SAMA API Client 2025

Author: Marco Caspers

Date: 2025-01-16

Version: 1.0

Contents

[Introduction 3](#_Toc188018506)

[REST API general information 4](#_Toc188018507)

[REST API Client core class 5](#_Toc188018508)

[SAMA API Client class 11](#_Toc188018509)

[Python Context Manager 13](#_Toc188018510)

[Type stubs 14](#_Toc188018511)

[RestApiClientCore type stub 15](#_Toc188018512)

[SamaApiClient type stub 17](#_Toc188018513)

[APIEndpoint class type stub 18](#_Toc188018514)

[VersionObject type stub 18](#_Toc188018515)

[LinkObject type stub 19](#_Toc188018516)

[DomainObject type stub 19](#_Toc188018517)

[Configuration 20](#_Toc188018518)

[Logging 22](#_Toc188018519)

[Conclusion 23](#_Toc188018520)

# Introduction

The SAMA API client is a REST API client.

REST stands for **Re**presentational **S**tate Transfer.

API stands for **A**pplication **P**rogramming **I**nterface.

REST is an architectural style for designing web API’s that enables communication between different software systems over the internet. It is an interface that allows two computer systems to exchange information securely using standard HTTP methods like GET, POST, PUT and DELETE.

Together these make up the standard CRUD Create (POST), Read (GET), update (PUT) and delete (DELETE). The data is typically exchanged in JSON or XML format.

Check Mk (SAMA) has such a REST API that we can use to configure it and retrieve data.

The reason for writing the SAMA API client (a Python library) is to provide an easy way to interact with this API.

In the design of this client, I made a split between what is specific to SAMA and what is specific to a REST API. The following chapters in this document will describe each part in more detail.

The SAMA API client is written in Python 3.10, and is compatible with Python 3.9.

NOTE: For easier readability of this document in Word, disable the squiggly lines.

# REST API general information

The REST API in its most basic form is a HTTP client that can query what is called API Endpoints.

These endpoints are URL’s which can be divided into several parts:

* Protocol (http/https)
* The server fqdn, like “www.westcon.com”.
* The API base URL, like “/api/v1”
* The endpoint “path”, like: “/version”

Data that is to be provided to the endpoint can be sent in several forms:

* As part of the URL for example for the endpoint “/users” you need to add “/<username>” to get the details for the user with that name.
* More elaborate data for instance when you call an endpoint for a create endpoint (POST) can be provided in the body of the request.
* URL parameters, in some REST API’s you need to provide the parameters for an endpoint as URL parameters, for example: “/users?username=<username>&filter=today”
* Headers, some REST API’s require specific data in the request header for example “If-Match: <GUID>” as an identifier for update/delete endpoints.

REST API’s also may require authentication, which can come in many different ways, commonly used methods are a username password combination or an access token (also called the API key).

The response of a REST API is usually a HTTP response code together with a JSON object, depending on the type of the response.

# REST API Client core class

The “RestApiClientCore” class implements the generic REST API functionality that is required to interact with a REST API.

Normally REST API Clients implement a separate function for each API endpoint.

This class however is unique in that it is completely API endpoint agnostic.

The authentication that is required is based on the Authentication http request header.

There are several ways to provide the credentials:

* A username and password in the initialization arguments.
* A username/password in a keyring.
* A username/password inside a .env file.
* A username/password inside OS environment variables.

The above list is in reverse order of precedence, i.e. the last item has the least precedence.

Init > keyring > .env file > OS env var.

Upon initialization the “RestApiClientCore” class requires the full REST API URL (without the endpoint of course) as an argument.

Code example:

from sama\_api\_client import RestApiClientCore

**client = RestApiClientCore(api\_url='https://your\_checkmk\_server/site\_id/check\_mk/api/1.0', username='your\_username', secret='your\_secret')**

API endpoints aren’t defined in the “RestApiClientCore” class.

They need to be added. There are three functions that can be used to add endpoints:

1). register\_endpoint

2). add\_endpoint

3). load\_endpoints

An endpoint is represented in the “ApiEndpoint” class. It has 4 parameters/properties:

* Name
* Method
* Path
* Headers

Code example:

from sama\_api\_client.object\_schemas import APIEndpoint

**api\_endpoint = APIEndpoint(name='Version', path='version', method='GET', headers={})**

The name is the identifier that is going to be used to identify and call the API endpoint.

The method is the HTTP method (GET, PUT, POST, DELETE) that is to be used for the endpoint.

The path is the endpoint specific part of the URL that is to be used to send the request.

The “headers” parameter is a dictionary (it can be empty) of optional HTTP headers that may be specific for the endpoint.

Note that “RestApiClientCore” has a set of default headers that can be augmented or overridden by these headers.

The “register\_endpoint” function requires the 4 input arguments for the “ApiEndpoint” class.

Code example:

from sama\_api\_client import RestApiClientCore

client = RestApiClientCore(api\_url='https://your\_checkmk\_server/site\_id/check\_mk/api/1.0')

client.register\_endpoint(name='Version', path='version', method='GET', headers={})

**client.register\_endpoint(name='Hosts', path='hosts', method='GET', headers={})**

The “add\_endpoint” function takes an “APIEndpoint” object as an argument.

Code example:

from sama\_api\_client import RestApiClientCore

from sama\_api\_client.object\_schemas import APIEndpoint

api\_endpoint = APIEndpoint(name='Version', path='version', method='GET', headers={})

client = RestApiClientCore(api\_url='https://your\_checkmk\_server/site\_id/check\_mk/api/1.0')

**client.add\_endpoint(api\_endpoint)**

**client.add\_endpoint(APIEndpoint(name=”Hosts”, path=”hosts”, method=”GET”, headers={}))**

The “load\_endpoints” function has an optional argument path. It loads a JSON file in

which the endpoints are defined as a list of “ApiEndpoint” entries.

This is the most convenient way to add multiple endpoints at once. It is also much more maintainable than other methods, as the only thing that needs to be changed if the API changes is this one JSON file.

Code example:

from sama\_api\_client import RestApiClientCore

client = RestApiClientCore(api\_url='https://your\_checkmk\_server/site\_id/check\_mk/api/1.0')

**if client.load\_endpoints('api\_endpoints.json') is False:**

**print('Could not load the API endpoints')**

**exit(1)**

To communicate over HTTP/S the “RestApiClientCore” class utilizes the standard

Python requests library.

The “connect” function will open a session and tries to connect to a pre-configured API endpoint.

Code example:

client = RestApiClientCore(api\_url='https://your\_checkmk\_server/site\_id/check\_mk/api/1.0')

**if client.connect() is False:**

**print('Could not authenticate with the Check Mk API')**

**exit(1)**

The property called “connect\_endpoint” is a string that holds this pre-configured API endpoint.

It contains the path to the endpoint and the rest of the URL is taken from the “api\_url” argument that is provided in the initialization.

This property is independent of any registered API endpoints, by default it is set to the “version” endpoint. The property default is defined in the configuration, and it can be overridden by assigning a new value once the class has been initialized.

The “close” function will close the open session if it exists.

Code example:

from sama\_api\_client import RestApiClientCore

client = RestApiClientCore(api\_url='https://your\_checkmk\_server/site\_id/check\_mk/api/1.0')

if client.connect() is False:

    print('Could not authenticate with the Check Mk API')

    exit(1)

# Do something with the client

client.close()

The “api\_request” function is the function that executes the endpoint with the provided data and retrieves the response from the REST API server.

This function takes the following arguments:

* endpoiont\_name
* no\_default\_headers flag
* data
* url\_data
* params
* headers
* log\_access flag
* debug flag

Code example:

from sama\_api\_client import RestApiClientCore

import json

client = RestApiClientCore(api\_url='https://your\_checkmk\_server/site\_id/check\_mk/api/1.0')

if client.connect() is False:

    print('Could not authenticate with the Check Mk API')

    exit(1)

**response = client.api\_request('Version')**

if response is None:

    print('Could not make the API request')

    exit(1)

print(response.json())

**response = client.api\_request(“ActivatePendingChanges”, data=json.dumps({“redirect”: “True”, “sites”: [“<target\_site>”], “force\_foreaign\_changes”: “True”}))**

if response is None:

    print('Could not make the API request')

    exit(1)

print(response.json())

“endpoint\_name” is the name that was used to create/register the endpoint.

The “no\_default\_headers” flag is a Boolean value. If this flag is set to true, the default headers are cleared for this API call.

The “data” argument is an optional (JSON) string that is provided as the body of the HTTP request to the endpoint. Usually this is required for create and update type endpoints.

The “url\_data” argument is an additional part of the URL that has to be provided to access the endpoint. An example of this would be providing the username as part of the URL to read the data of that user. The complete URL would look like this: “https://www.westcon.com/api/v1/users/<user name>”, where in this case the “<user name>” would be the “url\_data” part.

The “headers” argument is an optional dictionary containing key/value pairs with addition HTTP headers specific for this request, they can either augment the existing set, or override them when providing the “no\_default\_headers” flag.

The “log\_access” flag is a Boolean flag used to indicate if each access of the endpoint needs to be logged (by default only issues will be logged). If set to True it will log the access to the endpoint.

The “debug” flag is a Boolean flag used to indicate if the access to the endpoint is to be printed to the console (“sdt out”).

The output of this function is a standard “Response” object of the Python requests standard library.

# SAMA API Client class

Where the “RestApiClientCore” class implements all basic functionality to communicate with a REST API, the “SamaApiClient” class which is derived from that class provides all the functionality that is specific to SAMA (Check Mk).

Basically, it wraps around the core class and adds extra functions.

The initialization arguments are different from the core class.

The class initialization takes the “site\_id” and “server\_url” and a file path as arguments.

Code example:

from sama\_api\_client.api\_client import SamaApiClient

client: SamaApiClient = SamaApiClient(

        site\_id="test",

        server\_url="http://localhost",

        path="api\_endpoints.json",

)

The “site\_id” is the Check Mk site id (for example: “test”) of the site that needs to be queried.

The “server\_url” is the protocol and server FQDN or IP address of the server that the site is hosted on. (For example: “https://www.westcon.com”, or “http://127.0.0.1”).

The “path” is the full file path to the JSON file that contains the definitions of the endpoints.

The version API endpoint is specifically mapped to a function called “get\_version\_information”. The reason for this is that the version endpoint returns an entirely different set of data than any other Check Mk API endpoint.

The function doesn’t require any arguments, and it returns an optional “VersionObject” object. It will return a Python “None” value if unsuccessful.

Code example:

client: SamaApiClient = SamaApiClient(

    site\_id="test",

    server\_url="http://localhost",

    path="api\_endpoints.json",

)

sama\_version\_information = client.get\_version\_information()

print(sama\_version\_information)

¢The “request” function uses the same arguments as the “api\_request” function of the “RestApiClientCore” class, but it has one more argument called “etag”.

Code example:

client: SamaApiClient = SamaApiClient(

        site\_id="test",

        server\_url="http://localhost",

    )

    folder = client.request("GetFolder", url\_data="test\_folder")

    response = client.request("DeleteFolder", url\_data="test\_folder", etag=folder.Etag)

    print(response)

“etag” is an optional value for the “If-Match” HTTP header.

The value represents the id of a rule (or other object) in Check Mk.

It is required for update and delete API endpoints.

The “etag” is retrieved either by a call to a read (GET) or create (POST) API endpoint.

The return value of the “request” function is a so called “DomainObject” if the call was successful, otherwise a Python “None” object may be returned.

The “DomainObject” is an object that is returned by all API calls with the exception of the “Version” API endpoint which returns a “VersionObject”.

Note that the “DomainObject” that is returned by the “request” function will also have the “Etag” property, which corresponds to the “etag” header for those endpoints that will return it.

# Python Context Manager

To make it easy to use the “SamaApiClient” class (and/or the “RestApiClientCore” class) the class was implemented to be used with/as a Python context manager.

This means that you can use Python code that looks like this to use the class:

    with SamaApiClient(

        site\_id="test",

        server\_url="http://localhost",

        path="/endpoints/api\_endpoints.json",

    ) as client:

        if client is None:

            print("Could not create the client")

            exit(1)

        sama\_version\_information = client.get\_version\_information()

        all\_folders = client.request("GetAllFolders")

        if sama\_version\_information is None:

            print("Could not retrieve the version information")

            exit(1)

        if all\_folders is None:

            print("Could not retrieve the folders")

            exit(1)

        print(sama\_version\_information)

        print(all\_folders)

NOTE: This code is to demonstrate the syntax.

By using a context manager like this you don’t need to explicitly call the “connect” and “close” functions of the ‘SamaApiClient” class as this is done implicitly.

It makes shorter and more readable code and it is less error prone because the “connect” and “close” functions don’t need to be called prior and after one or more requests to API endpoints.

# Type stubs

In the following chapters show the type stubs for each class.

These show an overview of properties and methods that are available in those classes.

Methods starting with an underscore are “private” methods.

This means that they don’t need to be called by the user, they are internal functions used by other functions in the class and should not be used “outside” the class.

The Python PEP8 style guide states this convention: <https://peps.python.org/pep-0008/>

Methods starting and ending with a double underscore in the name are Python “magic” or “dunder” methods. These are special constructs used by Python itself, they don’t need to (nor should) be called directly, they are used implicitly.

Type stubs are used by type checkers like “MyPy” to verify if the correct value types are provided in for example the arguments of a method on the class.

They are also used by linters like “Pylance” to correctly infer which methods are provided by a class and provide syntax highlighting accordingly.

While it doesn’t add to the functionality of the code it is considered good practice to have type stubs. Also, it is an easier way to see what properties and methods a class provides compared to looking at the implementation of the class, especially if the class is of a larger size with many properties and methods/functions.

## RestApiClientCore type stub

class RestApiClientCore:

    api\_url: str

    user: Optional[str]

    secret: Optional[str]

    session: Session

    endpoints: list[APIEndpoint]

    logger: Logger

    headers: Dict[str, str]

    \_connect\_endpoint: str

    def \_\_init\_\_(

        self, api\_url: str, user: Optional[str] = None, secret: Optional[str] = None

    ) -> None: ...

    @property

    def connection\_endpoint(self) -> str: ...

    @connection\_endpoint.setter

    def connection\_endpoint(self, value: str) -> None: ...

    def \_init\_logger(self) -> Logger: ...

    def \_get\_env\_token(self) -> Optional[str]: ...

    def \_get\_keyring\_token(self) -> Optional[str]: ...

    def \_get\_token(self) -> Optional[str]: ...

    def \_build\_api\_call\_url(

        self,

        endpoint\_path: str,

        url\_data: Optional[str] = None,

        params: Optional[Dict[str, Any]] = None,

    ) -> str: ...

    def \_get\_endpoint(self, endpoint\_name: str) -> Optional[APIEndpoint]: ...

    def get\_version(self) -> str: ...

    def load\_endpoints(self, path: str = str(\_\_API\_DEFINITION\_PATH\_\_)) -> bool: ...

def register\_endpoint(

        self,

        endpoint\_name: str,

        endpoint\_path: str,

        method: str,

        endpoint\_headers: Optional[Dict[str, str]],

    ) -> None: ...

    def add\_endpoint(self, endpoint: APIEndpoint) -> None: ...

    def connect(self) -> bool: ...

    def close(self) -> None: ...

    def api\_request(

        self,

        endpoint\_name: str,

        no\_default\_headers: bool = False,

        data: Optional[str] = None,

        url\_data: Optional[str] = None,

        params: Optional[Dict[str, str]] = None,

        headers: Optional[Dict[str, str]] = None,

        log\_access: bool = False,

        debug: bool = False,

    ) -> Optional[Response]: ...

    def \_\_enter\_\_(self) -> Optional["RestApiClientCore"]: ...

    def \_\_exit\_\_(self, exc\_type, exc\_value, traceback) -> Literal[False]: ...

    def \_\_str\_\_(self) -> str: ...

    def \_\_repr\_\_(self) -> str: ...

## SamaApiClient type stub

@final

class SamaApiClient(RestApiClientCore):

    def \_\_init\_\_(

        self, site\_id: str, server\_url: str, path: str = str(\_\_API\_DEFINITION\_PATH\_\_)

    ) -> None: ...

    def \_build\_api\_url(self, server\_url: str, site\_id: str) -> str: ...

    def get\_version\_information(self) -> Optional[VersionObject]: ...

    def request(

        self,

        endpoint\_name: str,

        no\_default\_headers: bool = False,

        data: Optional[str] = None,

        url\_data: Optional[str] = None,

        params: Optional[Dict[str, str]] = None,

        headers: Optional[Dict[str, str]] = None,

        etag: Optional[str] = None,

        debug: bool = False,

        log\_access: bool = False,

    ) -> Optional[DomainObject]: ...

    def \_\_enter\_\_(self) -> Optional["SamaApiClient"]: ...

    def \_\_exit\_\_(self, exc\_type, exc\_value, traceback) -> Literal[False]: ...

    def \_\_str\_\_(self) -> str: ...

    def \_\_repr\_\_(self) -> str: ...

## APIEndpoint class type stub

class APIEndpoint:

    name: str

    path: str

    method: str

    headers: Optional[dict[str,str]]

    def \_\_init\_\_(

        self,

        name: str,

        path: str,

        method: str,

        headers: Optional[dict[str,str]] = None,

    ) -> None: ...

    def \_\_str\_\_(self) -> str: ...

    def \_\_repr\_\_(self) -> str: ...

## VersionObject type stub

class VersionObject(BaseModel):

    site: str

    group: str

    rest\_api: Dict[str,str]

    versions: Dict[str,str]

    edition: str

    demo: bool

## LinkObject type stub

class LinkObject(BaseModel):

    domainType: str

    rel: str

    href: str

    method: str

    type: str

## DomainObject type stub

class DomainObject(BaseModel):

    links: List[LinkObject]

    id: str

    instanceId: Optional[str]

    domainType: str

    title: Optional[str]

    members: Optional[Dict[str, Any]]

    value: Optional[List["DomainObject"]]

    values: Optional[List["DomainObject"]]

    extensions: Optional[Dict[str, Any]]

    resultType: Optional[str]

    result: Optional[List["DomainObject"]]

    ETag: Optional[str]

    @staticmethod

    def Status204() -> "DomainObject": ...

The static method Status204() returns a domain object that is used for indicating that the API call returned a 204 HTTP response code instead of a 200 HTTP response code.

A HTTP response code 200 means “OK”, and the response does contain data.

A HTTP response code 204 means “OK, no more data”. Which means that the request was performed correctly without errors, but no more data follows.

Because of this we need the option to return an “empty” domain object which is “not possible” (easily) with the initializer of this class.

# Configuration

The configuration.py module contains all the default constants that are used in the classes.

This was done to have a single point to find all configuration/default values together.

The default constants are all named with all caps.

You will find constants in the following categories:

Environment variables constants

These are the names of environment variables in the operating system that are queried by the core class to get user credentials, if they are defined.

ENV\_FILE\_NAME = “.env”, the name of an optional hidden file that can contain the environment variables if the user doesn’t want to set these in the profile or .bashrc.

ENV\_USER = “SAMA\_AUTOMATION\_USER”, the environment variable key to retrieve the username.

ENV\_SECRET = “SAMA\_AUTOMATION\_SECRET”, the environment variable key to retrieve the users’ password.

Keyring constants

These are the names of the operating system keyring variables to retrieve user credentials from the OS keyring, if they are defined.

KEYRING\_SERVICE = “sama\_api\_client”, the name of the keyring in which the information is stored.

KEYRING\_USER = “sama\_automation\_user”, the name of the key in the keyring under which the username is stored.

KEYRING\_SECRET = “sama\_automation\_secret”, the name of the key in the keyring under which the users’ password is stored.

Logger constants

These are the default settings for the logger that logs information, for example the log format and the name of the logger itself.

LOG\_FORMAT = "%(asctime)s %(name)s %(levelname)s %(message)s", the log format, providing the date time, the Python module name in which the error occurs or data is logged, the log level and the log message.

LOG\_LEVEL = INFO, the default log level that is set.

LOGGER\_NAME = "sama\_api\_client", the name of the logger.

API version endpoint, the default name/path of the endpoint that is used in the “connect\_endpoint” property of the SamaApiClient class.

API\_VERSION\_ENDPOINT = "version"

API Endpoint constants

The filename and path where to find the API definitions JSON file.

API\_ENDPOINT\_DEFINITION\_FILE = "api\_endpoints.json", the default name of the JSON file containing the definitions of the API endpoints.

API\_DEFINITION\_PATH = Path(\_\_file\_\_).parent / API\_ENDPOINT\_DEFINITION\_FILE

This is the path to the file for which the location in which the “configuration.py” module is stored in is used as the base.

Api base URL

The base URL of the Check Mk REST API.

API\_BASE\_URL = "/check\_mk/api/1.0"

The redacted string.

REDACTED = "<:This is redacted for security reasons:>"

Some information, like passwords, is sensitive information.

Whenever sensitive information is in risk of being exposed the information is replaced with this string. Of course, as with everything in Python it is always possible to override this, so it is no guarantee that the information can’t be exposed, but within the limitations that are available the information is protected.

# Logging

By default, the “RestApiClientCore” class provides logging to the terminal (“stdout”)

It is possible to add logging to a file by adding that to the basic configuration of the standard Python logging.

The following code shows an example on how to do this:

import logging

from sama\_api\_client import RestApiClientCore

logging.basicConfig(filename='sama\_api\_client.log', level=logging.INFO)

client = RestApiClientCore(api\_url='https://your\_checkmk\_server/site\_id/check\_mk/api/1.0', username='your\_username', secret='your\_secret')

if client.connect() is False:

    print('Could not authenticate with the Check Mk API')

    exit(1)

# Do something with the client

client.close()

# Conclusion

The “SamaApiClient” class provides an easy mechanism to access the Check Mk API endpoints returning Check Mk’s specific “DomainObject” and “VersionObject” responses.

The “RestApiClientCore” class provides an easy mechanism to access any REST API in an agnostic way.

Both classes are very light weight as it doesn’t have a method for each specific API endpoint.

Therefore, they are easy to maintain.

The implementation as context manager classes means that they can be used in a convenient and easy way that is less prone to errors than using them in the traditional manner where the “connect” and “close” need to be called explicitly.