domain model from a dict.

This also validates the dict's schema and raises a `ResponseValidationError` if any required keys are missing or the data is malformed.

Parameters **d** – The dict to deserialize from

Returns The domain model with the data from `d` filled in

schema = {'$schema': 'http://json-schema.org/draft-07/schema#', 'properties': {'api'

sf = <_io.TextIOWrapper name='/home/docs/checkouts/readthedocs.org/user_builds/pythx/c

to_dict()
Serialize the reponse model to a Python dict.

Returns A dict holding the request model data

Module contents

This module contains the PythX response domain models

Submodules

pythx.models.exceptions module

This module contains exceptions raised by PythX.

exception pythx.models.exceptions.**PythXAPIError**
Bases: *pythx.models.exceptions.PythXBaseException*

An exception denoting an API-related error.

This is usually raised when the API takes too long to respond, or a response contains an HTTP status code that is not 200 OK. In this case, the exception is passed on to the developer. This should give them early warnings about malformed data on their side, or recover in case the API is not available or experiences some kind of error we cannot handle.

exception pythx.models.exceptions.**PythXBaseException**
Bases: Exception

A base exception describing PythX-related errors.

exception pythx.models.exceptions.**RequestValidationError**
Bases: *pythx.models.exceptions.PythXBaseException*

A validation exception for API requests.

This is usually raised when the validation of a request fails. Validation is executed during serialization of requests. Often these are raised, because required keys are not present, or the API request could not be validated using the given JSON schema spec.

exception pythx.models.exceptions.**ResponseValidationError**
Bases: *pythx.models.exceptions.PythXBaseException*

A validation exception for API responses.

This is usually raised when the validation of a response fails. Validation is executed during deserialization of responses. Often these are raised, because required keys are not present, or the API response could not be validated using the given JSON schema spec.

pythx.models.util module

This module contains various utility functions for PythX domain models.

pythx.models.util.**deserialize_api_timestamp**(*timestamp_str: str*)
Deserialize a JavaScript API timestand into Python datetime format.

Parameters **timestamp_str** – The JS timestamp, e.g. `2019-01-10T01:29:38.410Z`

Returns A Python datetime object

pythx.models.util.**dict_delete_none_fields**(*d: Dict[KT, VT]*)
Remove all keys that have "None" values in a dict.

Parameters **d** – The dictionary to sanitize

Returns The dict instance with all "None keys" removed

pythx.models.util.**resolve_schema**(*module_path*, *filename*)
Return a path leading to the internal JSON schema files used for validation.

Parameters

- **module_path** – The calling module's path (used as base path)
- **filename** – The JSON schema file's name

Returns The complete path leading to the schema file (e.g. to be consumed by `open()`)

pythx.models.util.**serialize_api_timestamp**(*ts_obj: datetime.datetime*)
Serialize a Python datetime object to its JS equivalent.

Parameters **ts_obj** – A Python datetime object

Returns The JS timestamp, e.g. `2019-01-10T01:29:38.410Z`

Module contents

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

CHAPTER 6

Credits

6.1 Development Lead

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

6.2 Contributors

- Remi Pan <wen.pan@consensys.net>

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V