client

?

moduleclient

Global Variables

- DEFAULT_API_VERSION
- GIZA_TOKEN_VARIABLE
- MODEL_URL_HEADER
- API_KEY_HEADER
-

?

classApiClient

Implementation of the API client to interact with core-services

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None

```

?

methodretrieve_api_key

```

Copy retrieve_api_key()→None

```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```

Copy retrieve_token( user: Optional[str]=None, password: Optional[str]=None, renew: bool=False )→None

```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew
- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

?

classUsersClient

Client to interact withusers endpoint.

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None

```

?

methodcreate

```

Copy create(user: UserCreate)→ UserResponse

```

Call the API to create a new user

Args:

- user
- (users.UserCreate): information used to create a new user
-

Returns:

- users.UserResponse
- : the created user information
-

?

methodcreate_api_key

```

Copy create_api_key()

```

Call the API to create a new API key

Returns:

- users.UserResponse

- : the created user information
-

?

methodme

```

Copy me()→ UserResponse

```

Retrieve information about the current user. Must have a valid token to perform the operation, enforced by@auth

Returns:

- users.UserResponse
- : User information from the server
-

?

methodrequest_reset_password_token

```

Copy request_reset_password_token(email:str)→ Msg

```

Sends a request to the server to generate a password reset token. The token is sent to the user's email.

Args:

- email
- (str): The email of the user who wants to reset their password.
-

Returns:

- Msg
- : A message indicating the success or failure of the request.
-

?

methodresend_email

```

Copy resend_email(email:str)→ Msg

```

Resend the verification email to the user.

Args:

- email
- (EmailStr): The email of the user who wants to resend the verification email.
-

Returns:

- Msg
- : A message indicating the success or failure of the request.
-

?

methodreset_password

```
Copy reset_password(token:str, new_password:str)→ Msg
```

Resets the user's password using the provided token and new password.

Args:

- token
- (str): The password reset token sent to the user's email.
- new_password
- (str): The new password the user wants to set.
-

Returns:

- Msg
- : A message indicating the success or failure of the password reset.
-

?

methodretrieve_api_key

```
Copy retrieve_api_key()→None
```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```
Copy retrieve_token( user: Optional[str]=None, password: Optional[str]=None, renew: bool=False )→None
```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew
- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

?

classEndpointsClient

Client to interact withendpoints endpoint.

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None

```

?

methodcreate

```

Copy create( model_id:int, version_id:int, endpoint_create: EndpointCreate, f: BufferedReader )→ Endpoint

```

Create a new deployment.

Args:

- endpoint_create
- : Endpoint information to create
-

Returns: The recently created deployment information

?

methoddelete

```

Copy delete(endpoint_id:int)→None

```

Delete an endpoint.

Args:

- endpoint_id
- : Endpoint identifier
-

?

methoddownload_proof

```

Copy download_proof(endpoint_id:int, proof_id:int)→bytes

```

Download a proof.

Args:

- proof_id
- : Proof identifier
-

Returns: The proof binary file

?

methodget

```

Copy get(endpoint_id:int)→ Endpoint

```

Get a deployment.

Args:

- endpoint_id
- : Endpoint identifier
-

Returns: The deployment information

?

methodget_proof

```

Copy get_proof(endpoint_id:int, proof_id:int)→ Proof

```

Return information about a specific proof.proof_if is the identifier of the proof that can be a integer or the request id.

Returns: A proof created by the user

?

methodlist

```

Copy list(params: Optional[Dict[str,str]]=None)→ EndpointsList

```

List endpoints.

Returns: A list of endpoints created by the user

?

methodlist_jobs

```

Copy list_jobs(endpoint_id:int)→ JobList

```

List proofs.

Returns: A list of proofs created by the user

?

methodlist_proofs

```
Copy list_proofs(endpoint_id:int)→ ProofList
```

List proofs.

Returns: A list of proofs created by the user

?

methodretrieve_api_key

```
Copy retrieve_api_key()→None
```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```
Copy retrieve_token( user: Optional[str]=None, password: Optional[str]=None, renew: bool=False )→None
```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew
- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

?

classEndpointsClient

Client to interact withendpoints endpoint.

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None

```

?

methodcreate

```

Copy create( model_id:int, version_id:int, endpoint_create: EndpointCreate, f: BufferedReader )→ Endpoint

```

Create a new deployment.

Args:

- endpoint_create
- : Endpoint information to create
-

Returns: The recently created deployment information

?

methoddelete

```

Copy delete(endpoint_id:int)→None

```

Delete an endpoint.

Args:

- endpoint_id
- : Endpoint identifier
-

?

methoddownload_proof

```

Copy download_proof(endpoint_id:int, proof_id:int)→bytes

```

Download a proof.

Args:

- proof_id
- : Proof identifier
-

Returns: The proof binary file

?

methodget

```
Copy get(endpoint_id:int)→ Endpoint
```

Get a deployment.

Args:

- endpoint_id
- : Endpoint identifier
-

Returns: The deployment information

?

methodget_proof

```
Copy get_proof(endpoint_id:int, proof_id:int)→ Proof
```

Return information about a specific proof.proof_if is the identifier of the proof that can be a integer or the request id.

Returns: A proof created by the user

?

methodlist

```
Copy list(params: Optional[Dict[str,str]]=None)→ EndpointsList
```

List endpoints.

Returns: A list of endpoints created by the user

?

methodlist_jobs

```
Copy list_jobs(endpoint_id:int)→ JobList
```

List proofs.

Returns: A list of proofs created by the user

?

methodlist_proofs

```
Copy list_proofs(endpoint_id:int)→ ProofList
```

List proofs.

Returns: A list of proofs created by the user

?

methodretrieve_api_key

```

Copy retrieve_api_key()→None

```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```

Copy retrieve_token( user: Optional[str]=None, password: Optional[str]=None, renew: bool=False )→None

```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew
- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

?

classTranspileClient

Client to interact withusers endpoint.

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None
```

```

?

methodretrieve_api_key

```

Copy retrieve_api_key()→None

```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
- 

Returns:

- str
- : the API key
- 

?

methodretrieve_token

```

Copy retrieve_token(user: Optional[str]=None, password: Optional[str]=None, renew: bool=False)→None

```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew
- : for renewal of the JWT token by user login.
- 

Raises:

- Exception
- : if token could not be retrieved in any way
- 

?

methodtranspile

```

Copy transpile(f:)→ Response

```

Make a call to the API transpile endpoint with the model as a file.

Args:
```

- f
- (BinaryIO): model to send for transpilation
-

Returns:

- Response
- : raw response from the server with the transpiled model as a zip
-

?

methodupdate_transpilation

```

Copy update_transpilation( model_id:int, version_id:int, f: )→None

```

Make a call to the API transpile endpoint with the model as a file.

Args:

- f
- (BinaryIO): model to send for transpilation
-

Returns:

- Response
- : raw response from the server with the transpiled model as a zip
-

?

classModelsClient

Client to interact withmodels endpoint.

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None

```

?

methodcreate

```

Copy create(model_create: ModelCreate)→ Model

```

Create a new model.

Args:

- model_create
- : Model information to create
-

Raises:

- Exception

- : if there is no upload Url
-

Returns:

- Tuple[Model, str]
- : the recently created model and a url, used to upload the model.
-

?

methodget

```

Copy get(model_id:int,**kwargs)→ Model

```

Make a call to the API to retrieve model information.

Args:

- model_id
- : Model identfier to retrieve information
-

Returns:

- Model
- : model entity with the retrieved information
-

?

methodget_by_name

```

Copy get_by_name(model_name:str,**kwargs)→ Optional[Model]

```

Make a call to the API to retrieve model information by its name.

Args:

- model_name
- : Model name to retrieve information
-

Returns:

- Model
- : model entity with the retrieved information
-

?

methodlist

```

Copy list(**kwargs)→ ModelList

```

List all the models related to the user.

Returns: A list of models created by the user

?

methodretrieve_api_key

```
Copy retrieve_api_key()→None
```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```
Copy retrieve_token( user: Optional[str]=None, password: Optional[str]=None, renew: bool=False )→None
```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew
- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

?

methodupdate

```
Copy update(model_id:int, model_update: ModelUpdate)→ Model
```

Update a model.

Args:

- model_id
- : Model identfier to retrieve information
- model_update

- : body to partially update the model
-

Returns:

- Model
- : the updated model
-

?

classJobsClient

Client to interact withjobs endpoint.

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None

```

?

methodcreate

```

Copy create( job_create: JobCreate, trace: BufferedReader, memory: Optional[BufferedReader]=None )→ Job

```

Create a new job.

Args:

- job_create
- : Job information to create
- f
- : filed to upload, a CASM json
-

Raises:

- Exception
- : if there is no upload Url
-

Returns:

- Tuple[Model, str]
- : the recently created model and a url, used to upload the model.
-

?

methodget

```

Copy get(job_id:int, params: Optional[dict[str,str]]=None)→ Job

```

Make a call to the API to retrieve job information.

Args:

- job_id
- : Job identfier to retrieve information
-

Returns:

- Job
- : job entity with the retrieved information
-

?

methodlist

```

Copy list()→ List[Job]

```

List jobs.

Returns: A list of jobs created by the user

?

methodretrieve_api_key

```

Copy retrieve_api_key()→None

```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```

Copy retrieve_token( user: Optional[str]=None, password: Optional[str]=None, renew: bool=False )→None

```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew

- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

?

classVersionJobsClient

Client to interact withjobs endpoint.

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None

```

?

methodcreate

```

Copy create( model_id:int, version_id:int, job_create: JobCreate, f: BufferedReader )→ Job

```

Create a new job.

Args:

- job_create
- : Job information to create
- f
- : filed to upload, a CASM json
-

Raises:

- Exception
- : if there is no upload Url
-

Returns:

- Tuple[Model, str]
- : the recently created model and a url, used to upload the model.
-

?

methodget

```

Copy get(model_id:int, version_id:int, job_id:int)→ Job

```

Make a call to the API to retrieve job information.

Args:

- job_id
- : Job identfier to retrieve information
-

Returns:

- Job
- : job entity with the retrieved information
-

?

methodlist

```
Copy list(model_id:int, version_id:int)→ List[Job]
```

List jobs.

Returns: A list of jobs created by the user

?

methodretrieve_api_key

```
Copy retrieve_api_key()→None
```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```
Copy retrieve_token( user: Optional[str]=None, password: Optional[str]=None, renew: bool=False )→None
```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew

- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

?

classProofsClient

Client to interact withproofs endpoint.

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None

```

?

methoddownload

```

Copy download(proof_id:int)→bytes

```

Download a proof.

Args:

- proof_id
- : Proof identifier
-

Returns: The proof binary file

?

methodget

```

Copy get(proof_id:int)→ Proof

```

Make a call to the API to retrieve proof information.

Args:

- proof_id
- : Proof identfier to retrieve information
-

Returns:

- Proof
- : proof entity with the desired information
-

?

methodget_by_job_id

```
Copy get_by_job_id(job_id:int)→ Proof
```

Make a call to the API to retrieve proof information based on the job id.

Args:

- job_id
- : Job identifier to query by.
-

Returns:

- Proof
- : proof entity with the desired information
-

?

methodlist

```
Copy list()→ List[Proof]
```

List all the proofs related to the user.

Returns: A list of proofs created by the user

?

methodretrieve_api_key

```
Copy retrieve_api_key()→None
```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```
Copy retrieve_token( user: Optional[str]=None, password: Optional[str]=None, renew: bool=False )→None
```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the

API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew
- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

?

classVersionsClient

Client to interact withversions endpoint.

?

method__init__

```
Copy init( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None
```

?

methodcreate

```
Copy create( model_id:int, version_create: VersionCreate, filename: Optional[str]=None )→ Tuple[Version,str]
```

Create a new version.

Args:

- model_id
- : Model identifier
- version_create
- : Version information to create
-

Returns: The recently created version information

?

methoddownload

```
Copy download(model_id:int, version_id:int, params: Dict)→ Dict[str,bytes]
```

Download a version.

Args:

- model_id
- : Model identifier
- version_id
- : Version identifier
- params
- : Additional parameters to pass to the request
-

Returns: The version binary file

?

methoddownload_original

```
Copy download_original(model_id:int, version_id:int)→bytes
```

Download the original version.

Args:

- model_id
- : Model identifier
- version_id
- : Version identifier
-

Returns: The version binary file

?

methodget

```
Copy get(model_id:int, version_id:int)→ Version
```

Get a version.

Args:

- model_id
- : Model identifier
- version_id
- : Version identifier
-

Returns: The version information

?

methodlist

```
Copy list(model_id:int)→ VersionList
```

List all the versions related to a model.

Args:

- model_id
- : Model identifier
-

Returns: A list of versions related to the model

?

methodretrieve_api_key

```

Copy retrieve_api_key()→None

```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```

Copy retrieve_token( user: Optional[str]=None, password: Optional[str]=None, renew: bool=False )→None

```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
- it will be used to retrieve a new token.
- renew
- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

?

methodupdate

```

Copy update(model_id:int, version_id:int, version_update: VersionUpdate)→ Version

```

Update a specific version.

Args:

- model_id
- : Model identifier
- version_id
- : Version identifier
- version_update
- : Version information to update
-

Returns: The updated version information

?

methodupload_cairo

```

Copy upload_cairo(model_id:int, version_id:int, file_path:str)→str

```

Get the Cairo model URL.

Args:

- model_id
- : Model identifier
- version_id
- : Version identifier
-

Returns: The Cairo model URL

?

classWorkspaceClient

Client to interact withworkspaces endpoint.

?

method__init__

```

Copy **init**( host:str, token: Optional[str]=None, api_key: Optional[str]=None, api_version: str='v1', verify: bool=True, debug: Optional[bool]=False )→None

```

?

methodcreate

```

Copy create()→ Workspace

```

Call the API to create a new workspace. If the workspace already exists it will return a 400.

Returns:

- Workspace
- : the created workspace information
-

?

methoddelete

```
```

Copy delete()→None

```

Call the API to delete the workspace. If the workspace does not exist it will return a 404.

Returns: None

?

methodget

```

Copy get()→ Workspace

```

Make a call to the API to retrieve workspace information. Only one should exist.

Returns:

- Workspace
- : workspace information
-

?

methodretrieve_api_key

```

Copy retrieve_api_key()→None

```

Retrieve the API key from the~/.giza/.api_key.json file.

Raises:

- Exception
- : if the file does not exist
-

Returns:

- str
- : the API key
-

?

methodretrieve_token

```

Copy retrieve_token(user: Optional[str]=None, password: Optional[str]=None, renew: bool=False)→None

```

Get the JWT token.

First, it will try to get it from GIZA_TOKEN. Second, from ~/.giza/.credentials.json. And finally it will try to retrieve it from the API login the user in.

Args:

- user
- : if provided it will be used to check against current credentials and if provided withpassword
- used to retrieve a new token.
- password
- : if provided withuser
```

- it will be used to retrieve a new token.
- renew
- : for renewal of the JWT token by user login.
-

Raises:

- Exception
- : if token could not be retrieved in any way
-

Previous cli Next commands

Last updated4 days ago