

Summer Student Programme

Unified Python Reva Client

Version 1.0

Written by

Rasmus Oscar Welander
(rasmus.oscar.welander@cern.ch)

Supervisors

Giuseppe Lo Presti (giuseppe.lopresti@cern.ch)
Diogo Castro (diogo.castro@cern.ch)

Contents

1	Introduction	1
2	Background	1
3	System Design	3
4	Testing and Verification	5
5	Documentation	6
6	Future Work	9
7	Conclusion	10

List of Tables

1	Test Coverage and Manual Testing for CS3Client	5
---	--	---

List of Figures

1	State before the project	2
2	Goal after finishing the project.	2
3	Future goal that the project enables.	2
4	The architecture and the public interface of the CS3Client.	4

1 Introduction

The **CS3Client** is a Python library developed in the **CERNBox team at CERN** as part of the **cs3org** (Cloud Storage Synchronization and Sharing Organization) initiative. The cs3org initiative aims to foster collaboration and standardization among cloud storage services for science, creating a global ecosystem of interoperable storage solutions. CS3Client is designed to interact with the CS3 APIs defined by the cs3org, facilitating seamless communication with cloud storage services that support these protocols [4, 10].

By providing a simple and user-friendly interface, CS3Client enables developers to integrate CS3 services into their applications and workflows efficiently. It abstracts the complexities of direct API interactions, allowing users to perform file management, data transfer, and other cloud-based operations without delving into the underlying protocols.

2 Background

In the era of big data and collaborative research, efficient and secure access to cloud storage services is crucial. The **CS3 APIs** are a set of protocols and interfaces designed to standardize interactions with cloud storage systems, particularly in scientific and research contexts [5]. However, directly interfacing with these protocols can be complex and time-consuming.

Reva is a middleware that acts as a gateway to multiple storage providers, implementing the CS3 APIs and providing a unified interface for clients. It facilitates communication between clients and various storage backends, enabling seamless integration and interoperability [5, 7].

Currently, wopiserver and cs3api4lab interact with the underlying storage systems differently [2, 1]:

- **wopiserver**, which integrates cloud storage with applications like Office Online Server or Collabora Online, currently uses the xrootd protocol to communicate directly with EOS, the underlying storage system. This approach ties the wopiserver to a specific storage system and protocol [2, 8, 11].
- **cs3api4lab**, a JupyterLab extension that allows users to access CS3 storage, communicates with reva through gRPC, a high-performance, open-source universal RPC framework [1, 9].

This project aims to consolidate the two storage clients (in the wopiserver and the cs3api4lab) into a single easy-to-use python library as depicted in Figure 2. An illustration of the system currently in place at CERN is illustrated in Figure 1 [2, 1].

Eventually, the goal is to standardize the communication by having both the wopiserver and cs3api4lab communicate with reva exclusively through gRPC by removing the xrootd protocol from the wopiserver, thereby decoupling it from the EOS storage system and allowing it to interact with multiple storage providers via reva, as illustrated in Figure 3 [2, 1, 7, 9, 8].

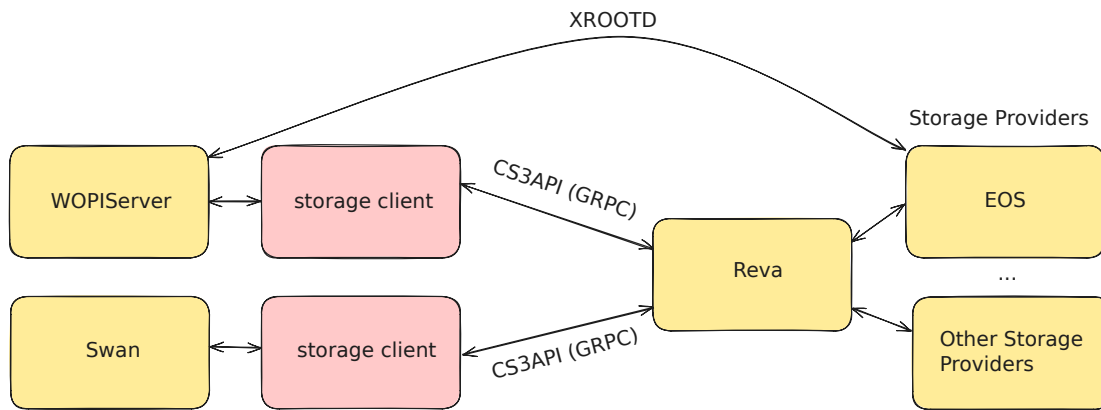


Figure 1: State before the project

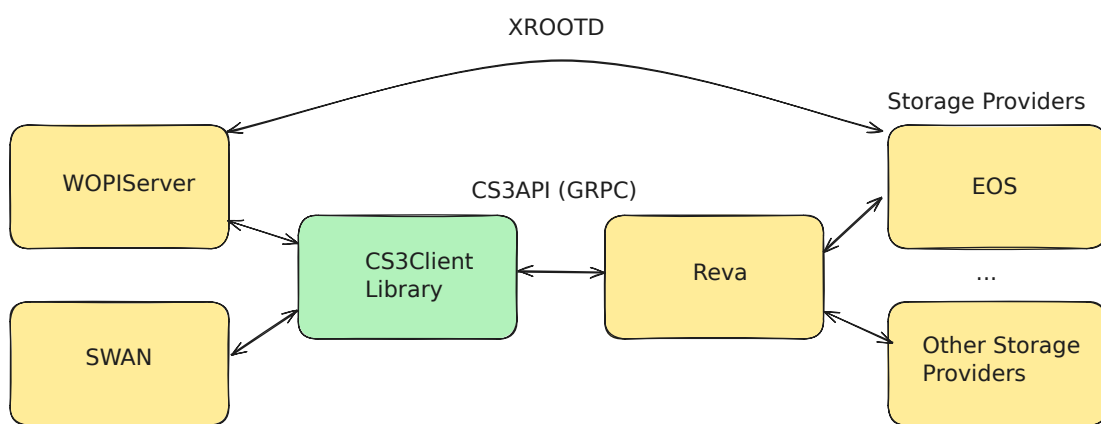


Figure 2: Goal after finishing the project.

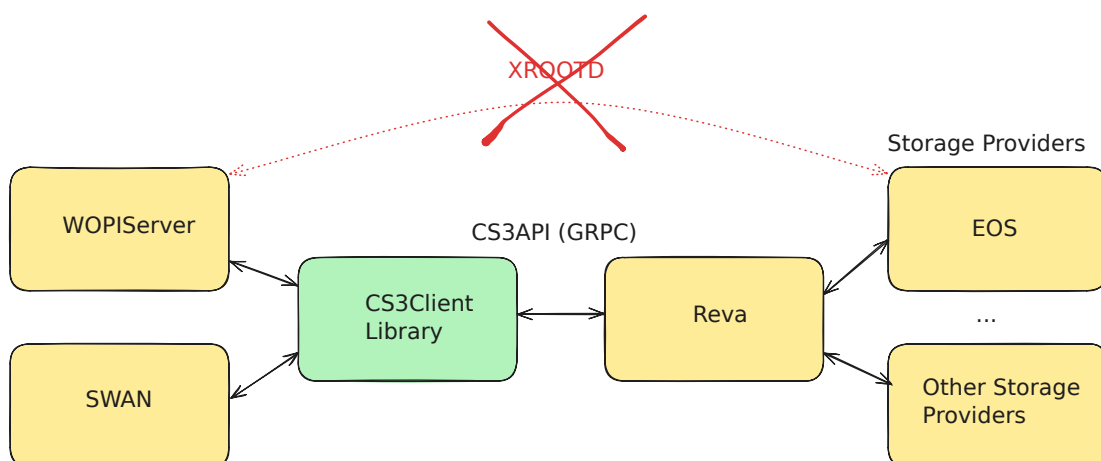


Figure 3: Future goal that the project enables.

3 System Design

CS3Client is designed with modularity and ease of use in mind. It leverages Python’s object-oriented programming features to encapsulate various functionalities into dedicated modules and classes [4]. The architecture and the public interface of the CS3Client can be seen in Figure 4 and the key components of the client include:

- **File Operations Module:** Handles common file operations such as reading, writing, deleting, renaming, and managing extended attributes.
- **Lock Management Module:** Provides mechanisms to set, get, refresh, and unlock files, ensuring coordinated access in collaborative environments.
- **Share Management Module:** Enables creation, updating, and deletion of shares, supporting user-to-user and public sharing mechanisms.
- **User Management Module:** Facilitates user information retrieval, searching for users, and managing user groups.
- **Checkpoint Module:** Supports restoring files through checkpoints and listing file versions, enhancing data recovery capabilities.
- **Application Integration Module:** Allows files to be opened in applications and lists available application providers, integrating with external tools.
- **Authentication and Authorization:** Handles authentication, supporting various login types and secure communication over SSL.
- **Logging and Error Handling:** Utilizes the provided logging object with preset levels to log messages (`'debug'` for detailed messages, `'info'` for essential information). Throws descriptive exceptions and logs events when errors are received from Reva.
- **Resource Module:** Manages CS3 resources using different file references—absolute paths, relative paths, or opaque file IDs—crucial for integrations like the wopiserver that handles file paths differently.
- **Configuration:** The CS3Client can be configured by passing an instance of `ConfigParser` when instantiating the client, for more information about the configuration see Section 5.

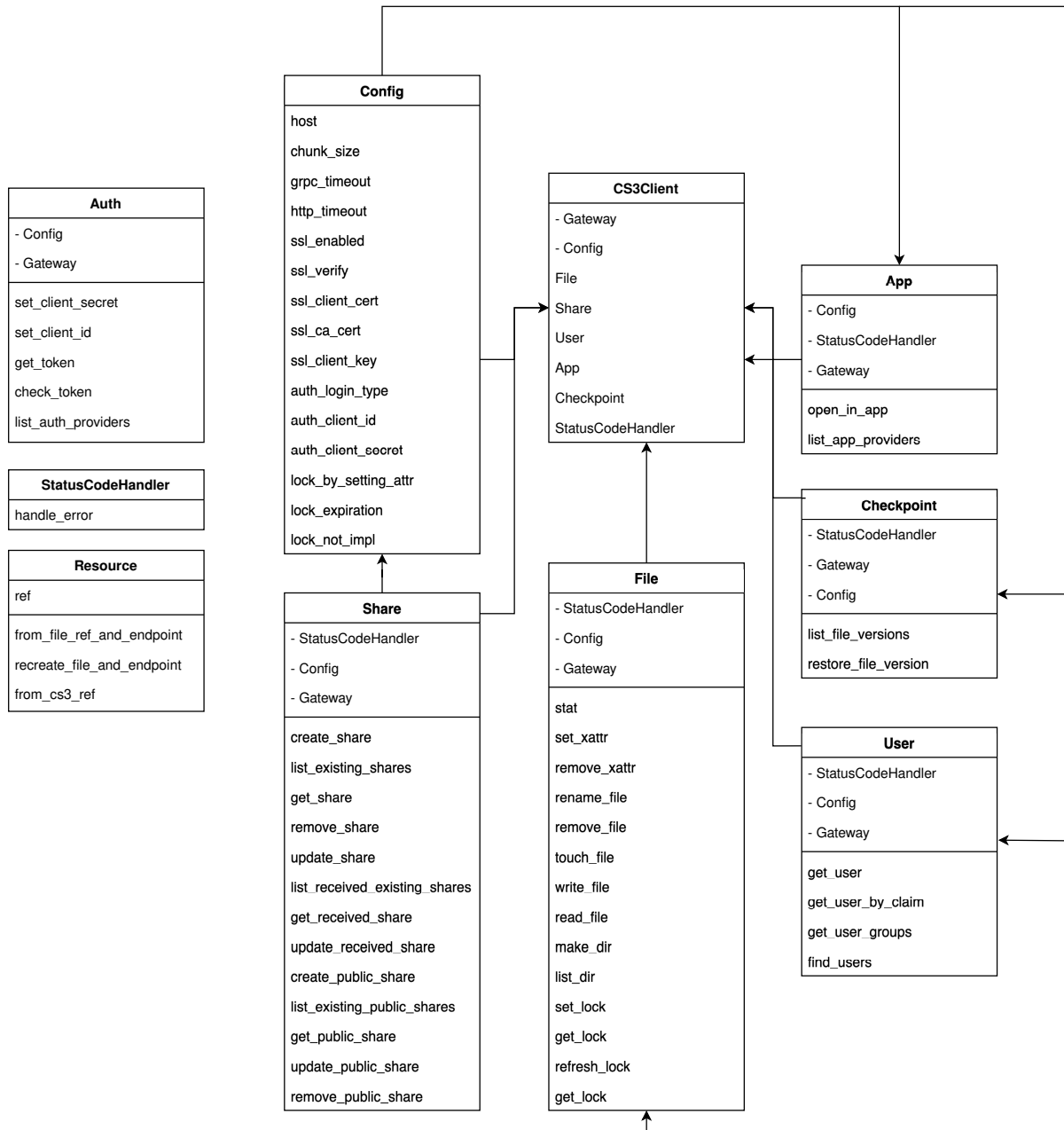


Figure 4: The architecture and the public interface of the CS3Client.

4 Testing and Verification

Continuous Integration (CI): The project utilizes GitHub Actions for continuous integration, implementing three dedicated CI pipelines to maintain high code quality and ensure functionality:

1. **Static Code Analysis:** This pipeline runs static code analysis tools to detect potential code issues.
2. **Building, Linting, and Unit Testing:** This pipeline builds the project, performs linting to ensure compliance with coding standards, and executes unit tests to verify that new changes do not break existing functionality.
3. **Publishing the pip Package:** This pipeline handles the packaging and publishing of the CS3Client to PyPI, making it available for installation via pip. This is triggered by tagging a commit with a version (e.g. `v.1.2.3`)

In table 1 the coverage of the unit tests can be seen, there are two outliers which are `file.py` and `auth.py` which have not been tested to the same extent as the others. This is because they were modified at the end of the project and there was no time to develop unit tests for the new additions.

Name	Statements	Missed	Coverage	Manual tests
app.py	30	0	100%	yes
auth.py	59	39	34%	yes
config.py	53	1	98%	yes
checkpoint.py	26	0	100%	yes
cs3client.py	34	3	91%	yes
cs3resource.py	80	4	95%	yes
exceptions.py	21	2	90%	yes
file.py	215	105	51%	yes
share.py	198	32	84%	yes
statuscodehandler.py	46	5	89%	yes
user.py	36	0	100%	yes
TOTAL	798	191	76%	-

Table 1: Test Coverage and Manual Testing for CS3Client

Manual Testing:

In addition to automated tests, manual testing was performed by interacting with real CS3-compliant cloud storage services. This ensures the client functions correctly in real-world scenarios and under various network conditions.

Integration Testing:

The client has been successfully integrated with the wopiserver, and integration tests have been conducted to ensure the wopiserver functions correctly when the storage client is switched to the CS3Client.

Integration with SWAN (via `cs3api4lab`) has not been undertaken, as it would require a significant amount of work on top of integrating this client to get it to a working state and that is outside of the scope of this project.

5 Documentation

The `CS3Client` includes comprehensive documentation that guides users through installation, configuration, and usage. The documentation is hosted on the project's GitHub repository and covers the following areas:

- **API Reference:** A complete reference of the available methods, parameters, and return values for interacting with CS3-compliant cloud storage services, this reference is generated by the user using Sphinx, details in the `README.md`.
- **Installation Guide:** Detailed steps for installing the client via `pip` and manually by cloning the GitHub repository [4].
- **Detailed comments in the code:** The code contains detailed comments on parameters for the public-facing interface along with type hints for all variables and parameters.
- **Examples:** Practical examples that show how to use the client for common operations such as uploading files, downloading files, sharing files, and so on. Some of these are shown in Listing 1, 3, and 2.
- **Configuration:** An example of a configuration and what the different parameters mean, an example configuration file can be found in Listing 4.

Listing 1: Authentication and Initialization

```
1 import logging
2 import configparser
3 from cs3client.cs3client import CS3Client
4 from cs3client.auth import Auth
5
6 config = configparser.ConfigParser()
7 with open("default.conf") as fdef:
8     config.read_file(fdef)
9 log = logging.getLogger(__name__)
10
11 client = CS3Client(config, "cs3client", log)
12 auth = Auth(client)
13 # Set the client id (can also be set in the config)
14 auth.set_client_id("<your_client_id_here>")
15 # Set client secret (can also be set in config)
16 auth.set_client_secret("<your_client_secret_here>")
17 # Checks if token is expired if not return ('x-access-token', <token>)
18 # if expired, request a new token from reva
19 auth_token = auth.get_token()
20
21 # OR if you already have a reva token
22 # Checks if token is expired if not return ('x-access-token', <token>)
```



```

23 # if expired, throws an AuthenticationException (so you can refresh your
    reva token)
24 token = "<your_reva_token>"
25 auth_token = Auth.check_token(token)

```

Listing 2: Lock Operations

```

1 # Set lock
2 client.file.set_lock(auth_token, resource, app_name="a", lock_id="some_lock
    ")
3
4 # Get lock
5 res = client.file.get_lock(auth_token, resource)
6 lock_id = res["lock_id"]
7
8 # Unlock
9 res = client.file.unlock(auth_token, resource, app_name="a", lock_id=
    lock_id)
10
11 # Refresh lock
12 client.file.set_lock(auth_token, resource, app_name="a", lock_id="some_lock
    ")
13 res = client.file.refresh_lock(
14     auth_token, resource, app_name="a", lock_id="new_lock",
15     existing_lock_id=lock_id
16 )
17 res = client.file.get_lock(auth_token, resource)

```

Listing 3: File operations

```

1 # mkdir
2 directory_resource = Resource(abs_path=f"/eos/user/r/rwelande/
    test_directory")
3 res = client.file.make_dir(auth.get_token(), directory_resource)
4
5 # touchfile
6 touch_resource = Resource(abs_path="/eos/user/r/rwelande/touch_file.txt")
7 res = client.file.touch_file(auth.get_token(), touch_resource)
8
9 # setxattr
10 resource = Resource(abs_path="/eos/user/r/rwelande/text_file.txt")
11 res = client.file.set_xattr(auth.get_token(), resource, "iop.wopi.
    lastwritetime", str(1720696124))
12
13 # rmxattr
14 res = client.file.remove_xattr(auth.get_token(), resource, "iop.wopi.
    lastwritetime")
15
16 # stat
17 res = client.file.stat(auth.get_token(), resource)
18
19 # removefile
20 res = client.file.remove_file(auth.get_token(), touch_resource)
21
22 # rename
23 rename_resource = Resource(abs_path="/eos/user/r/rwelande/rename_file.txt")
24 res = client.file.rename_file(auth.get_token(), resource, rename_resource)

```

```

25
26 # writefile
27 content = b"Hello World"
28 size = len(content)
29 res = client.file.write_file(auth.get_token(), rename_resource, content,
    size)
30
31 # listdir
32 list_directory_resource = Resource(abs_path="/eos/user/r/rwelande")
33 res = client.file.list_dir(auth.get_token(), list_directory_resource)
34
35
36 # readfile
37 file_res = client.file.read_file(auth.get_token(), rename_resource)

```

Listing 4: Configuration

```

1 [cs3client]
2
3 # Required
4 host = localhost:19000
5 # Optional, defaults to 4194304
6 chunk_size = 4194304
7 # Optional, defaults to 10
8 grpc_timeout = 10
9 # Optional, defaults to 10
10 http_timeout = 10
11
12 # Optional, defaults to True
13 tus_enabled = False
14
15 # Optional, defaults to True
16 ssl_enabled = False
17 # Optional, defaults to True
18 ssl_verify = False
19 # Optional, defaults to an empty string
20 ssl_client_cert = test_client_cert
21 # Optional, defaults to an empty string
22 ssl_client_key = test_client_key
23 # Optional, defaults to an empty string
24 ssl_ca_cert = test_ca_cert
25
26 # Optional, defaults to an empty string
27 auth_client_id = einstein
28 # Optional, defaults to basic
29 auth_login_type = basic
30 # Optional (Can also be set after instantiating the Auth object)
31 auth_client_secret = relativity
32
33 # Optional, defaults to False
34 # This configuration is used to enable/disable the fallback mechanism
35 # if the locks are not implemented in the storage provider
36 lock_by_setting_attr = False
37 # This configuration is used to enable/disable the fallback mechanism
38 # if the locks are not implemented in the storage provider
39 # Optional, defaults to False
40 lock_not_impl = False
41 # Optional, defaults to 1800

```

42 `lock_expiration = 1800`

6 Future Work

While CS3Client has greatly simplified the use of the `cs3apis`, several improvements could be made to enhance its functionality and continue with the integration:

1. **SWAN Integration:** A key future goal is to integrate CS3Client with SWAN, CERN's platform for web-based scientific analysis. This integration will streamline data management within research workflows, allowing seamless access and storage of large datasets across cloud platforms.
2. **Enhanced Testing and Exception Handling:** While the client has undergone rigorous testing, further unit tests should be developed for the `File` and `Auth` classes to get them up to the same coverage level as the other modules.
3. **Adding context managers:** To further follow the POSIX way of dealing with files (and thereby the standard way Python deals with files), the CS3Client should utilize context managers for operations such as reading files and writing files.
4. **Remove the need for a ConfigParser:** When configuring the CS3Client a `ConfigParser` instance has to be passed when instantiating the class. This could be simplified by passing a dictionary with the configuration, allowing the user more flexibility when configuring the client.
5. **Adding an OCM shares module:** Although not in use at CERN, adding a module for handling OCM shares could increase the adoption of the client.

7 Conclusion

CS3Client is a robust and user-friendly interface to the CS3 API, simplifying interactions with cloud storage services in scientific and research environments. Its modular design and comprehensive features make it a valuable tool for developers and researchers who need to manage files, shares, and user accounts across distributed storage systems. The client abstracts the complexities of direct API interactions, enabling users to focus on their core tasks without worrying about the underlying protocols.

By integrating with Reva through gRPC, CS3Client and associated projects like wopiserver and cs3api4lab can leverage a unified communication protocol that supports multiple storage providers. This enhances scalability, reduces dependencies on specific storage systems like EOS, and streamlines the overall architecture.

Through rigorous testing and thorough documentation, CS3Client ensures reliability and ease of adoption. It accelerates development workflows and enhances collaboration by providing essential cloud storage functionalities in an accessible manner.

References

- [1] <https://github.com/sciencemesh/cs3api4lab>
- [2] <https://github.com/cs3org/wopiserver>
- [3] <https://github.com/cs3org/>
- [4] <https://github.com/cs3org/cs3-python-client>
- [5] <https://github.com/cs3org/cs3apis>
- [6] <https://github.com/cs3org/reva>
- [7] <https://reva.link/>
- [8] <https://xrootd.slac.stanford.edu/index.html>
- [9] <https://grpc.io/>
- [10] <https://www.cs3community.org/>
- [11] <https://eos-web.web.cern.ch/eos-web/>