



SEVENTH FRAMEWORK PROGRAMME

CloudSpaces

(FP7-ICT-2011-8)

Open Service Platform for the Next Generation of Personal Clouds

D5.3 Final software release of the service platform

Due date of deliverable: 30-09-2015

Actual submission date: 06-10-2015

Start date of project: 01-10-2012

Duration: 36 months

Summary of the document

Document Type	Deliverable
Dissemination level	Public
State	Final
Number of pages	220
WP/Task related to this document	WP5
WP/Task responsible	NEC
Author(s)	Alberto Gómez (NEC), Raquel Sánchez (EOS), José Miguel García (TST)
Partner(s) Contributing	NEC, EOS, TST
Document ID	CLOUDSPACES_D5.3_151006_Public.pdf
Abstract	This document is a report on the software developed and final results obtained through the research carried out in the context of the Cloudspaces project.
Keywords	Cloud storage, interoperability, Personal Cloud, Storage API, traces

Table of Contents

1. Executive summary.....	6
2. NEC	7
2.1. Introduction.....	7
2.1.1 Product features	10
2.2. Storage API.....	13
2.2.1 Introduction.....	13
2.2.2 Authentication.....	14
2.2.3 Error handling	23
2.2.4 Folder operations	23
2.2.5 File operations.....	33
2.3. Interoperability protocol	43
2.3.1 Introduction.....	43
2.3.2 Prerequisites	43
2.3.3 Protocol foundations	44
2.3.4 Interoperability process.....	44
2.3.5 Sequence diagram	52
2.4. Traces	54
2.4.1 Introduction.....	54
2.4.2 Description of each field.....	55
2.4.3 WebDAV module	56
2.4.4 AdvancedFeatures module.....	57
2.4.5 Sample.....	57
2.4.6 Items information.....	57
2.4.7 Sharing information	58
3. eyeOS.....	59
3.1 Introduction	59

3.2 eyeOS integration with Personal Clouds	61
3.2.1 Integration	61
3.2.2 Authorization and authentication (log in/out)	61
3.2.3 Storage API.....	70
3.3 Personal Clouds and eyeOS interoperability	74
3.3.1 Implementation	74
3.3.2 Examples.....	76
3.4 Collaborative editing tool.....	83
3.5 Replacement U1DB to API.....	88
3.5.1 Implementation	88
3.5.2 Comments	90
3.5.3 Calendar	94
4. Tissat	108
4.1 Validation and Feedback analysis from open Internet trials	108
4.1.1 General description	108
4.1.2 Setting up a metering service for a storage system	108
4.2 Service Platform reference prototype	116
4.2.1 Migrate Keystone v2.0 to Keystone v3	116
4.2.2 Secure our StackSync platform with SSL	118
4.2.3 Development of group-based membership for StackSync users	118
4.2.4 Development of a group-based quota web application	121
4.2.5 Migrate web interface back-end from PHP to Python.....	125
4.2.6 Refactoring of StackSync web client.....	125
4.2.7 Implemented a folder sharing functionality for StackSync users	126
4.2.8 StackSync support for the ownCloud application	129
4.2.9 Development of IOS App	136
4.3 Adding Security Features to StackSync Components	137

4.3.1 Encryption. Android version.....	139
4.3.2 Encryption. StackSync Mobile App for IOS	141
4.3.3 Implementation of encryption. Web Client (Python)	141
4.3.4 Encryption. Admin Web Client (Python)	142
4.3.5 Encryption. StackSync Desktop Client.....	142
4.4 Interoperability between clouds. Integration with ownCloud	143
4.5 Traces Analysis	145
Annexes.....	148
Annex 1. Configuration file	148
Annex 2. Oauth Manager	149
Annex 3. Oauth API.....	150
Annex 4. Storage Manager.....	152
Annex 5. Storage API	186

1. Executive summary

In this document, we explain the contributions of NEC, eyeOS and Tissat to the services building block of CloudSpaces project.

It explains how it has done every development. We also provide empirical evaluations and demonstrations of our software contributions.

NEC explains:

1. Interoperability protocol: used to share files or folders between different personal clouds.
2. Storage API: used to access these resources
3. Installation of a traces management system to conduct a thorough analysis of them to see the behaviour of a real use of a personal cloud.

EyeOS, meanwhile, has integrated its remote virtual desktop with different personal clouds like StackSync and Cloud Storage. We can see the resources of any personal cloud and the interoperability among them.

Finally, Tissat has installed a traces management system that can monitor several parameters of use of the platform. Also, it has done several developments and new features in Stacksync like the new iOS client.

2. NEC

2.1. Introduction

NEC Cloud Storage is a carrier oriented online storage platform integrated with multiple fixed and mobile devices for ubiquitous information access.

With NEC Cloud Storage, companies no longer need to care where the data is physically located neither from which device it is accessed: all you need to know is what information you want and you get it transparently. Totally integrated with Windows folders (the new storage unit in the cloud appears as if local), and where Cloud Storage uses its small internal memory as a cache, so that the most common files are available even without coverage or access to the Internet. It's a unique virtual storage service (in the network) allows you to have a share space available with your employees, customers and providers, and in addition have your own private space to store your personal information. With the so called network disk you will have available a private storage space and a shared one. Everything you store will be automatically synchronized between your device and in the cloud. Thanks to the CloudSpaces project, Cloud Storage has been able to improve the product and make several features in which is able to interact with other Personal Clouds.

- Have a private and shared storage space. Everything you store there will be available on your local device (PC, Mobile, Tablet) and "the cloud" so you can access it from any computer connected to the Internet.
- All the information is point to point encrypted. If you lose your device no one will be able to access your files and you'll be able to restore the information within minutes from the cloud.
- Get access to Value Added Services such as Maillet, sharing with people inside and outside Cloud Storage, etc.
- Capability to share files between Personal Clouds

- Access to content among Personal Clouds through the Storage API explained in section 2.2. Cloud Storage will be able to access to the information of files and folders from StackSync and vice versa.
- Use as storage for third-party applications. eyeOS will be able to connect, browse among folders and perform any operation in Cloud Storage.



Main Benefits of NEC Cloud Storage:

- Mobile storage expanded to the Cloud
- Multi device Support: PC, Mobile (iOS, Android), Web.
- Secure Storage on PC, Mobile & Cloud
- Incremental revenues from additional space allocated on-the-flight
- Tight the users to the operator reducing churn
- Offline most used content availability (My favourites)
- Secure document sharing within a company or with external users
- Fully integrated with operators backend/billing systems
- Files seamlessly synchronized to the cloud
- Remotely administrated by the operator/company IT department
- Create groups of work (for departments, projects, etc...)
- Share files with your group or other groups
- Backup your files just by sending an e-mail to an e-mail account (mailet)

NEC Cloud Storage allows seamless and secure data synchronization among all end user devices such as smartphones, tablets, PCs and the Cloud. With NEC Cloud Storage, companies no longer need to care where the data is physically located neither from which device it is accessed: all you need to know is what information you want and you get it transparently.

Totally integrated with Windows folders (the new storage unit in the cloud appears as if local), and where Cloud Storage uses its small internal memory as a cache, so that the most common files are available even without coverage or access to the Internet.

It's a unique virtual storage service (in the network) allows you to have a share space available with your employees, customers and providers, and in addition have your own private space to store your personal information.

With the so called network disk you will have available a private storage space and a shared one. Everything you store will be automatically synchronized between your device and in the cloud.

Some features included are:

- Have a private and shared storage space. Everything you store there will be available on your local device (PC, Mobile, 3G modem) and "the cloud" so you can access it from any computer connected to the Internet.
- Get more benefits than with classical memories (memory stick) because the information is also stored in the cloud and you can recover in case of loss or theft
- All the information is point to point encrypted. If you lose your device no one will be able to access your files and you'll be able to restore the information within minutes from the cloud.

NEC Cloud Storage is not only a social network for the company to share ideas, but is a whole working environment to assign tasks, upload files, create wiki-like pages, manage incoming emails, track time, manage permissions, etc.

NEC Cloud Storage is available as a web application for all browsers and as mobile applications for iPhone, iPad and Android.

NEC Cloud Storage is considered extremely intuitive and easy to use by our customers. Setting up an account takes less than 30 seconds, no training needed.

The benefits for a company using NEC Cloud Storage are:

- Improved internal communication
- Effortless project management through progress updates
- More accountability for actions to be performed by team members
- Better tracking of information exchange with customers/providers
- Enriched knowledge sharing through Wiki-style pages and conversations
- Higher engagement of the team members with the company
- Peace of mind thanks to an strict international data privacy and security policy

2.1.1 Product features

2.1.1.1 File management

- User can upload, delete and modify files
- Multi device experience: from PC to Mobile Device
- Full integration with OS file explorer (Windows PC Client)
- Compatible with all applications (Windows PC Client)
- Files are automatically and seamlessly uploaded to the Cloud
- Local copy of files to support off-line functioning
- Automatic: Most used files, most recent files
- Direct sharing link: User can share files within a group or company or with external users
- Quota definition
- Multigroup: Groups creation
- Files preview for images

2.1.1.2 Multiple Download/Upload

Several files can be uploaded / downloaded by bulk file operation allowing windows like *Drag&Drop* Experience.

- Upload:
 - Possibility to select up to 1024 files at a single upload action not exceeding the total amount of free space left.
 - Drag&Drop up to 1024 files to the Drag&Drop upload space specified in the Folder structure environment.
 - While being uploaded each file will have a corresponding progress bar indicating the upload status.
 - All the files will be uploaded sequentially.
- Download:
 - Files are downloaded individually from the web interface.
 - Multiple files/folders can be selected from the PC client to move them to any other folder

In case the file size is higher than the allowed (block files bigger than 2 GB) the user will be notified on the attempt to synchronize/upload the file

Whenever the file is dragged and dropped in windows environment or a mobile device and exceed the limits, the write action to the protected cache partition will be blocked.

2.1.1.3 Maillet

NEC Cloud Storage incorporates the possibility to allow users to upload files to their cloud storage accounts through just sending this file as an attachment to a personal cloud storage mail account. The files sent to NEC Cloud Storage are stored by default in the root directory of the personal folder. The file limit size is limited to 20 MB on NEC Cloud Storage side but might be lower depending on user outgoing mail server configuration.

The user has the possibility to activate, deactivate or renew its NEC Cloud Storage mail account (by default disabled) in the personal account tab. If the mail account is updated the previous mail account is automatically deleted and won't be available.

2.1.1.4 Sharing folders

The user will be able to belong to different groups and be able to select any folder in the personal directory and invite users (internal and external) to access the files. The sharing capability has the following features:

- All the shared folder properties will be available through the Cloud Storage web interface.
- The owner of the shared folder can restrict invited users rights to read/write/download for different users.
- Accepting being part of a shared folder doesn't increments the storage occupied by the user as it only occupies the storage of the use who originally shared the file or folder.
- The owner (creator) of the shared folder can add or remove users from the folder as well as change their access rights.
- Any shared folder can receive comments and be protected with an access password.
- External (unregistered) users will be able to access the folders through a protected link, getting access to a folder navigation structure. Main folder action features will be kept except sharing.
- Editing: Depending on the rights assigned by the user who shared the file/folder, all users will be able to edit shared contents

2.1.1.5 Shared files editing

After a file or folder is shared, the user can review the shared file status and edit the sharing options. Sharing expiration date can be extended. Users in the sharing list can be added or removed.

2.1.1.6 Shared file URL

Every shared file or folder has an associated link. The link can be retrieved in the shared files list for external access to the shared information or distribution through alternative systems like email.

2.1.1.7 Individual sharing configuration (comments, Permissions)

While sharing a file or folder, the user can select individual permission per shared contact as well as write individual sharing messages to each participant. This allows enough flexibility to control profile access to the shared data as well as the possibility to set individual tasks.

2.1.1.8 File expiration configuration

While sharing a file or folder, the user can set up the time frame for shared link expiration. Available time slots range from 1 hour to no expiration at all. After a file or folder is shared, the user can update the expiration period at any time. If the link is expired the user can reactivate it by using the sharing properties menu.

2.1.1.9 User groups (Multigroup)

It is possible to create multiple groups of users for the purpose of sharing files within these groups. This could be useful to share file for a specific project or for a department. It is possible to configure the assigned quota of these groups as well as manage the groups defining which user is included or not at any time. It is also possible to watch the use of the group.

2.2. Storage API

2.2.1 Introduction

Storage API has been created in order to gain access to the information of the resources of Cloud Storage from another personal cloud.

In this case, once shared via interoperability protocol, StackSync, for instance, will be able to order information about a specific file.

Following, we are going to explain how to access these data and the operations can be done.

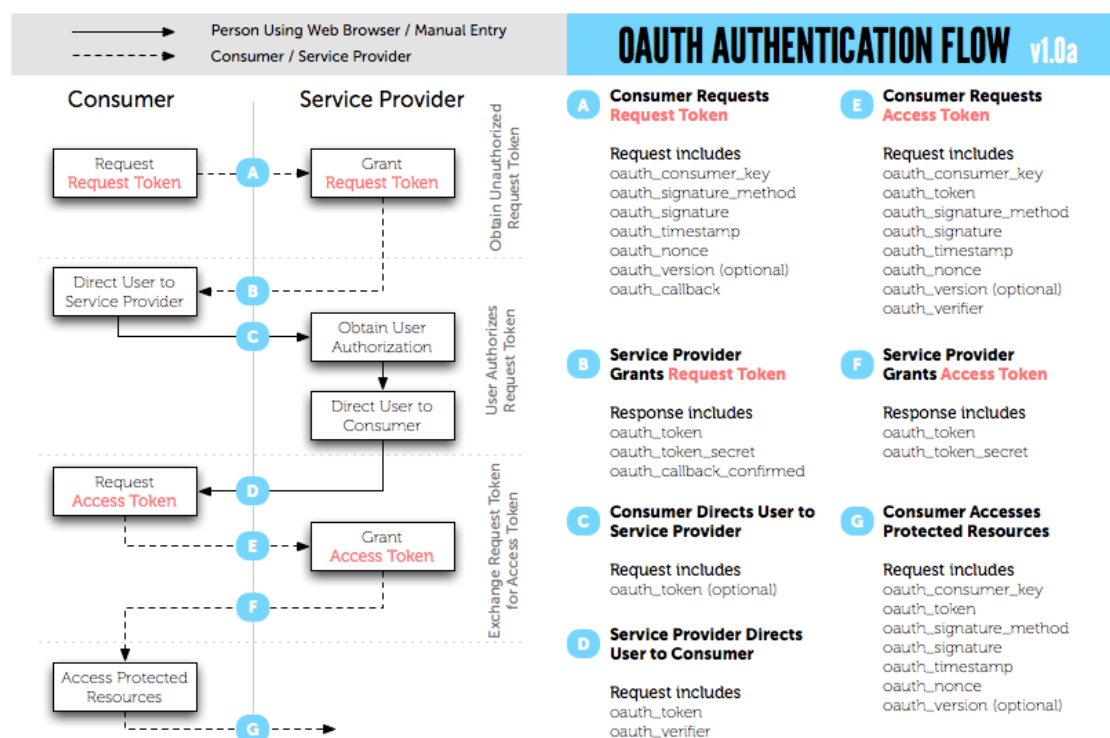
2.2.2 Authentication

Before calling any operation, all requests must be signed. We use the OAuth protocol which explain below.

2.2.2.1 Brief description OAuth

The OAuth protocol enables websites or applications (Consumers) to access Protected Resources from a web service (Service Provider) via an API, without requiring Users to disclose their Service Provider credentials to the Consumers. More generally, OAuth creates a freely-implementable and generic methodology for API authentication.

The Cloud Storage API uses OAuth 1.0a to authenticate the user and enable him to authorize the client application to access his data.



2.2.2.2 Definitions

Field	Description
Service provider	A web application that allows access via Auth.
User	An individual who has an account with the Service Provider.
Consumer	A website or application that uses OAuth to access the Service Provider on behalf of the User.
Protected resource(s)	Data controlled by the Service Provider, which the Consumer can access through authentication.
Consumer developer	An individual or organization that implements a Consumer.
Consumer key	A value used by the Consumer to identify itself to the Service Provider.
Consumer secret	A secret used by the Consumer to establish ownership of the Consumer Key.
Request token	A value used by the Consumer to obtain authorization from the User, and exchanged for an Access Token.
Access token	A value used by the Consumer to gain access to the Protected Resources on behalf of the User, instead of using the User's Service Provider credentials.
Token secret	A secret used by the Consumer to establish ownership of a given Token.
OAuth protocol parameters	Parameters with names beginning with oauth_.

2.2.2.3 Signed request

In this section we will look at how the signature process is handled by Cloud Storage and how each parameter is used with references to flows.

Cloud Storage takes all the information it has gathered and generated and places it in a single location. There are two ways of transporting this information, through the OAuth header or Query string. We highly recommend going the header route for better security.

Before we can generate this string we must gather all the required parameters and their values, some of these are used inside of the string directly and others indirectly through the encryption or encoding of the signature.

Signature Base String

Gathering the Method of the request, the URL of the request and the Parameters joined together by the ‘&’ symbol would look like this:

Considering:

Method: GET

URL: http://photos.example.net/photos

Parameters:

file=testfile.pdf&oauth_consumer_key=0Fkc84aAH3vP07&oauth_nonce=9e287ba35fe96f24&oauth_signature_method=HMAC-SHA1&oauth_timestamp=1191242096&oauth_token=f8a025ba36023&oauth_version=1.0&size=original

The Signature Base String would be:

GET&http%3A%2F%2Fphotos.example.net%2Fphotos&file%3Dtestfile.pdf%26oauth_consumer_key%0Fkc84aAH3vP07%26oauth_nonce%9e287ba35fe96f24%26oauth_signature_method%3DHMAC-SHA1%26oauth_timestamp%3D1191242096%26oauth_token%f8a025ba36023%26oauth_version%3D1.0%26size%3Doriginal

Encoding the Signature

There are three ways we can do this. PLAINTEXT, HMAC, and RSA Encryption. Each method is slightly different, and carries own set of pros and cons.

We have used HMAC method. This encoding method outputs our key into binary which we update our base with, which after this step gets Base64 encoded into it is final signature string:

OAuth Header

The OAuth header is a part of the signed request, it contains the `oauth_signature` and `oauth_signature_method` parameters and their values. It is a single string and separated generally by a comma (spaces are supported here by some services, stick to comma by default unless told otherwise by the service) and named Authorization with OAuth being the Bearer, in other flows this may change such as the Mac Bearer and other similar methods.

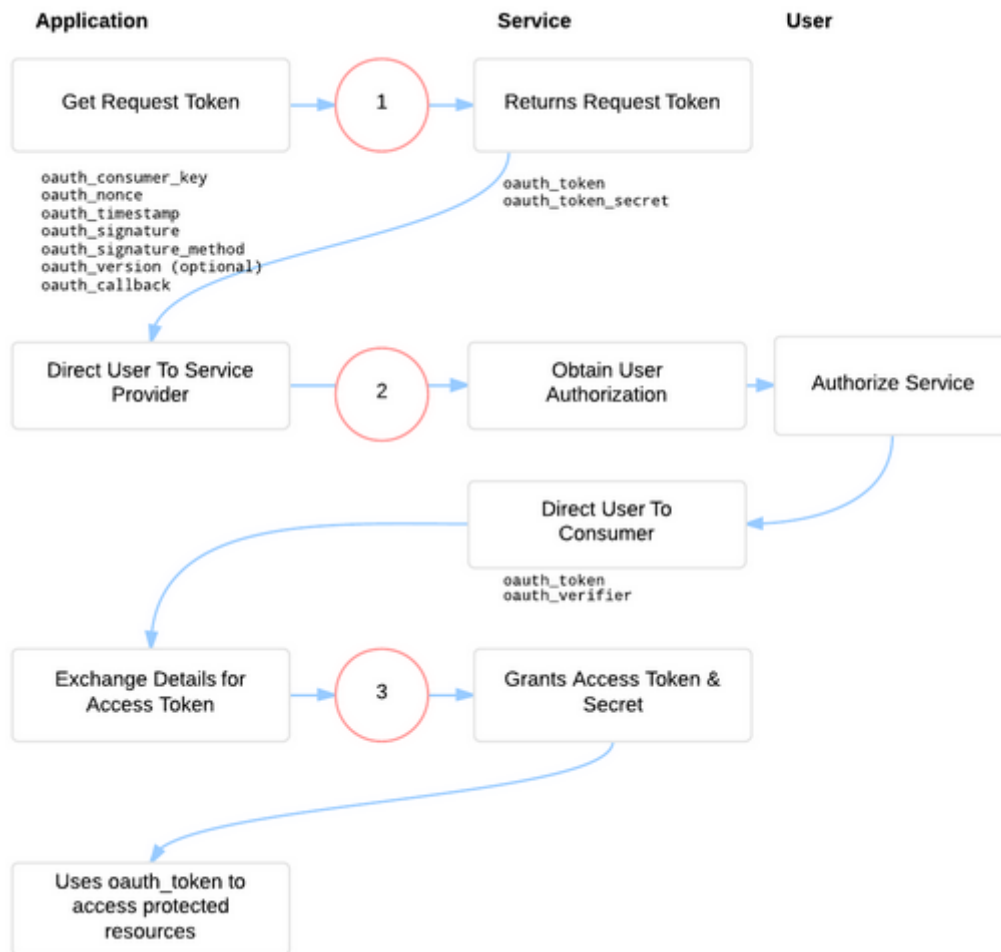
The header itself is built up by all the `oauth_*` parameters sorted (by name, then some more complex things). Here is an example for getting a Request Token:

```
POST /oauth/request_token HTTP/1.1
User-Agent: xxxxxx
Host: www.neccloudstorage.com
Accept: */*
Authorization:
    OAuth oauth_callback="http%3A%2F%2Flocalhost%2Fsign-in-with-
cloudstorage%2F",
    oauth_consumer_key="cChZNFj6T5R0TigYB9yd1w",
    oauth_nonce="ea9ec8429b68d6b77cd5600adbba0456",
    oauth_signature="F1Li3tvehgcraF8DMJ7OyxO4w9Y%3D",
    oauth_signature_method="HMAC-SHA1",
    oauth_timestamp="1318467427",
    oauth_version="1.0"
```

The `oauth_callback` is what Cloud Storage will invoke or respond to when the authentication step happens.

2.2.2.4 OAuth 1.0a (Three legged)

The diagram below shows the complete flow of an OAuth request:



Here is a brief explanation of the steps to follow:

1. Application sends a **signed** request for a Request Token:
 - `oauth_consumer_key`
 - `oauth_timestamp`
 - `oauth_nonce`
 - `oauth_signature`
 - `oauth_signature_method`
 - `oauth_version` *Optional*
 - `oauth_callback`
2. Grants application Request Token:
 - `oauth_token`
 - `oauth_token_secret`
 - `oauth_callback_confirmed`
 - ... Additional Parameters / Arguments
3. Send user to authorize url using:

- `oauth_token`
- 4. Prompts user to authorize / grant access
- 5. User grants access
- 6. Directs back to application with:
 - `oauth_token`
 - `oauth_verifier`
- 7. Exchange Request Token / Verifier for Access Token, **signed** request
 - `oauth_token` *Request Token*;
 - `oauth_consumer_key`
 - `oauth_nonce`
 - `oauth_signature`
 - `oauth_signature_method`
 - `oauth_version`
 - `oauth_verifier`
- 8. Service grants Access Token & Token Secret (same arguments generally as Step 2)

Application uses `oauth_token` & `oauth_token_secret` to access protected resources.

2.2.2.5 Getting request token

The first step to obtain authorization for a user is to get a **Request Token** using your **Consumer Key**. This is a temporary token that will be used to authenticate the user to your application. This token, along with a token secret, will later be exchanged for an **Access Token**.

URL structure

`/oauth/request_token`

Method

GET, POST

Parameters

Field	Description
-------	-------------

oauth_consumer_key	The Consumer Key.
oauth_signature_method	The signature method the Consumer used to sign the request. Options are “PLAINTEXT” and “HMAC-SHA1”.
oauth_signature	The signature as defined in Signing Requests.
oauth_timestamp	The timestamp is expressed in the number of seconds since January 1, 1970 00:00:00 GMT. The timestamp value must be a positive integer and must be equal or greater than the timestamp used in previous requests.
oauth_nonce	Value that is unique for all requests with that timestamp. A nonce is a random string, uniquely generated for each request.
oauth_version	OPTIONAL. If present, value must be “1.0”. Service Providers must assume the protocol version to be 1.0 if this parameter is not present.
oauth_callback	An absolute URL to which the Service Provider will redirect the User back when the Obtaining User Authorization step is completed. If the Consumer is unable to receive callbacks or a callback URL has been established via other means, the parameter value must be set to “oob” (case sensitive), to indicate an out-of-band configuration.

Response

A request token and the corresponding request token secret, URL-encoded. This token/secret pair is meant to be used with /oauth/access_token to complete the authentication process and cannot be used for any other API calls.

```
oauth_token=fa78ff7dc2a2559d&oauth_token_secret=a15eb83bfd521a9320
```

2.2.2.6 Getting the user authorization

After getting the Request Token, your application needs to present the Cloud Storage authorization page to the user, where they will be asked to give

permission to your application to access their data. The authorization page will present the user with a list of permissions your application is requesting, as defined in the authentication flow settings.

URL structure

/oauth/authorize

Query parameters

Field	Description
oauth_token	The request token obtained previously.

After authorization is complete, Cloud Storage will redirect the user back to your application using the **oauth_callback** specified with your Request Token.

If the oauth_callback parameter is set to “**oob**”, the application must find some other way of determining when the authorization step is complete. For example, the application can have the user explicitly indicate to it that this step is complete, but this flow may be less intuitive for users than the redirect flow

2.2.2.7 Obtaining the access token

Once the user has completed the authentication and authorization step, the application is expected to try to exchange its previously obtained request token for an access token.

This access token will be used in subsequent calls to retrieve actual Cloud Storage data. Your application is expected to store this access token so subsequent application starts won't force the user to go through the OAuth handshake again.

This call can be invoked via POST or GET.

URL structure

/oauth/access_token

Method

GET, POST

Parameters

Field	Description
oauth_consumer_key	The Consumer Key.
oauth_signature_method	The signature method the Consumer used to sign the request. Options are “PLAINTEXT” and “HMAC-SHA1”.
oauth_signature	The signature as defined in Signing Requests.
oauth_timestamp	The timestamp is expressed in the number of seconds since January 1, 1970 00:00:00 GMT. The timestamp value must be a positive integer and must be equal or greater than the timestamp used in previous requests.
oauth_nonce	Value that is unique for all requests with that timestamp. A nonce is a random string, uniquely generated for each request.
oauth_version	OPTIONAL. If present, value must be “1.0”. Service Providers must assume the protocol version to be 1.0 if this parameter is not present.
oauth_verifier	The value return in the previous step after successful user authorization.

Response

As a response to the request, your Access Token will be in the "oauth_token" field and the Token Secret (for signing purposes) will be in the “oauth_token_secret” field.

```
oauth_token= 0Fkc84aAH3vP07wLMd34o&oauth_token_secret=
MoB36unfR0g18gyEW5v10U
```

2.2.3 Error handling

Errors are returned using standard HTTP error code syntax. Any additional info is included in the body of the return call, JSON-formatted.

Standard API errors:

Code	Description
400	Bad input parameter. Error message should indicate which one and why.
401	Authorization required. The presented credentials, if any, were not sufficient to access the folder resource. Returned if an application attempts to use an access token after it has expired.
403	Forbidden. The requester does not have permission to access the specified resource.
404	File or folder not found at the specified path.
405	Request method not expected (generally should be GET or POST).
5xx	Internal server error

2.2.4 Folder operations

2.2.4.1 Get folder content metadata

Retrieve the information about the folder and all resources inside (only one level).

Request

URL structure

GET /folder/{id}/contents - To get information about root folder, you must set the id to "0".

Example

```
GET /folder/8704/contents
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

Field	Description
id	A unique identifier for a file or folder.
parent_id	A unique identifier for the folder's parent.
name	Name of the file or folder.
is_folder	Flag indicating whether it is a file or folder.
status	Status of a file or folder.
version	Not implemented. It always returns 1.0.
size	The number of items inside the folder.
mimetype	The media type of the file. http://www.iana.org/assignments/media-types
is_root	Flag indicating whether it is the root folder or not.
modified_at	It is the date in which the file or folder has been modified for last time.
contents	A list of each file/folder's metadata.

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json;
Content-Length: 248
{
```



```
"id": "8704",
"parent_id": null,
"name": "Personal",
"is_folder": true,
"status": "Cached",
"version": "1.0",
"size": 0,
"mimetype": null,
"is_root": true,
"modified_at": "2015-05-14T17:56:03.0101203+02:00",
"contents": [{
  "id": "9216",
  "parent_id": "8704",
  "name": "Interoperability",
  "is_folder": true,
  "status": "Cached",
  "version": "1.0",
  "size": 2,
  "mimetype": null,
  "is_root": false,
  "modified_at": "2015-05-14T17:56:03.0572075+02:00"
},
{
  "id": "12800",
  "parent_id": "8704",
  "name": "Test1",
  "is_folder": true,
  "status": "Cached",
  "version": "1.0",
  "size": 1,
  "mimetype": null,
  "is_root": false,
  "modified_at": "2015-05-14T17:56:03.0572075+02:00"
}]
}
```

2.2.4.2 Get folder metadata

Retrieve the information about the folder.

Request

URL structure

GET /folder/{id} - To get information about root folder, you must set the id to “0”.

Example

```
GET /folder/9216
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

Field	Description
id	A unique identifier for a file or folder.
parent_id	A unique identifier for the folder’s parent.
name	Name of the file or folder.
is_folder	Flag indicating whether it is a file or folder.
status	Status of a file or folder.
version	Not implemented. It always returns 1.0.
size	The number of items inside the folder.
mimetype	The media type of the file. http://www.iana.org/assignments/media-types
is_root	Flag indicating whether it is the root folder or not.
modified_at	It is the date in which the file or folder has been modified for last time.

contents	A list of each file/folder's metadata.
----------	--

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json;
Content-Length: 248
{
  "id": "9216",
  "parent_id": "8704",
  "name": "Interoperability",
  "is_folder": true,
  "status": "Cached",
  "version": "1.0",
  "size": 2,
  "mimetype": null,
  "is_root": false,
  "modified_at": "2015-05-14T17:56:03.0572075+02:00"
}
```

2.2.4.3 Create folder

Create a folder where the application needs to provide as input, a JSON that identifies the display name of the folder to be created.

Request

URL structure

POST /folder

Request Headers

Field	Description
Content-Type	The content type and character encoding of the response. The content type must be application/json, and the character encoding must be UTF-8.

Request Body

Field	Description
name	The user-visible name of the folder to be created.
parent_id	ID of the folder where the folder is going to be uploaded.

Example

```
POST /folder
Content-Type: application/json
{
  "name": "TestFolder",
  "parent_id": "1536"
}
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

Field	Description
id	A unique identifier for a file or folder.
parent_id	A unique identifier for the folder's parent.
name	Name of the file or folder.
is_folder	Flag indicating whether it is a file or folder.
status	Status of a file or folder.
version	Not implemented. It always returns 1.0.
size	The number of items inside the folder.

mimetype	The media type of the file. http://www.iana.org/assignments/media-types
is_root	Flag indicating whether it is the root folder or not.
modified_at	It is the date in which the file or folder has been modified for last time.

Response example

HTTP/1.1 201 Created

Content-Type: application/json;

Content-Length: 248

```
{
  "id": "9216",
  "parent_id": "8950",
  "name": " TestFolder",
  "is_folder": true,
  "status": "New",
  "version": "1.0",
  "size": 0,
  "mimetype": null,
  "is_root": false,
  "modified_at": "2015-05-14T17:56:03.0572075+02:00"
}
```

2.2.4.4 Delete folder

Delete a folder and all its content.

Request

URL structure

DELETE /folder/{id}

Example

```
DELETE /folder/9216
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

Field	Description
id	A unique identifier for a file or folder.
parent_id	A unique identifier for the folder's parent.
name	Name of the file or folder.
is_folder	Flag indicating whether it is a file or folder.
status	Status of a file or folder.
version	Not implemented. It always returns 1.0.
size	The number of items inside the folder.
mimetype	The media type of the file. http://www.iana.org/assignments/media-types
is_root	Flag indicating whether it is the root folder or not.
modified_at	It is the date in which the file or folder has been modified for last time.

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json;
Content-Length: 248
{
  "id": "9216",
  "parent_id": "8950",
```

```
"name": " TestFolder",  
  "is_folder": true,  
  "status": "Deleted",  
  "version": "1.0",  
  "size": 0,  
  "mimetype": null,  
  "is_root": false,  
  "modified_at": "2015-05-14T17:56:03.0572075+02:00"  
}
```

2.2.4.5 Update folder metadata (move and rename)

An application can update various attributes of a folder. The application needs to provide as input, JSON that identifies the new attribute values for the folder.

This attributes are:

- name: an application can rename a folder changing the value of the name element.
- parent_id: an application can move a folder to a different parent folder by changing the value of the parent element.

Request

URL structure

PUT /folder/{id}

Request Headers

Field	Description
Content-Type	The content type and character encoding of the response. The content type must be application/json, and the character encoding must be UTF-8.

Request Body

Field	Description
name	The user-visible name of the folder to be renamed.

parent_id	ID of the folder where the folder is going to be moved.
-----------	---

Example

```
PUT /folder/9216
Content-Type: application/json
{
  "name": "FolderRenamed",
  "parent_id": "8950"
}
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

Field	Description
id	A unique identifier for a file or folder.
parent_id	A unique identifier for the folder's parent.
name	Name of the file or folder.
is_folder	Flag indicating whether it is a file or folder.
status	Status of a file or folder.
version	Not implemented. It always returns 1.0.
size	The number of items inside the folder.
mimetype	The media type of the file. http://www.iana.org/assignments/media-types

is_root	Flag indicating whether it is the root folder or not.
modified_at	It is the date in which the file or folder has been modified for last time.

Response example

```
HTTP/1.1 200 OK

Content-Type: application/json;
Content-Length: 248

{
  "id": "9216",
  "parent_id": "8950",
  "name": " FolderRenamed ",
  "is_folder": true,
  "status": "Modified",
  "version": "1.0",
  "size": 0,
  "mimetype": null,
  "is_root": false,
  "modified_at": "2015-05-14T17:56:03.0572075+02:00"
}
```

2.2.5 File operations

2.2.5.1 Get file metadata

Retrieve the information about the file.

Request

URL structure

GET /file/{id}

Example

```
GET /file/7712
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

Field	Description
id	A unique identifier for a file or folder.
parent_id	A unique identifier for the folder's parent.
name	Name of the file or folder.
is_folder	Flag indicating whether it is a file or folder.
status	Status of a file or folder.
version	Not implemented. It always returns 1.0.
size	The number of items inside the folder.
mimetype	The media type of the file. http://www.iana.org/assignments/media-types
is_root	Flag indicating whether it is the root folder or not.
modified_at	It is the date in which the file or folder has been modified for last time.

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json;
Content-Length: 248
{
  "id": "7712",
```

```
{
  "parent_id": "8704",
  "name": "document.pdf",
  "is_folder": false,
  "status": "Cached",
  "version": "1.0",
  "size": 758525,
  "mimetype": "application/pdf",
  "is_root": false,
  "modified_at": "2015-05-14T17:56:03.0572075+02:00"
}
```

2.2.5.2 Download file

Retrieve the file data.

Request

URL structure

GET /file/{id}/data

Example

```
GET /file/7712/data
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

The response body contains the retrieved file data.

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json;
```

Content-Length: 248
<File content.... >

2.2.5.3 Create a file

To create a file, the application needs to provide the file binary in the body and the file name as a query argument. Optionally, it can also provide the parent argument to locate the file in a specific folder. Otherwise, the file will be placed in the root folder.

Request

URL structure

POST /file

Request Headers

Field	Description
Content-Type	The content type and character encoding of the response. The content type must be application/json, and the character encoding must be UTF-8.

Request query arguments

Field	Description
name	The user-visible name of the file to be created.
parent_id	ID of the folder where the file is going to be created. If no ID is passed, it will use the top-level folder.

Example

POST /file?name=document.pdf&parent_id=1536
Content-Length: 294
<file binary>

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

Field	Description
id	A unique identifier for a file or folder.
parent_id	A unique identifier for the folder's parent.
name	Name of the file or folder.
is_folder	Flag indicating whether it is a file or folder.
status	Status of a file or folder.
version	Not implemented. It always returns 1.0.
size	The number of items inside the folder.
mimetype	The media type of the file. http://www.iana.org/assignments/media-types
is_root	Flag indicating whether it is the root folder or not.
modified_at	It is the date in which the file or folder has been modified for last time.

Response example

```
HTTP/1.1 201 Created
Content-Type: application/json;
Content-Length: 294
{
  "id": "5687",
  "parent_id": "1536",
  "name": "document.pdf",
  "is_folder": false,
  "status": "Cached",
```

```
"version": "1.0",  
"size": 758525,  
"mimetype": "application/pdf",  
"is_root": false,  
"modified_at": "2015-05-14T17:56:03.0572075+02:00"  
}
```

2.2.5.4 Upload file data

An application can upload data to a file. The file binary will be sent in the request body.

Request

URL structure

PUT /file/{id}/data

Request Headers

Field	Description
Content-Type	The content type and character encoding of the response. The content type must be application/json, and the character encoding must be UTF-8.

Example

```
PUT /file/5687/data  
Content-Length: 294  
<file binary>
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json;
Content-Length: 294
```

2.2.5.5 Delete file

Delete a file.

Request

URL structure

DELETE /file/{id}

Example

```
DELETE /file/5687
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

Field	Description
id	A unique identifier for a file or folder.
parent_id	A unique identifier for the folder's parent.
name	Name of the file or folder.
is_folder	Flag indicating whether it is a file or folder.
status	Status of a file or folder.
version	Not implemented. It always returns 1.0.

size	The number of items inside the folder.
mimetype	The media type of the file. http://www.iana.org/assignments/media-types
is_root	Flag indicating whether it is the root folder or not.
modified_at	It is the date in which the file or folder has been modified for last time.

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json;
Content-Length: 248
{
  "id": "5687",
  "parent_id": "8704",
  "name": "document.pdf",
  "is_folder": false,
  "status": "Deleted",
  "version": "1.0",
  "size": 758525,
  "mimetype": "application/pdf",
  "is_root": false,
  "modified_at": "2015-05-14T17:56:03.0572075+02:00"
}
```

2.2.5.6 Upload file metadata (move and rename)

An application can update various attributes of a file. The application needs to provide as input, JSON that identifies the new attribute values for the folder.

This attributes are:

- name: an application can rename a file changing the value of the name element.
- parent_id: an application can move a file to a different parent folder by changing the value of the parent element.

Request

URL structure

PUT /file/{id}

Request Headers

Field	Description
Content-Type	The content type and character encoding of the response. The content type must be application/json, and the character encoding must be UTF-8.

Request Body

Field	Description
name	The user-visible name of the folder to be renamed.
parent_id	ID of the folder where the folder is going to be moved.

Example

```
PUT /folder/5687
Content-Type: application/json
{
  "name": "documentRenamed.pdf",
  "parent_id": "8950"
}
```

Response

Header

Field	Description
Content-Length	The length of the retrieved content.
Content-Type	The content type and character encoding of the response.

Response Body

Field	Description
-------	-------------

id	A unique identifier for a file or folder.
parent_id	A unique identifier for the folder's parent.
name	Name of the file or folder.
is_folder	Flag indicating whether it is a file or folder.
status	Status of a file or folder.
version	Not implemented. It always returns 1.0.
size	The number of items inside the folder.
mimetype	The media type of the file. http://www.iana.org/assignments/media-types
is_root	Flag indicating whether it is the root folder or not.
modified_at	It is the date in which the file or folder has been modified for last time.

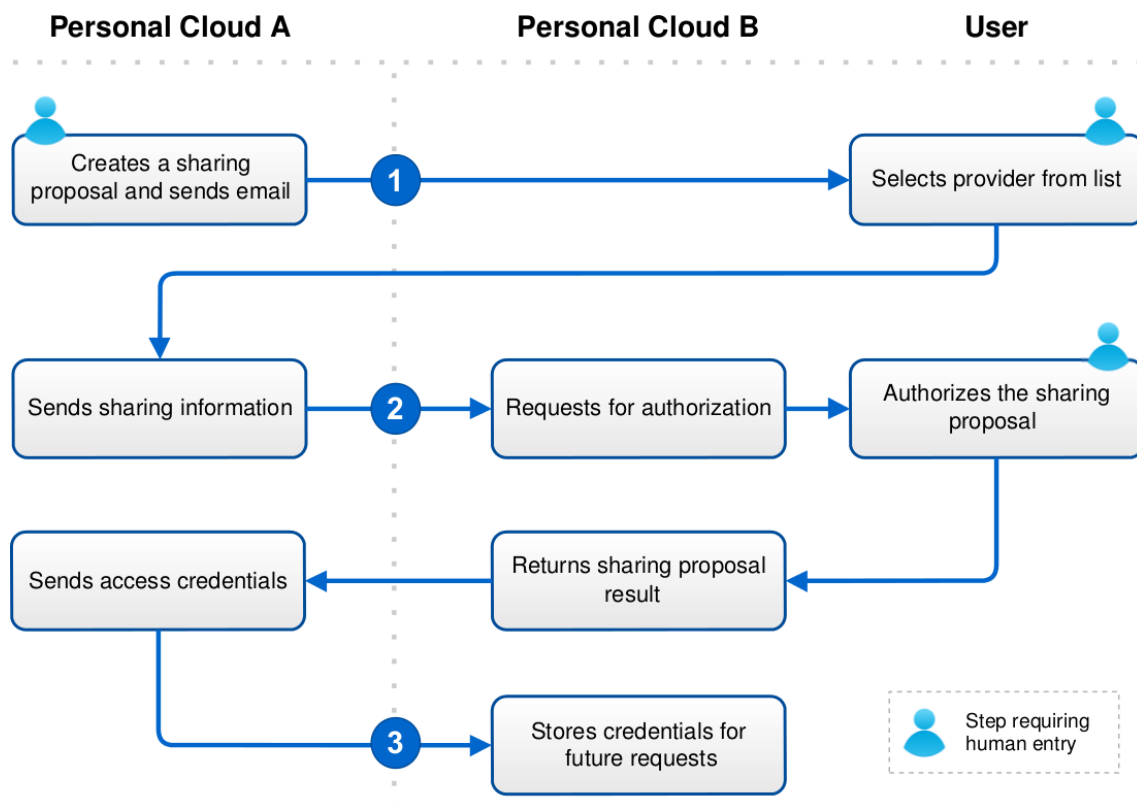
Response example

```
HTTP/1.1 200 OK
Content-Type: application/json;
Content-Length: 248
{
  "id": "5687",
  "parent_id": "8950",
  "name": "documentRenamed.pdf ",
  "is_folder": false,
  "status": "Modified",
  "version": "1.0",
  "size": 758525,
  "mimetype": "application/pdf",
  "is_root": false,
  "modified_at": "2015-05-14T17:56:03.0572075+02:00"
}
```

2.3. Interoperability protocol

2.3.1 Introduction

The interoperability protocol enables different Personal Clouds to share resources among them via an Application Programming Interface (API), without forcing users to be in the same provider. More generally, the interoperability protocol creates a freely-implementable and generic methodology for allowing Personal Cloud interoperability.



2.3.2 Prerequisites

Having two Personal Clouds (PC1 and PC2) that wish to interoperate with each other. They must meet the following requirements before using the present specification.

- Once the interoperability process is completed, Personal Clouds must use APIs to access protected resources. In case they do not implement the

same Storage API, Personal Cloud 1 must implement an adapter to access Personal Cloud 2 API, and vice versa.

- Personal Cloud 1 must be registered in Personal Cloud 2 and validated as an authorized service in order to obtain its credentials, and vice versa. The method in which Personal Clouds register with each other and agree to cooperate is beyond the scope of this specification.

2.3.3 Protocol foundations

The interoperability protocol is based in three key features/components that are the foundations of the protocol:

1. **OAuth:** Is an open standard for authorization. We use it in order to guarantee data protection and control access to the resources exposed by Personal Clouds through the interoperability protocol.
2. **IDs resource naming:** Using unique IDs to identify Cloud resources instead of other ways of identifying them, such as paths, makes the protocol more robust, less ambiguous and requires less computational effort.
3. **Storage API:** In order to be able to share and access other Clouds resources, a Storage API must be implemented, so all the Clouds have a common way to interact.

2.3.4 Interoperability process

The interoperability protocol used by NEC Cloud Storage is divided in three steps explained below.

- User invitation
- Invitation acceptance
- Access credentials

From a high level perspective, the interoperability process is a very straight forward process between Personal Clouds and users.

Here there is a brief description of the process from the NEC CS point of view.

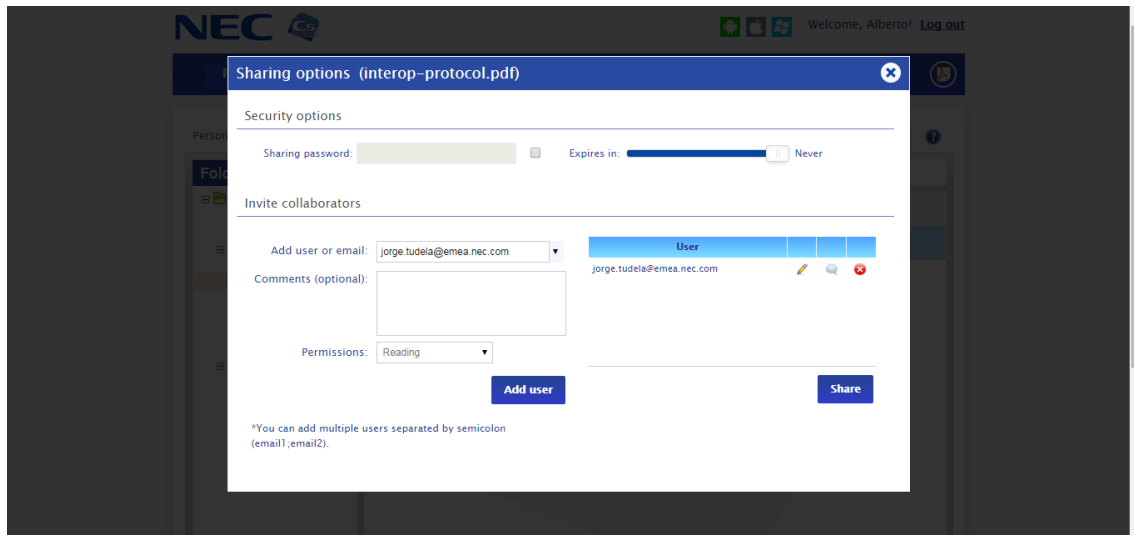
Actors	Action
NEC CS	1. Creates a sharing proposal
	2. Sends a mail to the user
User	3. Opens the sharing proposal
	4. Choose Personal Cloud
	5. Authorizes the sharing proposal
Personal Cloud	6. Opens the shared resource

More detailed explanations can be found along this document.

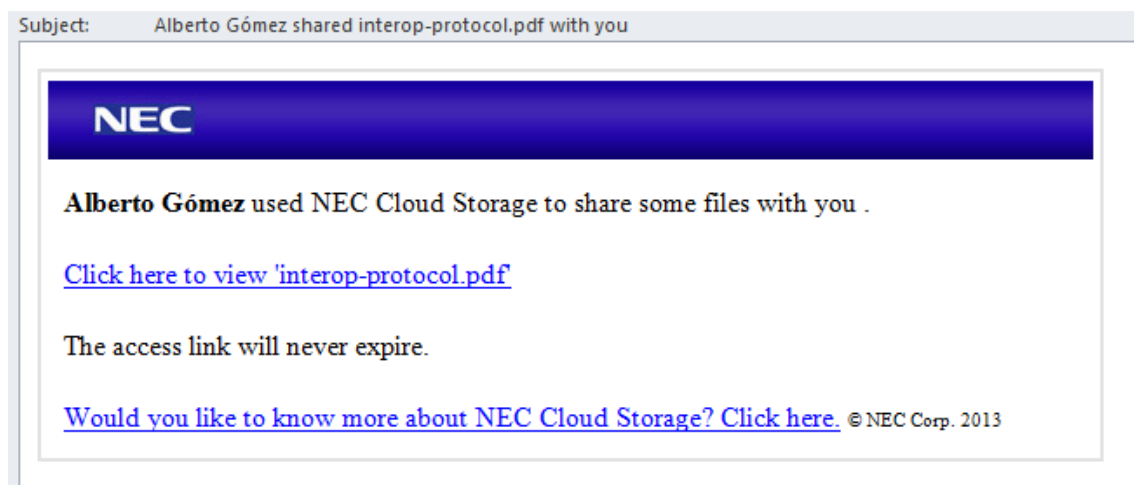
3.3.4.1 User invitation

1. User sends an invitation

NEC Cloud Storage user wants to share a file or folder with another user. Therefore, the user login in his/her NEC Cloud Storage account, selects the file or folder he/she wants to share, and enters the email of the user.

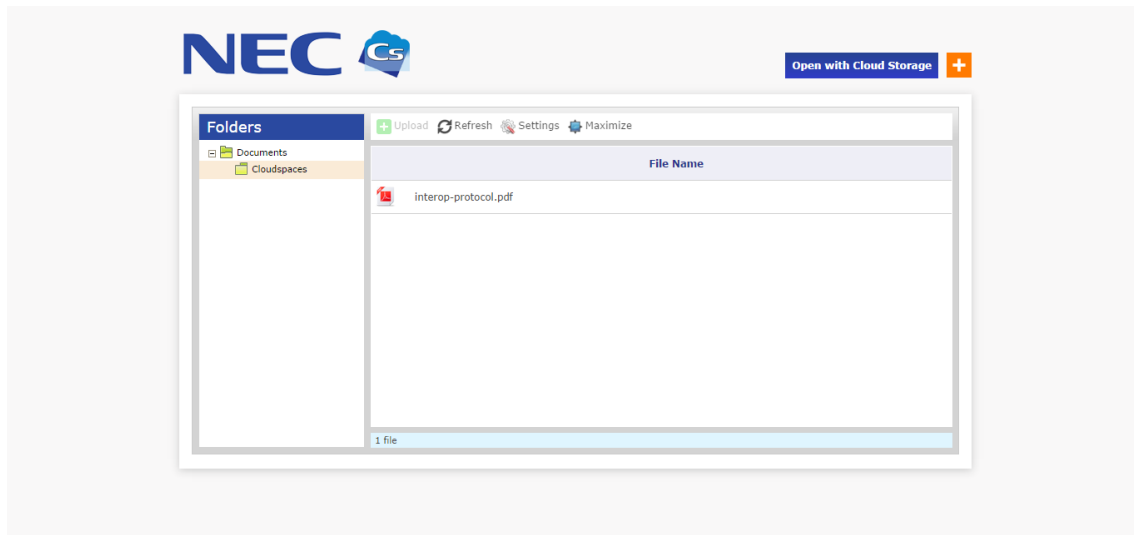


The recipient will receive an email indicating the intention of the user from NEC Cloud Storage to share a file or folder with him/her and a link to a website located on NEC Cloud Storage.

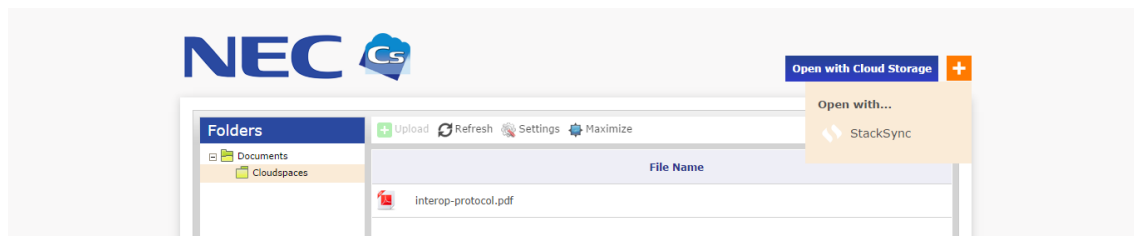


2. The recipient selects its Personal Cloud

The user who has received the email clicks on the link and is redirected to a NEC Cloud Storage page (as the origin of the sharing is NEC Cloud Storage the link should point to this site), where he/she is asked to select a Personal Cloud from a list of services that have an agreement among different Personal Clouds. In this case, as an example, the user will select StackSync, because he/she already have an account on this Cloud.



In the image above, the user can see the file that someone has shared with him/her.



On the right top of the image, the user will be able to choose the Personal Cloud wished.

3. Creating the interoperability proposal

At this time, NEC Cloud Storage creates the interoperability proposal sending an HTTP POST request to StackSync's share URL (in this example case, another instance of NEC Cloud Storage) previously established and that we previously saved.

Request

POST /AdvancedFeatures/ExternalSharingProposal.aspx

Request parameters

Parameter	Type	Description
-----------	------	-------------

Parameter	Type	Description
share_id	string	A unique value that identifies the interoperability proposal. This is auto generated with a GUID format.
resource_url	string	An absolute URL to access the shared resource located in NEC Cloud Storage.
owner_name	string	The name corresponding to the owner of the folder.
owner_email	string	The email corresponding to the owner of the folder.
resource_name	string	The name of the folder or file.
permission	string	Permissions granted to the recipient. Options are read-only and read-write.
recipient	string	The email corresponding to the user who the folder has been shared with.
callback	string	An absolute URL to which the destination Private Cloud (e.g. StackSync) will redirect the user back when the invitation step is completed.
protocol_version	string	MUST be set to 1.0. Services MUST assume the protocol version to be 1.0 if this parameter is not present.

Request sample

POST

/AdvancedFeatures/ExternalSharingProposal.aspx HTTP/1.1
Host: dev.neccloudstorage.com:8090
Content-Length: 384
Content-Type: application/x-www-form-urlencoded


```
share_id=bd346a81-67fa-4e2c-8637-  
e173135fab59&resource_url=%2FPersonal%2FCloudspaces%2Finterop-  
protocol.pdf&owner_name=Alberto&owner_email=alberto.gomez%40emea.nec.  
com&folder_name=interop-protocol.pdf&permission=read-  
only&recipient=carlos.torrillas%40emea.nec.com&callback=http%3A%2F%2Fd  
ev.neccloudstorage.com%3A8090%2FAdvancedFeatures%2FExternalSharingRe  
sponse.aspx&protocol_version=1.0
```

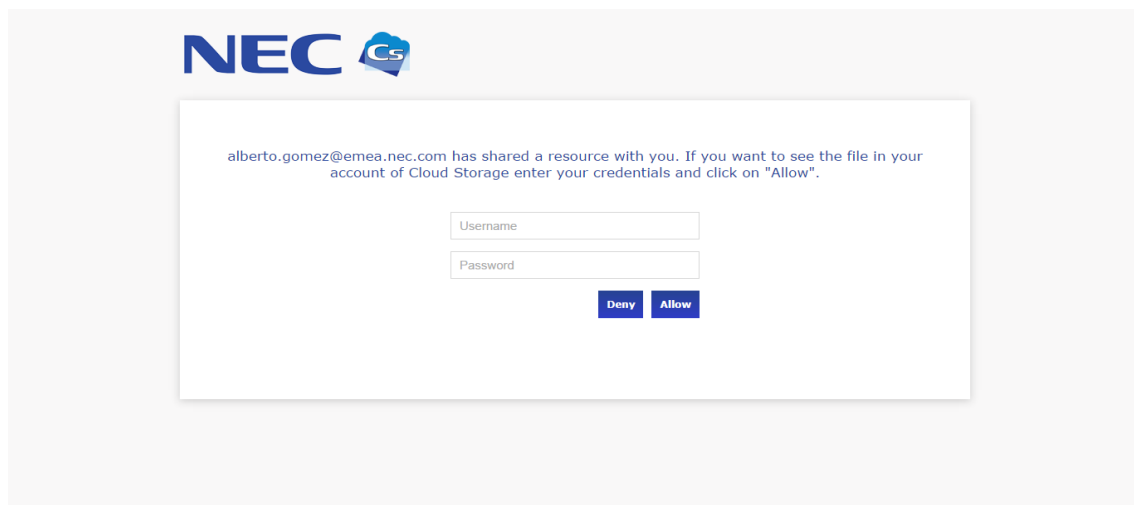
RESPONSE

HTTP/1.1 200 OK

3.3.4.2 Invitation acceptance

1. The user accepts the invitation

StackSync (in this case NEC Cloud Storage) shows the details of the file or folder share invitation request to the recipient and the user must provide its credentials and explicitly accept the invitation.



The screenshot shows a web interface for NEC Cloud Storage. At the top left is the NEC logo with a blue 'Cs' icon. Below it is a white dialog box with a light gray border. Inside the dialog box, the text reads: "alberto.gomez@emea.nec.com has shared a resource with you. If you want to see the file in your account of Cloud Storage enter your credentials and click on 'Allow'." Below this text are two input fields: "Username" and "Password". At the bottom right of the dialog box are two buttons: "Deny" and "Allow".

2. Returning the proposal response

Once StackSync (2nd instance of NEC Cloud Storage) has obtained approval or denial from user, it uses the callback obtained from the NEC Cloud Storage request, shown above, to inform NEC Cloud Storage about the user decision.

Then, StackSync uses the callback to construct an HTTP GET request, and directs the user's web browser to that URL.

Request

GET /AdvancedFeatures/ExternalSharingResponse/stacksync

Request parameters

Field	Type	Description
share_id	string	A unique value that identifies the interoperability proposal.
accepted	string	A string indicating whether the invitation has been accepted or denied. true and false are the only possible values.

Request sample

GET
/AdvancedFeatures/ExternalSharingResponse.aspx?share_id=bd346a81-67fa-4e2c-8637-e173135fab59&accepted=true HTTP/1.1
Host: dev.neccloudstorage.com:8090

RESPONSE
HTTP/1.1 302 Found

3.3.4.3 Access credentials

1. Granting access to the service

When NEC Cloud Storage receives the proposal result, firstly it validates these data and then generates the access credentials. Finally, it sends an HTTP POST request to StackSync where the resource will be displayed.

NEC Cloud Storage specifies what type of authentication protocol and version must be used to access the resource. In this case, the authentication protocol

used is OAuth 1.0a and at this point, NEC Cloud Storage web module initiates a standard OAuth process to generate the required token that will be sent to the second Personal Cloud.

For this purpose, a specific module for processing OAuth authentication mechanism has been included which provides both consumer and provider implementations.

To this end, the second Personal Cloud must check the auth_protocol and auth_protocol_version parameters, and also the specific parameters that will be included in the request. This information will be used when the user try to access to the shared resources.

Request

POST /Files.aspx

Request parameters

Field	Type	Description
share_id	string	A randomly generated value that uniquely identifies the interoperability proposal.
auth_protocol	string	The authentication protocol used to access the shared resource (e.g.oauth)
auth_protocol_version	string	The version of the authentication protocol (e.g. 1.0a).

Other authentication-specific parameters are sent together with the above parameters, these parameters may include values like tokens, timestamps or signatures which will be necessary to get and perform the actions with the resource depending on the kind of authentication protocol used.

NOTE:

In this first release, Cloud Storage does not send the HTTP POST request. It does a SSO with the data obtained instead.

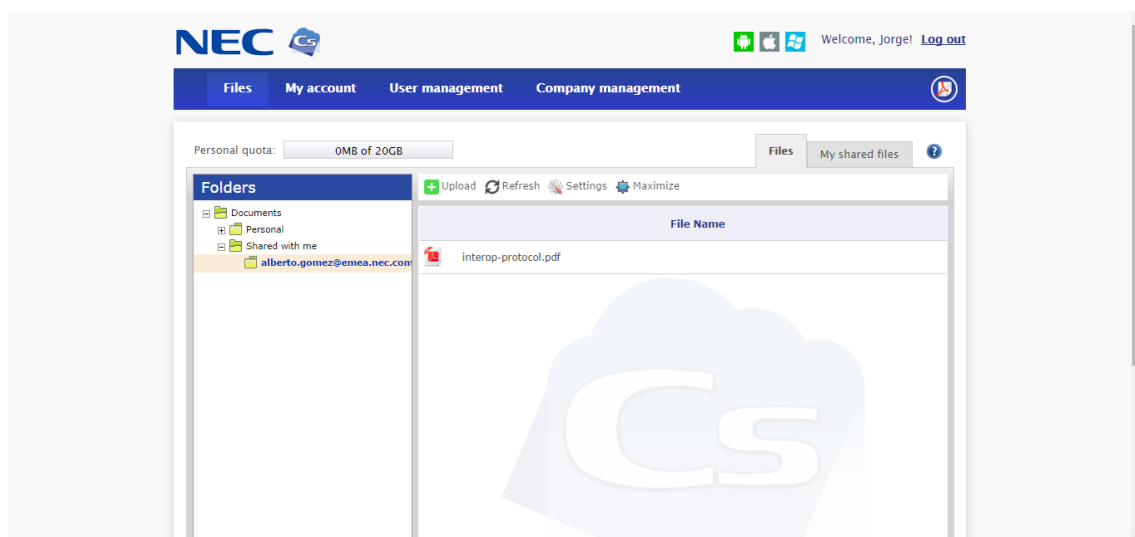
Request sample (No HTTP POST)

GET

```
/?SessionId=wMWdX3xy%2bPt89Mq87dGqGkPBjj%2fzmQej68IrO%2bag1hec  
OQiipOxCi0460RbJ1Pu1G7MO3IHAPw5tCypVI8wqUA%3d%3d&action=none  
HTTP/1.1 Host: dev.neccloudstorage.com:8090
```

RESPONSE

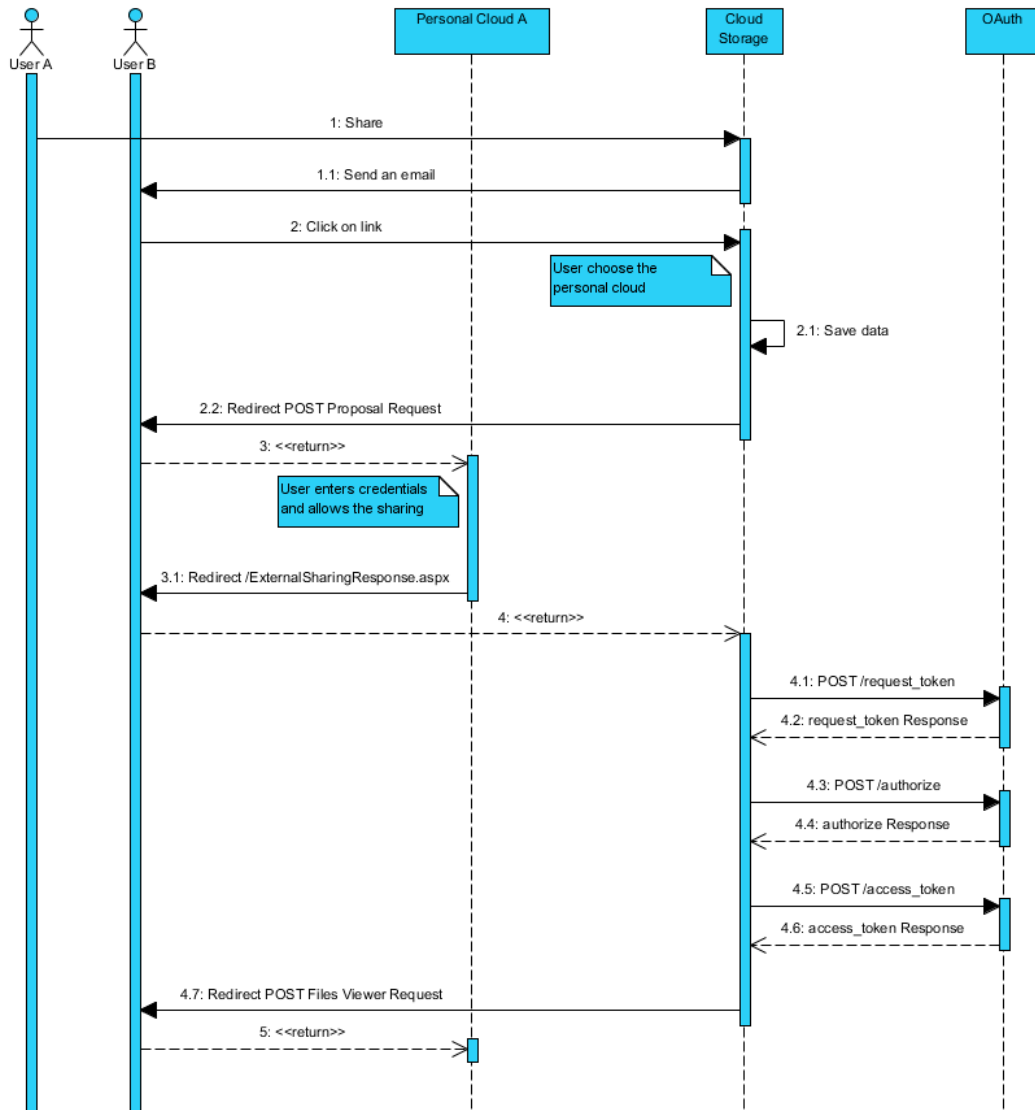
HTTP/1.1 302 Found



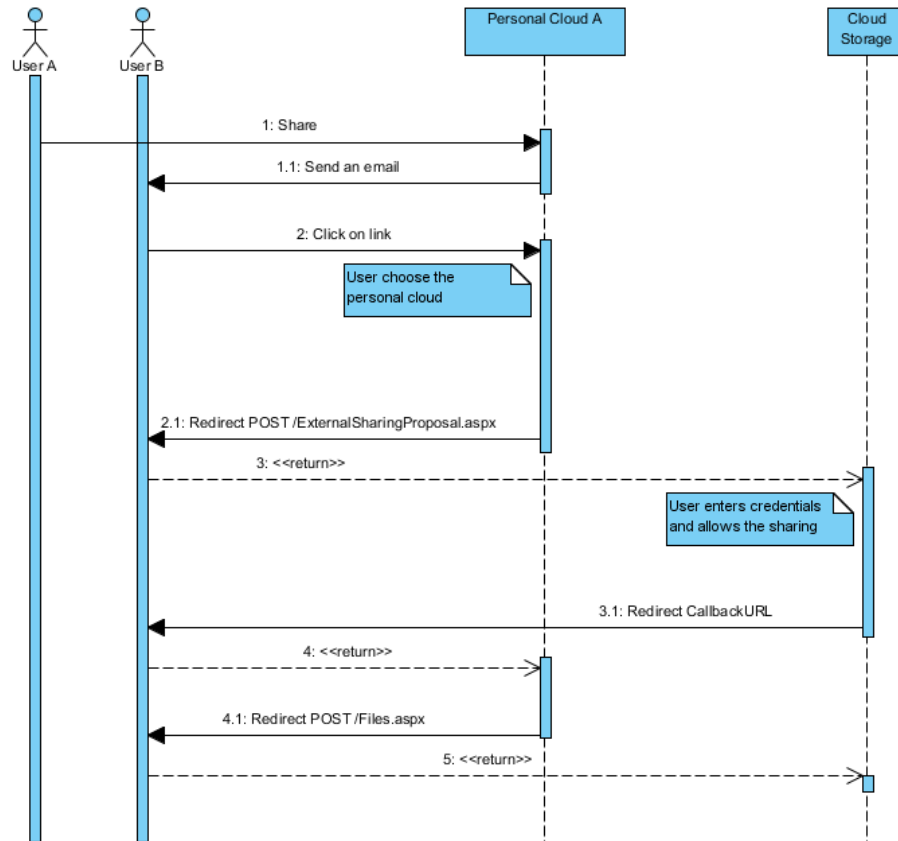
In the image above, StackSync (2nd instance of NEC Cloud Storage) shows the shared resource. To be able to access to the resource is necessary to use the tokens sent in last step.

2.3.5 Sequence diagram

Here we can see a complete workflow of the interoperability protocol where an NEC Cloud Storage user (User A) shares a file or folder with an external Personal Cloud user (User B).



Also, we can see an example where an external Personal Cloud user (User A) shares a resource with a NEC Cloud Storage user (User B).



2.4. Traces

2.4.1 Introduction

Increasingly, the role of production datasets in today's research in Cloud computing is gaining importance. Such valuable information, coming from the real use of widely deployed Cloud services, constitutes a solid ground for researchers to devise novel techniques and advances.

In this way, a development was carried out to make a comprehensive analysis of the use of Cloud Storage. This aims to benefit from the cooperation between industry partners and universities to produce high-quality research.

This will benefit partners, as well as the whole project, in a twofold manner:

1. Academic partners will work with real-world data, giving to their research technical strength and impact.

2. The advances achieved using a specific dataset are applicable to the service where such a dataset comes from. This represents that many potentially beneficial and novel techniques will be ready for exploration to industry partners during the development of the project.

To do this, we have created a log file for each module with a "comma separate values" (csv) as a type format.

The number of fields is always the same, although not all operations require all fields. In that case, the fields will be empty, but still separated by commas.

It has been done in this way in order to the log files can be analysed easier later. Otherwise, it would be very difficult to analyse a log file with irregular expressions and variable messages.

User records are totally anonymous and no personal information of users are recorded (e.g. email addresses, IPs). This principle is supported by the specification of the datasets, which does not include any type of sensitive user information.

2.4.2 Description of each field

Here is a description for each field:

Field	Description
RequestID	Unique identifier for operation
Timestamp	Time in which the trace is captured
UserID	Unique identifier for the user
OrganizationID	Unique identifier for the organization belonging to the user who is doing the action.
FileID	Unique identifier for the file.
FileExtension	Extension of the file (.txt, .pdf, ...)

FileSize	Size (in bytes) of the file is being uploading, downloading...
Failed	Information of the error. In which operation happened
FreeBytes	Available space in user account (in bytes)
Operation	Name of the operation (upload file, move folder, ...)
RequestMsg	Information of the request (Init request, End request, ...)
Module	Module where the operation has been made
FolderID	Unique identifier for the folder
Server	Unique identifier for the server
SharedBy	Unique identifier for the user is sharing
SharedTo	Unique identifier for the user has been shared
TotalBytes	Total space a user has in its account
UsedBytes	Used space in user account (in bytes)
Source	Client where the request is sent (pc, web, mobile)
Device	Detail of the device (iOS, Android)

2.4.3 WebDAV module

WebDAV is responsible for managing files and we get the operations below:

- Login
- Delete file or folder
- Copy file or folder
- Move file or folder
- Create folder
- Upload file
- Download file

- Get resource information

2.4.4 AdvancedFeatures module

This module is responsible for performing advanced tasks related to users and sharing files.

We get the following operations:

- Add account
- Delete account
- Add user
- Delete user
- Share
- Unshare
- Get total quota
- Get used quota

2.4.5 Sample

Here is an example of a user has made two actions from different devices:

RequestID	Timestamp	UserID	OrganizationID
adbd956-121f-4e0a-83fa-0603f4dea65b	10/09/2015 08:07.52.299	OGpXNNGeiFIEbW0q4ygck9blunU=	GqJpCTAOSbFAKCDH95QYHdbho+s=
62ea7780-9075-4c97-b7b4-d66748c510c6	10/09/2015 08:25.21.491	OGpXNNGeiFIEbW0q4ygck9blunU=	GqJpCTAOSbFAKCDH95QYHdbho+s=
62ea7780-9075-4c97-b7b4-d66748c510c6	10/09/2015 08:25.22.102	OGpXNNGeiFIEbW0q4ygck9blunU=	GqJpCTAOSbFAKCDH95QYHdbho+s=
fbdd83e9-def0-45f7-b1d7-0a121130bbad	10/09/2015 08:31.52.017	OGpXNNGeiFIEbW0q4ygck9blunU=	GqJpCTAOSbFAKCDH95QYHdbho+s=
fbdd83e9-def0-45f7-b1d7-0a121130bbad	10/09/2015 08:31.52.153	OGpXNNGeiFIEbW0q4ygck9blunU=	GqJpCTAOSbFAKCDH95QYHdbho+s=

FileID	FileExtension	FileSize	Failed	FreeBytes	Operation	RequestMsg	Module	FolderID
				10737418240	Login	Login successfull	WebDav	
0YhJdzOffYIYoXnU/1D1Nsct3E0=.jpg	.jpg	879394		10737418240	UploadFile	Initiating request	WebDav	
0YhJdzOffYIYoXnU/1D1Nsct3E0=.jpg	.jpg	879394		10736538846	UploadFile	Operation completed	WebDav	
KLcrvFTyIPwE1vWelPHfj79GPg=				10736538846	Share	Initiating request	AdvancedFeatures	tQbjJMki8+0roeVks52aLU/VqM=
KLcrvFTyIPwE1vWelPHfj79GPg=				10736538846	Share	Operation completed	AdvancedFeatures	tQbjJMki8+0roeVks52aLU/VqM=

Server	SharedBy	SharedTo	TotalBytes	UsedBytes	Source	Device
KLcrvFTyIPwE1vWelPHfj79GPg=			21474836480	10737418240	Mobile	iOS
KLcrvFTyIPwE1vWelPHfj79GPg=			21474836480	10737418240	Mobile	iOS
KLcrvFTyIPwE1vWelPHfj79GPg=			21474836480	10738297634	Mobile	iOS
KLcrvFTyIPwE1vWelPHfj79GPg=	OGpXNNGeiFIEbW0q4ygck9blunU=	nLkaYeCnDZSKsH5D2I5wbK/rm/o=	21474836480	10738297634	Web	
KLcrvFTyIPwE1vWelPHfj79GPg=	OGpXNNGeiFIEbW0q4ygck9blunU=	nLkaYeCnDZSKsH5D2I5wbK/rm/o=	21474836480	10738297634	Web	

2.4.6 Items information

We have obtained the user, the size and the location for each file.

Files	User	File Size	ContainerID
GGykhU4L/t9aq5ISKRxtGT1OtdQ=	OGpXNNGeiFIEbW0q4ygcK9blunU=	1035541842	fxVWF+uJCrj/gMqFZsamAeTnc50=
jBj1lHENJpeug7FZJFn47iksLrk=	OGpXNNGeiFIEbW0q4ygcK9blunU=	854232	nLkaYeCnDZSKsH5D2I5wbK/rm/o=
62ea7780-9075-4c97-b7b4-d66748c510c6	OGpXNNGeiFIEbW0q4ygcK9blunU=	2525481	0YhJdzOffYIYoXnU/1D1Nsct3E0=
KogcGy1mlqHT84UcTw6GOXnV5CI=	GqJpCTAOSbFAKCDH95QYHdbho+s=	4452621	7m+JfSTxP2kwYtlegtbl207X0Qc=
us7KfNoxwBnkNDtw6+GeBKBSSH4=	GqJpCTAOSbFAKCDH95QYHdbho+s=	142586258	KLcrvFTyIPwE1vWelPHfjj79GPg=

2.4.7 Sharing information

We have obtained all shared resources. To do this, can see the user who has shared, the user who has been shared, the file or folder shared, the size of the file and the container where the resource is.

User1	User2	File	File Size	ContainerID
OzDvLsjUiCifb9cqmq3i/br9tU4=	GGykhU4L/t9aq5ISKRxtGT1OtdQ=	GGykhU4L/t9aq5ISKRxtGT1OtdQ=	1035541842	fxVWF+uJCrj/gMqFZsamAeTnc50=
OzDvLsjUiCifb9cqmq3i/br9tU4=	jBj1lHENJpeug7FZJFn47iksLrk=	jBj1lHENJpeug7FZJFn47iksLrk=	854232	nLkaYeCnDZSKsH5D2I5wbK/rm/o=
OzDvLsjUiCifb9cqmq3i/br9tU4=	fh2JfCothAnkNPtwe+GeBKBUGC2=	62ea7780-9075-4c97-b7b4-d66748c510c6	2525481	0YhJdzOffYIYoXnU/1D1Nsct3E0=
jBj1lHENJpeug7FZJFn47iksLrk=	OzDvLsjUiCifb9cqmq3i/br9tU4=	KogcGy1mlqHT84UcTw6GOXnV5CI=	4452621	7m+JfSTxP2kwYtlegtbl207X0Qc=
jBj1lHENJpeug7FZJFn47iksLrk=	GGykhU4L/t9aq5ISKRxtGT1OtdQ=	us7KfNoxwBnkNDtw6+GeBKBSSH4=	142586258	KLcrvFTyIPwE1vWelPHfjj79GPg=

3. eyeOS

3.1 Introduction

eyeOS is a web platform that provides a remote virtual desktop for the end user.

The overall user experience is strongly influenced by the classic desktop design, widely known thanks to the most popular operating system on the market. eyeOS Personal Web Desktop includes several features such as: file manager, contacts, groups and other collaborative capabilities. eyeOS Personal Web Desktop is a disruptive technology that fits in perfectly with the CloudSpaces Open Personal Cloud paradigm.

One of the key values that eyeOS provides is the possibility to work directly with files in the cloud. eyeOS does not require users to manually download any files onto their computer nor is it necessary to install anything locally, so the experience is totally transparent: users just log into a website and start working with their files normally.

Furthermore, eyeOS lets you add additional services and applications within the web desktop, so that all the company or organization's web resources are available within a single controlled environment that can be accessed using single sign-on.

By combining eyeOS' web file management capabilities with Personal Cloud, users can access their Personal Cloud contents via web, with a user experience very similar to local desktop environments.

eyeOS allows users to quickly and safely access private data stored in their Personal Cloud and use them remotely. They are able to access the information from multiple devices. Desktop PC, laptop, tablet, smartphone, anywhere and any time.

In a constantly changing environment such as today's, being able to access data regardless of the device used, and share said data with other users who only need an Internet connection and a browser, constitutes an important advantage.

Within the eyeOS platform, one of its key features is file management. It provides an interface that allows the user to access the files stored in their Personal Cloud directly from the browser, with an experience similar to the file manager of any desktop operating system, such as Microsoft Windows™ or GNU/Linux. For example users can see any saved documents they have saved online on Stacksync and NEC, create directories, move files, share documents, etc.

In order for eyeOS to provide these services to the user, it needs to communicate with the Personal Cloud to obtain all the user information it needs. This communication is made using the Storage API and it provides all the resources necessary that allows the user to manage their files efficiently.

User can share their information with other users, regardless of the Personal Cloud that they use (interoperability). For example, a Stacksync user can share files with another Stacksync user or with another NEC user. eyeOS provides an interface that helps the user at all times to decide whether the content should continue to be shared and which users may access it.

A main feature of the eyeOS platform is to establish collaborative environments so that users can exchange information in real time. It provides the eyeDocs application, a word processor that allows users to edit files previously shared between them in a collaborative environment.

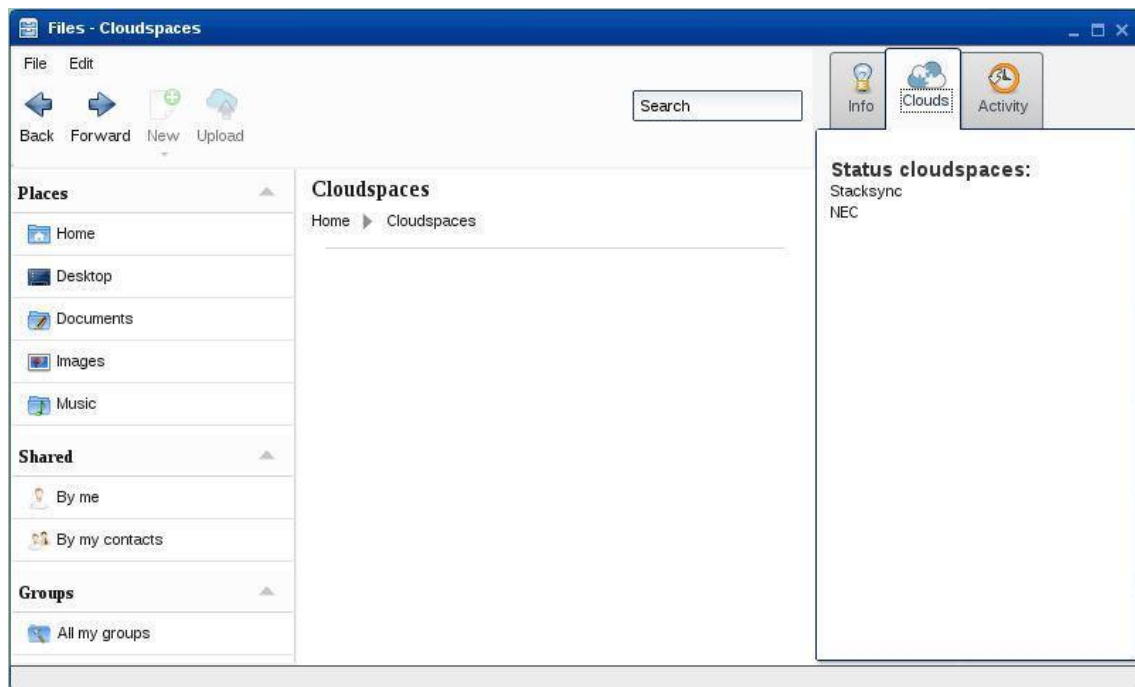
eyeOS users also have tools to add comments to files and manage their calendars within their Personal Cloud. An API Sync has been developed which allows all of these functions to be developed, replacing the U1DB synchronisation API previously used for this purpose.

3.2 eyeOS integration with Personal Clouds

3.2.1 Integration

The eyeOS platform integrates the services of the Personal Cloud in the file manager, which allows users to go online and see files they have saved in the Personal Cloud, create directories, move files, share documents, etc.

This file manager is developed in JavaScript, HTML, and CSS, allowing users to directly view their files in the browser. The interface is similar to any file manager in any operative system. Users can carry out different operations on files in a very intuitive manner.

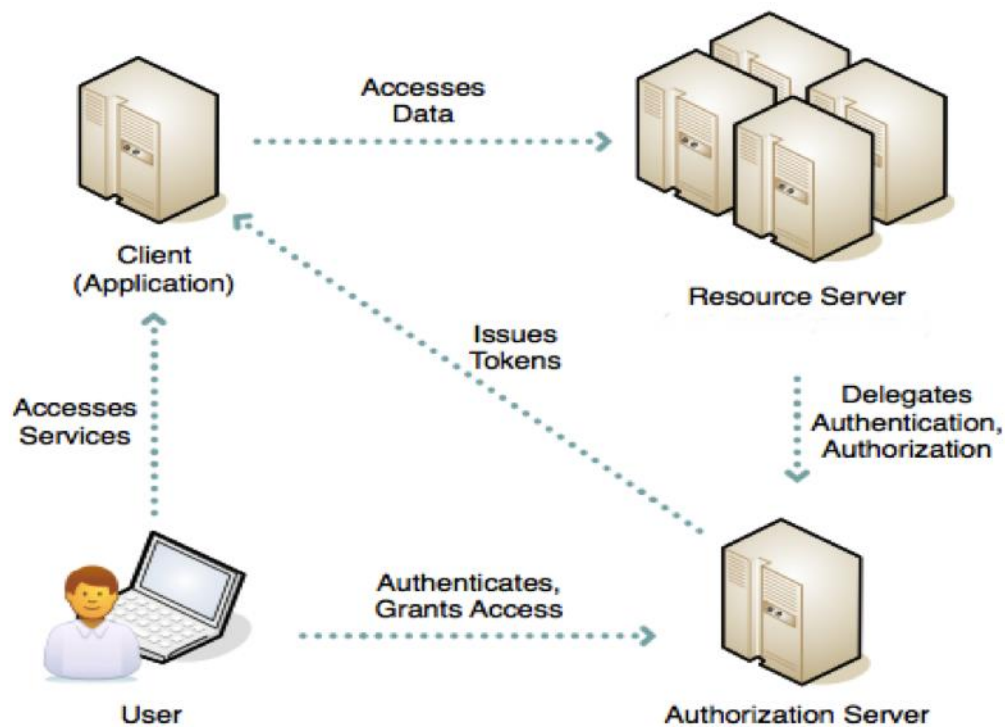


Once the user is in CloudSpaces they can access their protected data by selecting a specific and existing cloud in the clouds configuration of eyeOS (see annex 1) which is listed in the “Clouds” tab to configure access.

3.2.2 Authorization and authentication (log in/out)

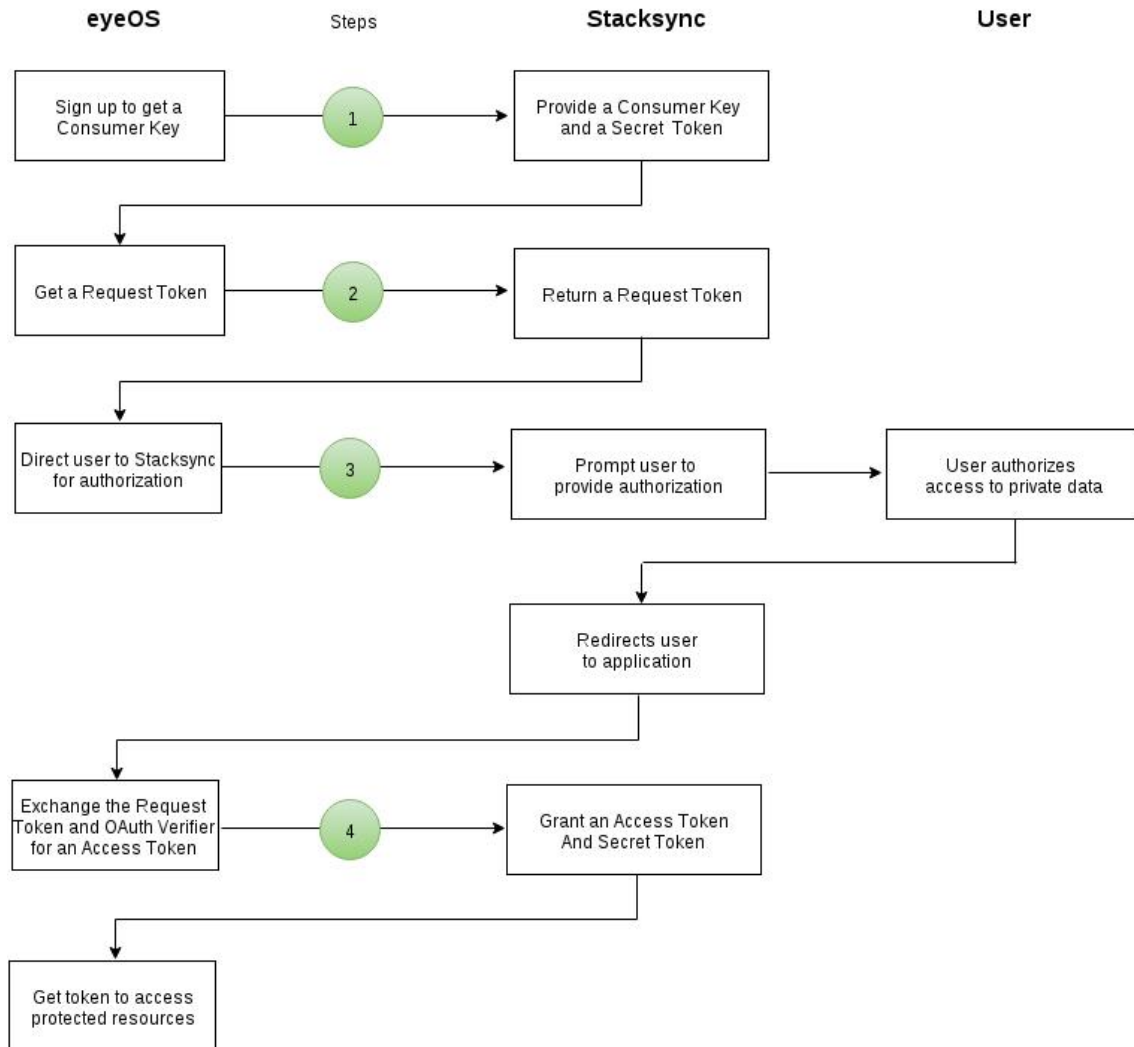
The eyeOS platform integrates the OAuth authorization to interact with the user’s protected data stored in the Personal Cloud. OAuth is an authorization protocol which allows the user (owner of certain resources) to authorize eyeOS to access

said resources in their name but without having to at any time give their authentication credentials; that is, without giving their name and password.



The first time that access is requested to the resources of a specific user of a Personal Cloud, authentication is used to obtain a security token for that same cloud, which allows for interaction with the user's data. The keys are stored in the "token" table of the DBMS database based in MySQL. These keys are linked to the cloud and user who started the session in the platform, so the system can determine the access token for a specific user trying to use the services at any stage.

Below the communication dialog box is shown:



Step 1:

Request the key and secret token from StackSync that identifies eyeOS as a resource user of CloudSpaces. This communication was carried out via email.

Step 2:

Request the token and provide Stacksync with the forwarding URL to eyeOS once the user has given authorization.

StackSync responds to the aforementioned request with a valid token and authorization URL.

Step 3:

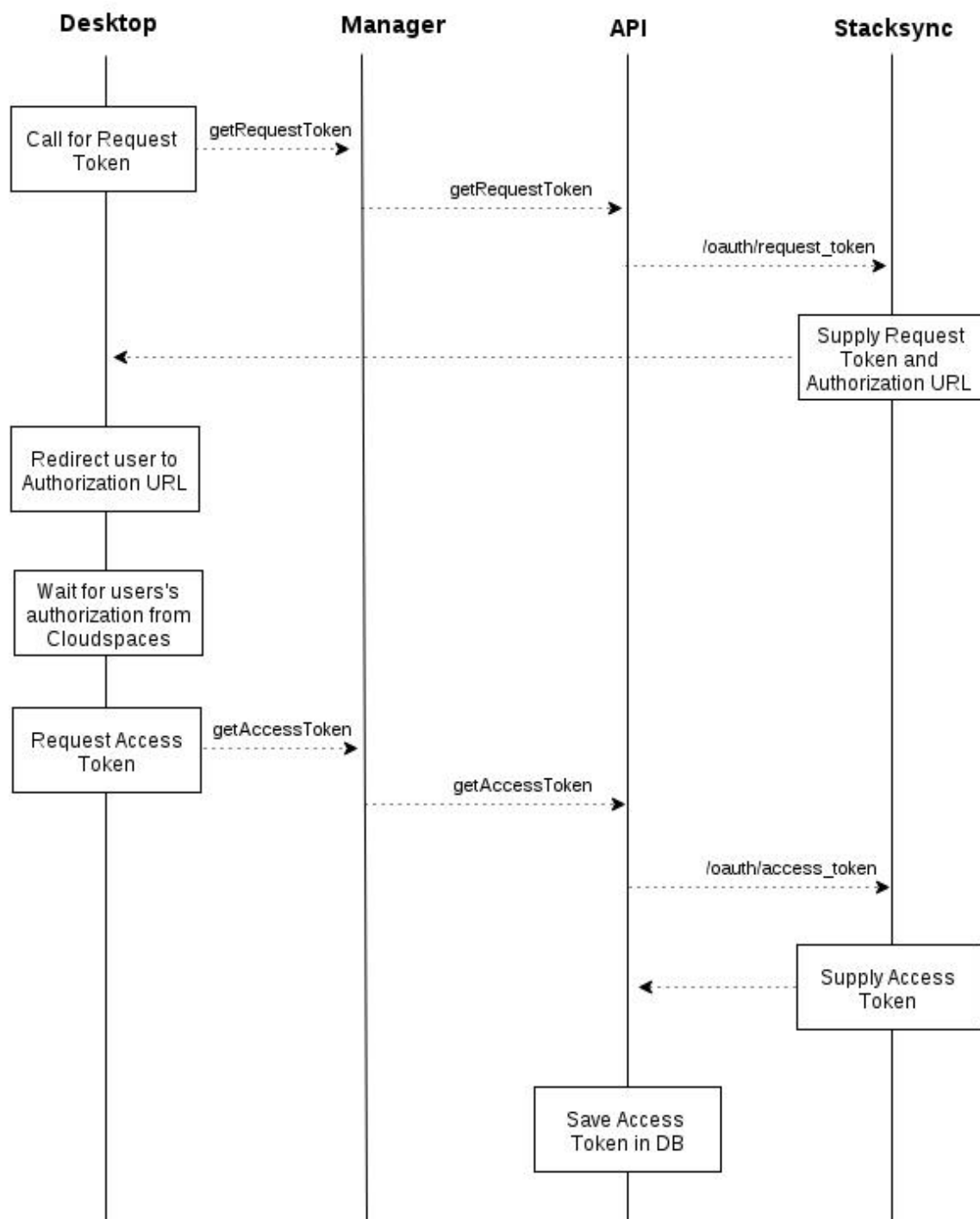
Redirect the user to the authorization URL where the user gives eyeOS access to their private area.

Once StackSync verifies the user, the user is redirected to the eyeOS URL provided in the previous step.

Paso 4:

Request the access token and secret token from StackSync, which will be used to identify eyeOS when it accesses the private area of the user in CloudSpaces.

The implementation of the authentication in eyeOS is detailed in the following diagram:

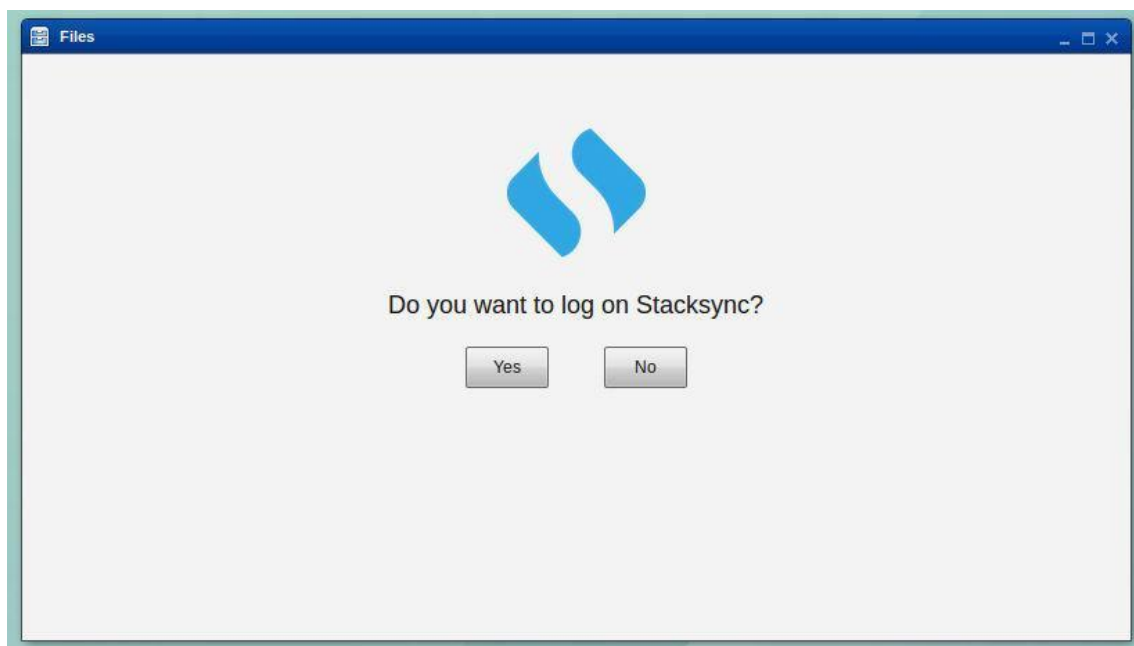


The Oauth Manager and Oauth API functions are detailed in annexes 2 and 3 respectively.

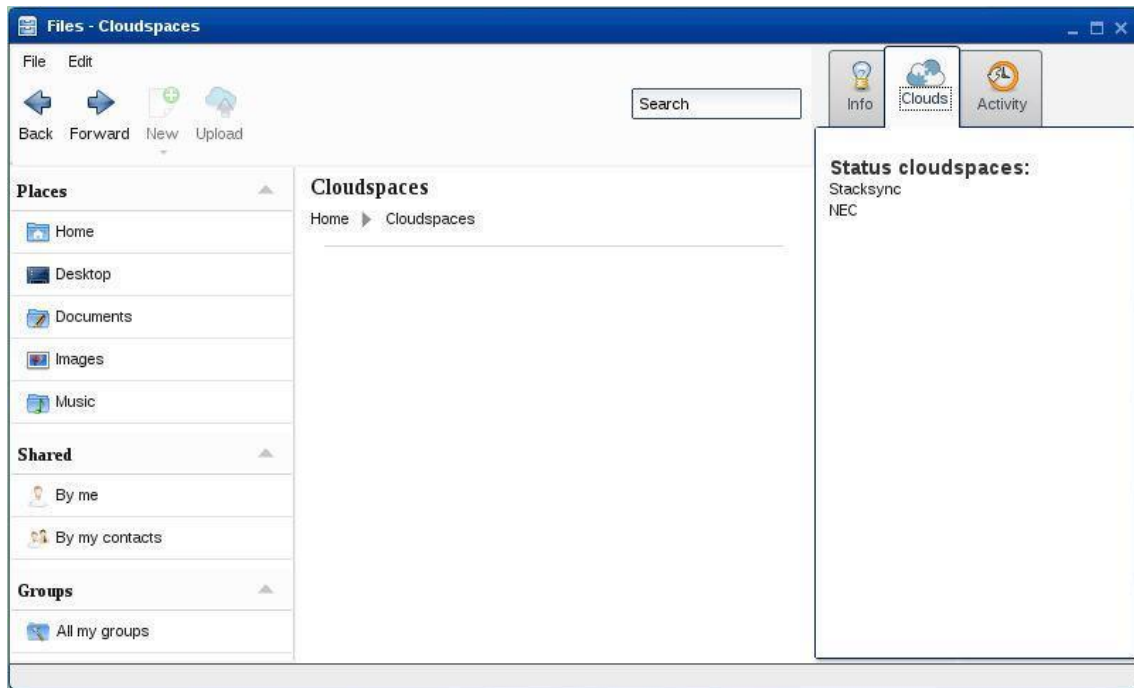
Below is a visual representation of the authorization process for allowing eyeOS access to the user's private area in a private or public cloud. This process is only carried out once, when the platform does not have the security token of the active user:

Access the "CloudSpaces" directory from Home and select the cloud for which access is going to be authorized to eyeOS, for example, StackSync, from the "Clouds" tab located on the bar on the right (social bar).

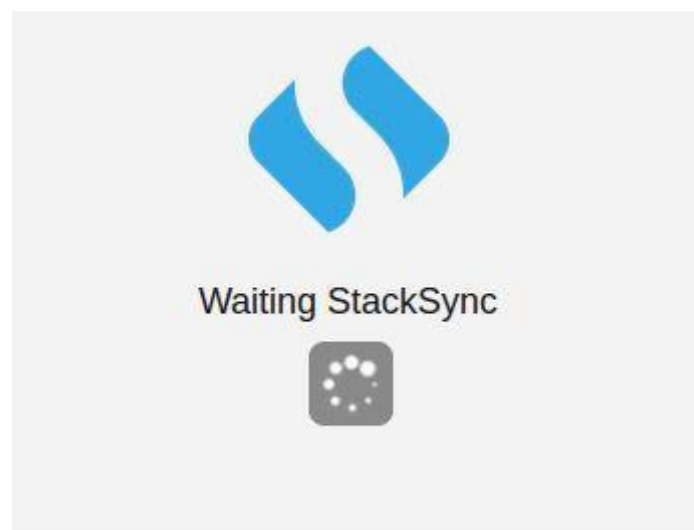
The user is asked whether they provide their credentials to allow eyeOS to access their protected data.



- If the user selects "No", the eyeOS file structure is shown without displaying the directory of the "StackSync" cloud.



- If the user selects “Yes” in the first screen of the process, communication with StackSync begins to obtain the accesstoken.

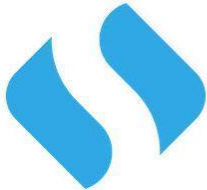


- A new window is opened in the browser, which redirects the user to the authorization URL received from StackSync, where the user gives eyeOS access to their private area.

api.stacksync.com:8080/oauth/authorize?oauth_token=SpieqFLYQw3LAawCWlWxqlSm8eFZoa

StackSync

The following application requests access to your files:
EyeOS Test Application
This is a test application for EyeOS



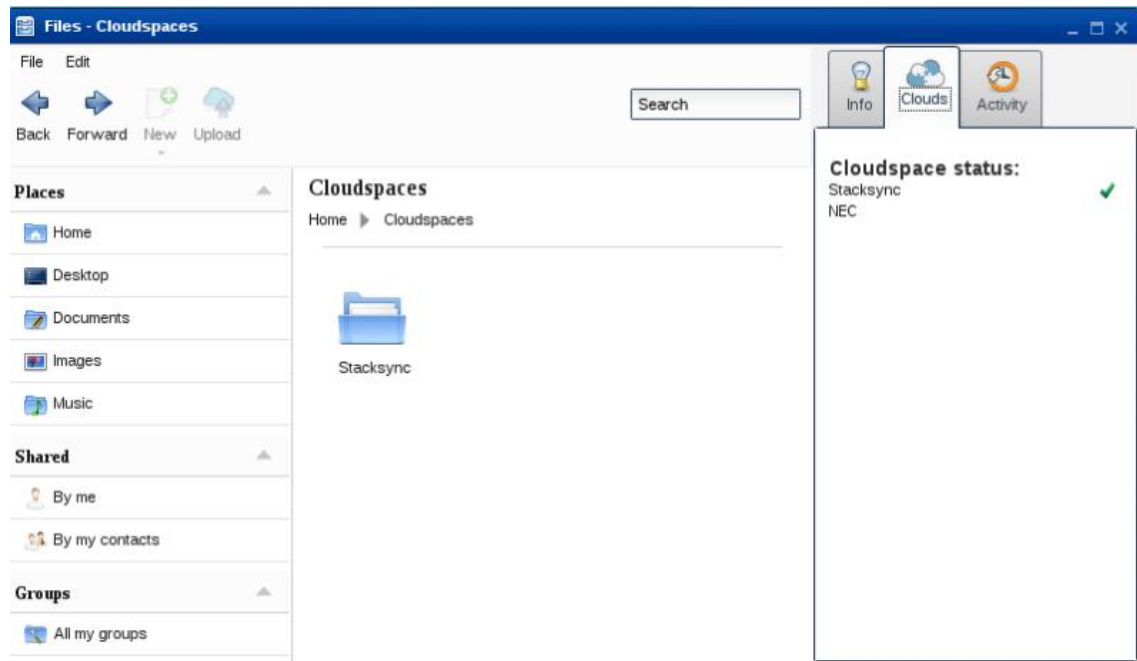
☒ Allow access to this app
☐ Deny access to this app

Submit

- Once access to StackSync has been given, the user is redirected to the URL provided by eyeOS to obtain the requesttoken. The page informs the user that the process has been successfully completed and they can return to the eyeOS desktop.

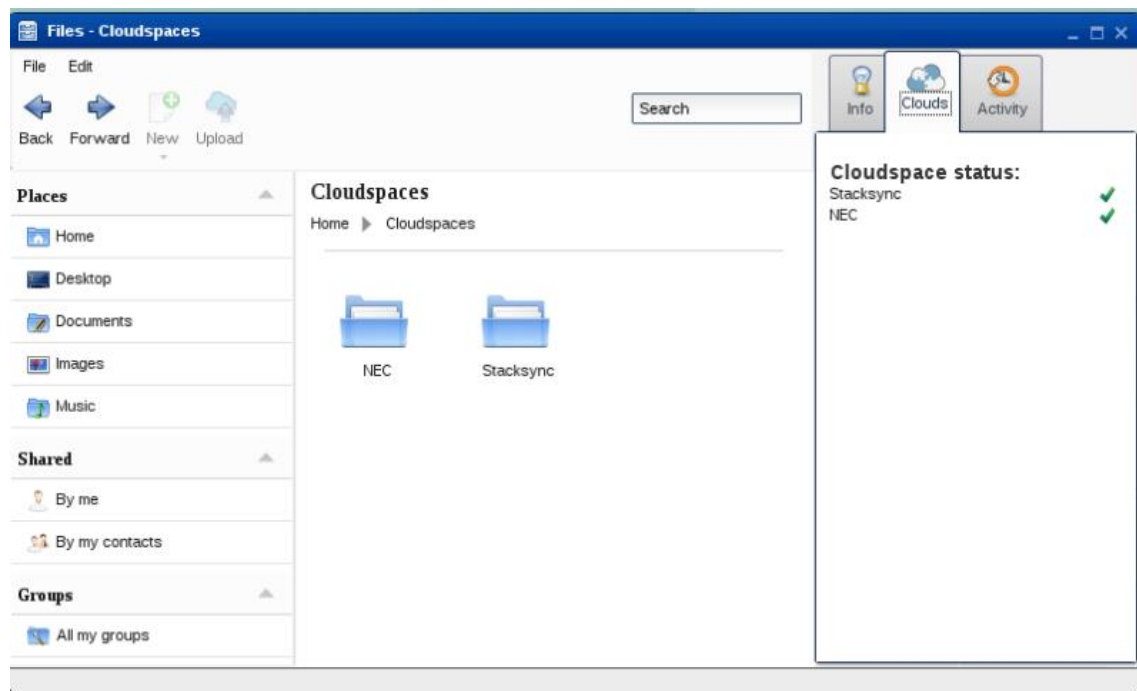


- The accesstoken is saved for the current user of eyeOS. From that moment the user can access their protected data from the StackSync directory without having to log in again.

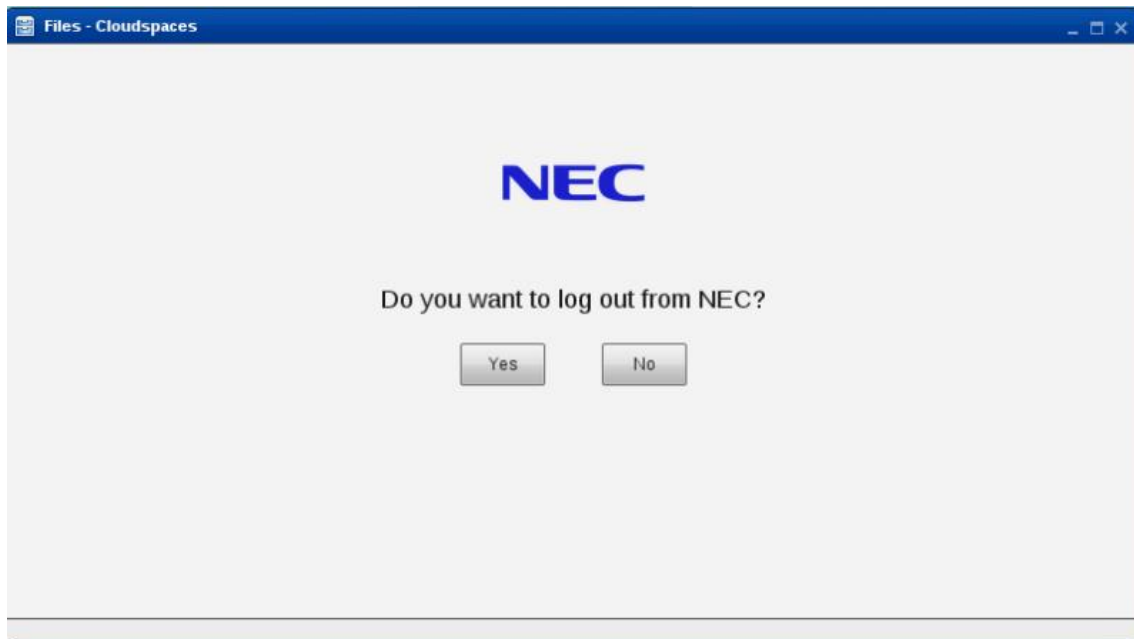


If the cloud appears active in the list and its folder is present in “CloudSpaces”, access can be removed by selecting the name again from the list in the “Clouds” tab. The de-authorization process is started on the following screens:

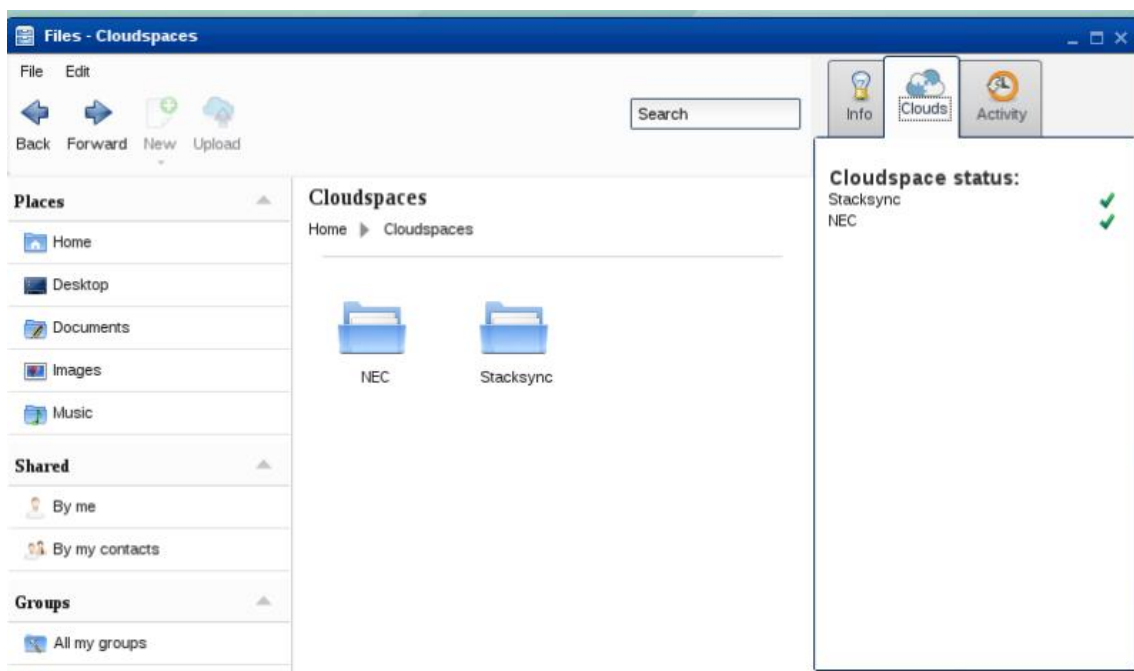
- Access the “CloudSpaces” directory from Home and select the cloud for which eyeOS access is to be removed, for example NEC, from the “Clouds” tab located on the right bar (social bar).



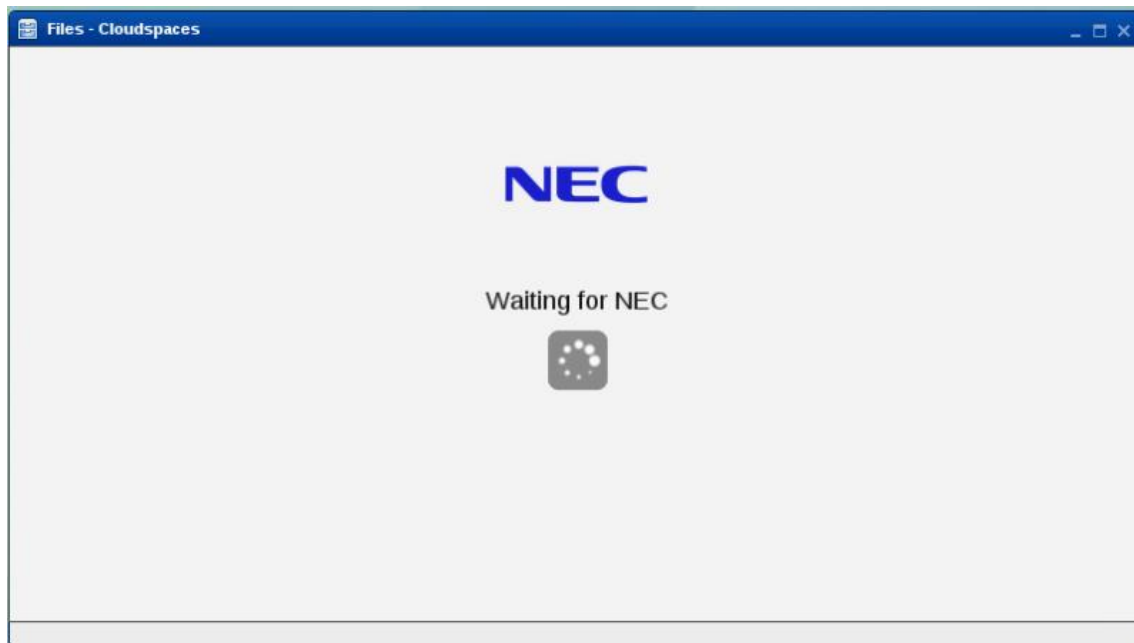
- The user is asked whether they want to close access with NEC.



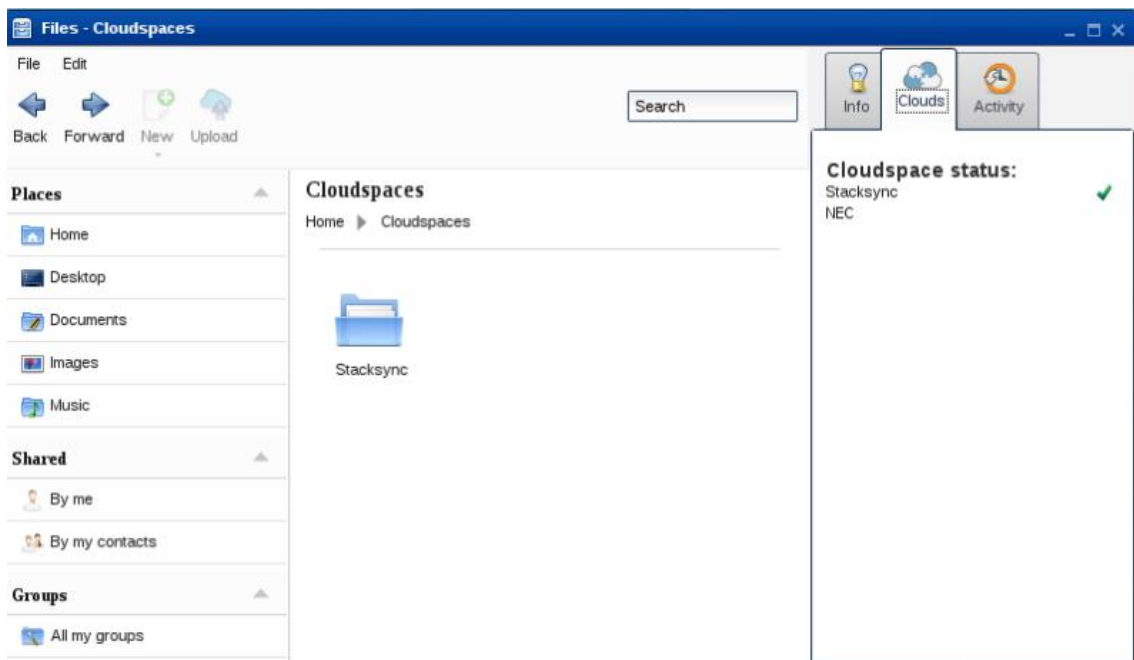
- If the user selects “No”, the file structure of eyeOS is shown with the “NEC” cloud directory.



- If the user selects “Yes” in the initial screen of the process, the process of deleting the data related to the specific user and cloud begins.



- The accesstoken of the current eyeOS user is deleted. From that moment the user cannot access their protected data from the NEC directory without starting the authentication process.

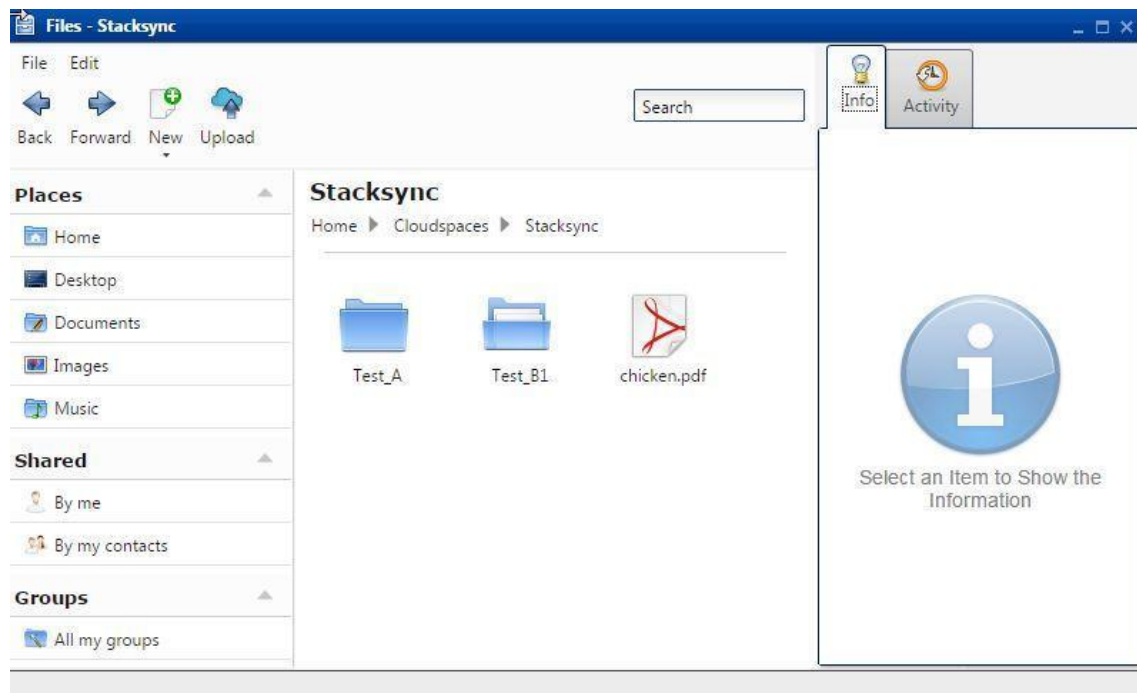


3.2.3 Storage API

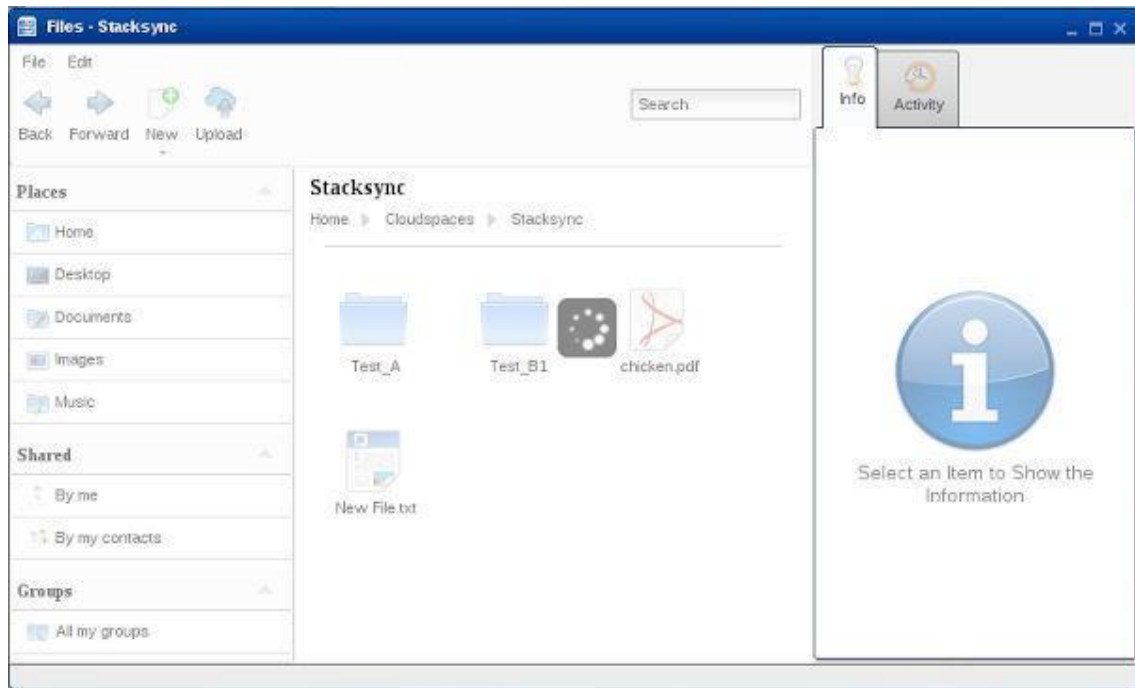
Once access to the cloud has been established, the private data can be accessed from eyeOS by sending queries to the Storage API. This returns metadata with

all the structural information regarding directories and files, which uses eyeOS to generate a local replica.

The directories and files are created without content. Once an element is selected in any operation is carried out, such as opening, moving, or copying, the content is then downloaded from it. This process manages not to overload the system, recovering information that the user is not going to use at that moment. If a file or directory has been open before and no change has been made, it will not be updated.



The content of the current directory is synchronized with the Personal Cloud directory through a process carried out in the background, which sends queries every 10 seconds to check whether any changes have been made. If there are changes the current structure is updated.



The API Storage of the Personal Cloud is essential in the file manager, because it provides different services which allow the user to carry out operations with their files.

The importance of the API Storage will be better understood with the following example, where the user accesses the Comments folder in StackSync and creates the folder New Folder.

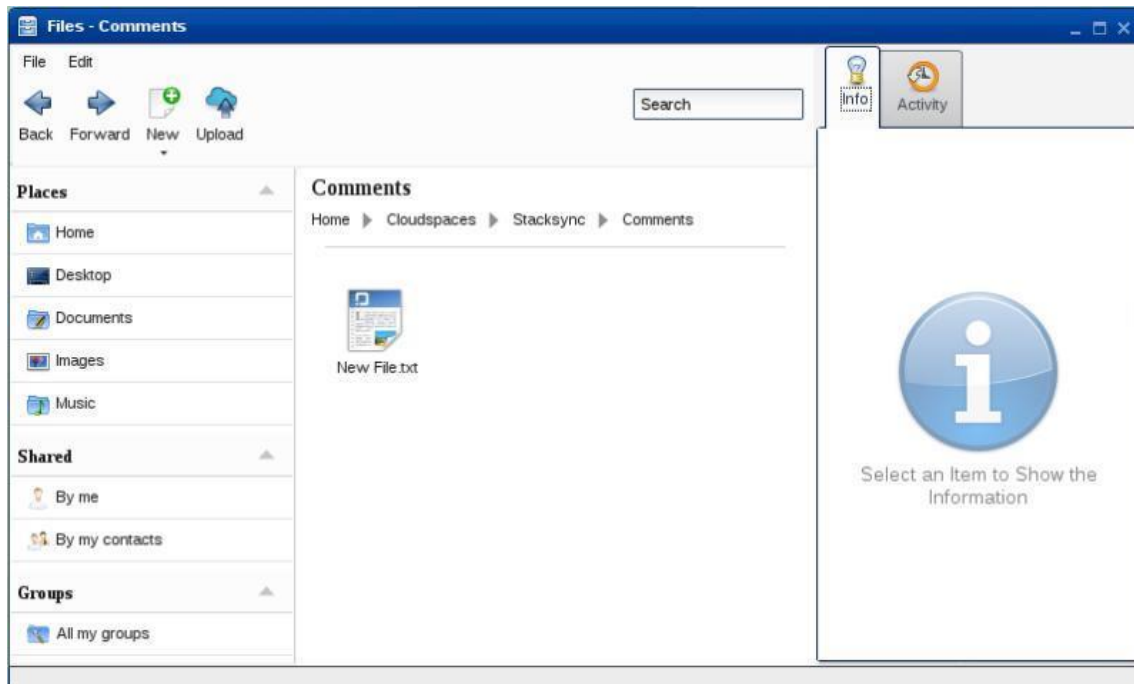
A call is made with the GET method using a token that is valid in StackSync to check the content of the Comments folder (id: 2053).

URL parameters: {"id": "2053"}

URL: <http://api.stacksync.com:8080/v1/folder/:id/contents>

It returns metadata with the files in the Comments folder.

```
{"status": "RENAMED", "mimetype": "inode/directory", "checksum": 0,
"modified_at": "2015-06-12 09:53:55.312", "filename": "Comments", "is_root":
false, "parent_id": "null", "version": 2, "is_folder": true, "id": 2053, "contents":
[{"status": "CHANGED", "mimetype": "text/plain", "checksum": 296026785,
"modified_at": "2015-06-15 12:59:57.414", "filename": "New File.txt",
"parent_id": 2053, "version": 2, "is_folder": false, "chunks": [], "id": 2059,
"size": 10}], "size": 0}
```

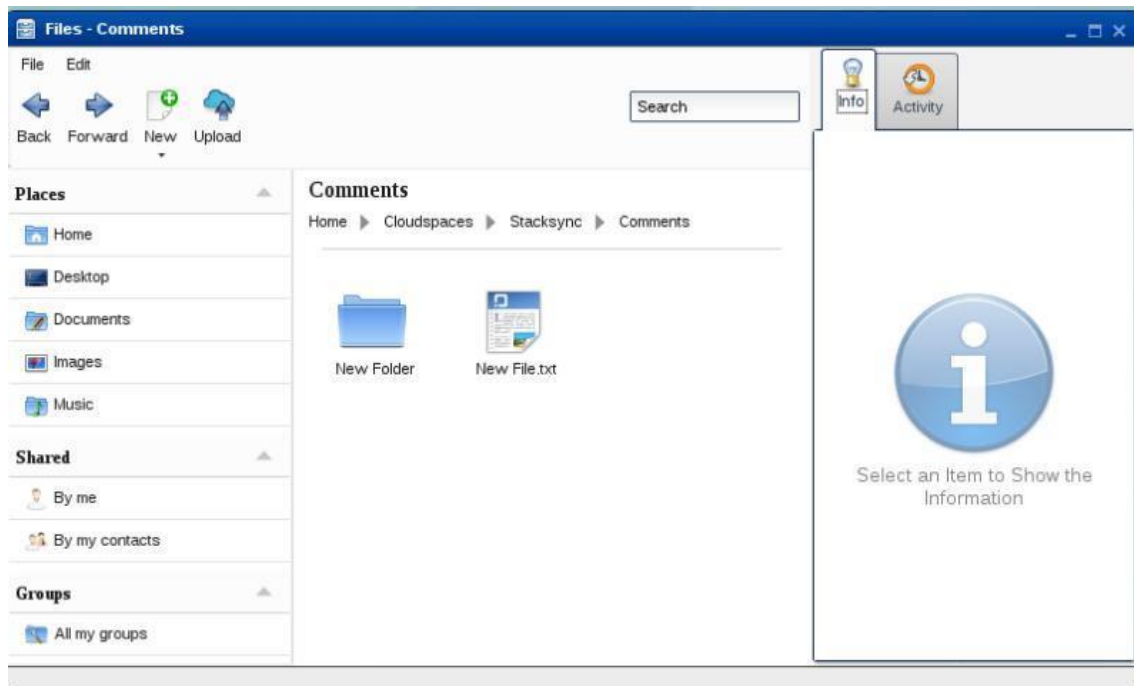
A call is made with the POST method to the Storage API to create the folder New Folder within the Comments folder (id: 2053).

URL: <http://api.stacksync.com:8080/v1/folder>

POST: {"name": "New Folder","parent":2053}

The following metadata confirms that the folder has been created correctly:

```
{ "status": "RENAMED", "mimetype": "inode/directory", "checksum": 0,
  "modified_at": "2015-06-12 09:53:55.312", "filename": "Comments", "is_root":
false, "parent_id": "null", "version": 2, "is_folder": true, "id": 2053, "contents":
[{"status": "CHANGED", "mimetype": "text/plain", "checksum": 296026785,
  "modified_at": "2015-06-15 12:59:57.414", "filename": "New File.txt",
  "parent_id": 2053, "version": 2, "is_folder": false, "chunks": [], "id": 2059,
  "size": 10}, {"status": "NEW", "mimetype": "inode/directory", "checksum": 0,
  "modified_at": "2015-09-05 12:13:55.107", "filename": "New Folder", "is_root":
false, "parent_id": 2053, "version": 1, "is_folder": true, "id": 2062, "size": 0}],
  "size": 0}
```



Annexes 4 and 5 detail the queries made by the file manager to the Storage API to allow the user to carry out different actions in their files.

3.3 Personal Clouds and eyeOS interoperability

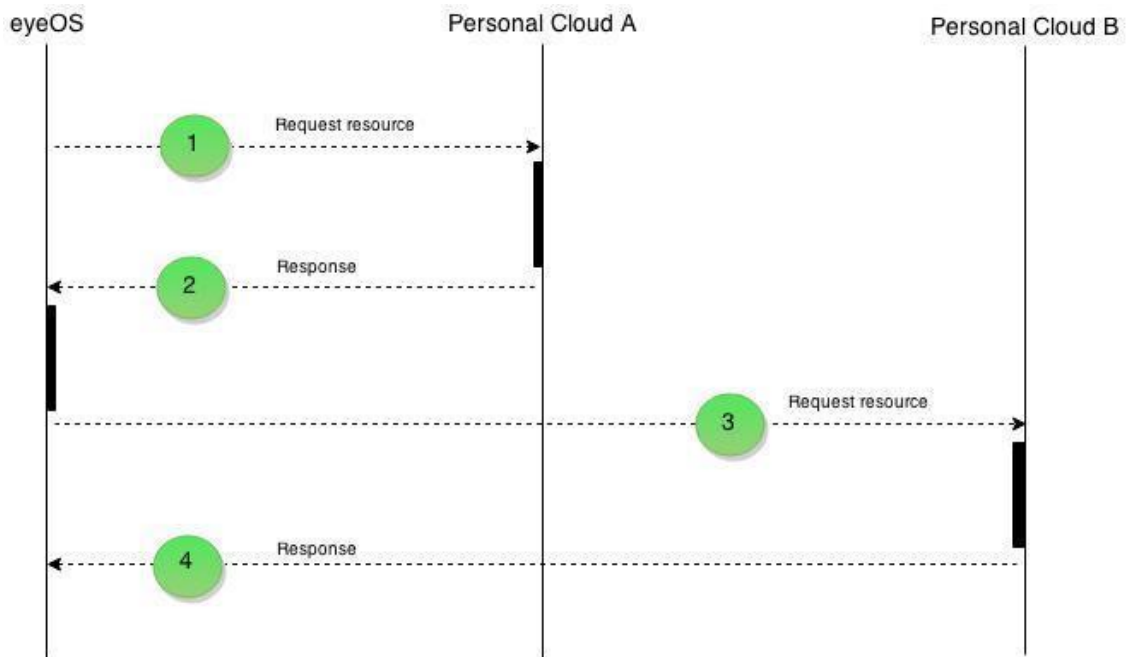
3.3.1 Implementation

The eyeOS platform provides access to multiple clouds as well as interoperability between them, both in a private and public environment.

A possible definition of interoperability would be: “It is the capacity of information systems and the procedures which they support to share data and enable the exchange of information and knowledge between them”.

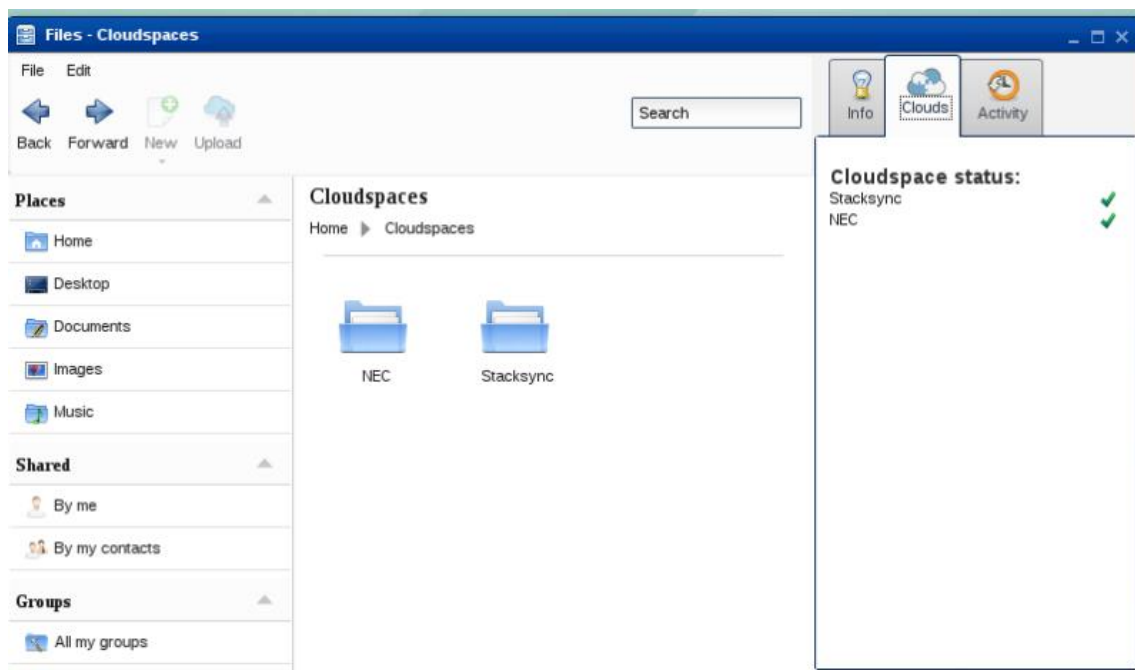
Interoperability is the condition which allows different systems or products to relate to each other to exchange data. The aim is to be able to share information with any user regardless of the system that they use.

The implementation of interoperability is detailed in the following diagram, with the exchange of information between two Clouds (Personal Cloud A and Personal Cloud B) and the eyeOS platform:



Step 1:

The user makes a request to Personal Cloud A with the URL and a valid ACCESS TOKEN from Personal Cloud A, to list all of the directories and files that a directory identified with a specific ID contains



Step 2:

Personal Cloud A returns a structure of directories and files. In this structure it is possible to find directories shared with Personal Cloud B. The feature which

identifies these directories is a URL and a valid ACCESS TOKEN from Personal Cloud B.

Step 3:

To list the content of this directory, the user makes a request to the API of Personal Cloud B with the URL, directory ID, and ACCESS TOKEN received in Step 2.

Step 4:

Personal Cloud B returns a structure of directories and files corresponding to the directory specified in the URL. From this point, all requests implemented in the Storage API which are carried out in this directory will use the URL and the ACCESS TOKEN of Personal Cloud B.

3.3.2 Examples

The following example details how a StackSync user creates a folder called New Folder in the folder interop1 shared by another NEC user:

A call is made with the GET method using a valid StackSync token to check the content of the StackSync root folder:

URL parameters: {"id": "0"}

URL: <http://api.stacksync.com:8080/v1/folder/:id/contents>

It returns metadata with the files in the StackSync root folder. Within this metadata there are new tags: resource_url, access_token_key and access_token_secret. These tags show us that the folder is shared with another NEC user and it provides the connection URL and a token to be able to make calls to the Storage API of NEC:

```
{"status": "null", "mimetype": "null", "checksum": "null", "filename": "root",  
"is_root": true, "parent_id": "null", "version": "null", "is_folder": true, "id":  
"null", "contents": [{"status": "NEW", "mimetype": "application/pdf",  
"checksum": 2230714779, "modified_at": "2015-03-27 16:46:33.243",  
"filename": "chicken.pdf", "parent_id": "null", "version": 1, "is_folder": false,
```

```
"chunks": [], "id": 1587, "size": 51500},{ "status": "RENAMED", "mimetype":  
"inode/directory", "checksum": 0, "modified_at": "2015-04-23 12:11:09.351",  
"filename": "interop1", "is_root": false, "parent_id": "null", "version": 2,  
"is_folder": true, "id": 1972, "size": 0, "resource_url":  
"http://csdev.neccloudhub.com:1080/api/cloudspaces/", "access_token_key":  
"e7e2b8e-14bc-4a75-942b-d757fe7035da", "access_token_secret": "30c60a55-  
0b50-4262-9720-c50e0e3489f0"}], "size": "null"}
```

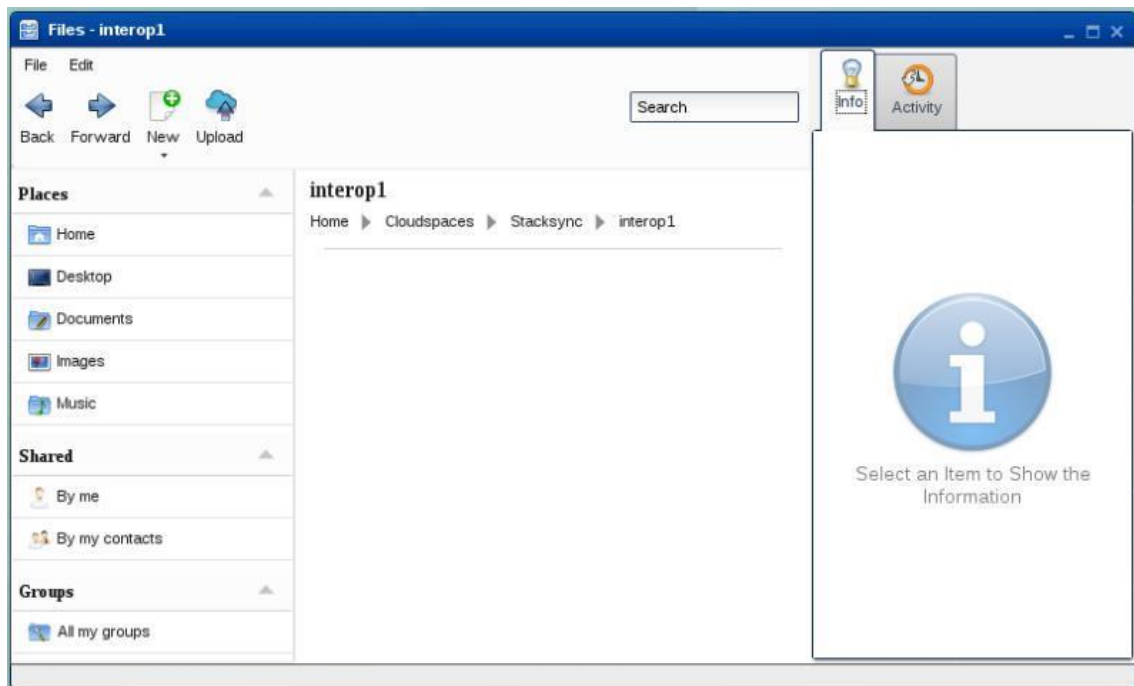
A call is made with the GET method using the URL and token of NEC recovered in the previous query to find out the content of the folder interop1 (id: 1972):

URL parameters: {"id": "1972" }

URL: http://csdev.neccloudhub.com:1080/api/cloudspaces/folder/:id/contents

It returns metadata with the files contained in the folder interop1 of NEC:

```
{"status": "RENAMED", "mimetype": "inode/directory", "checksum": 0,  
"modified_at": "2015-06-12 09:53:55.312", "filename": "interop1", "is_root":  
false, "parent_id": "null", "version": 2, "is_folder": true, "id": 2053, "contents":  
[], "size": 0}
```



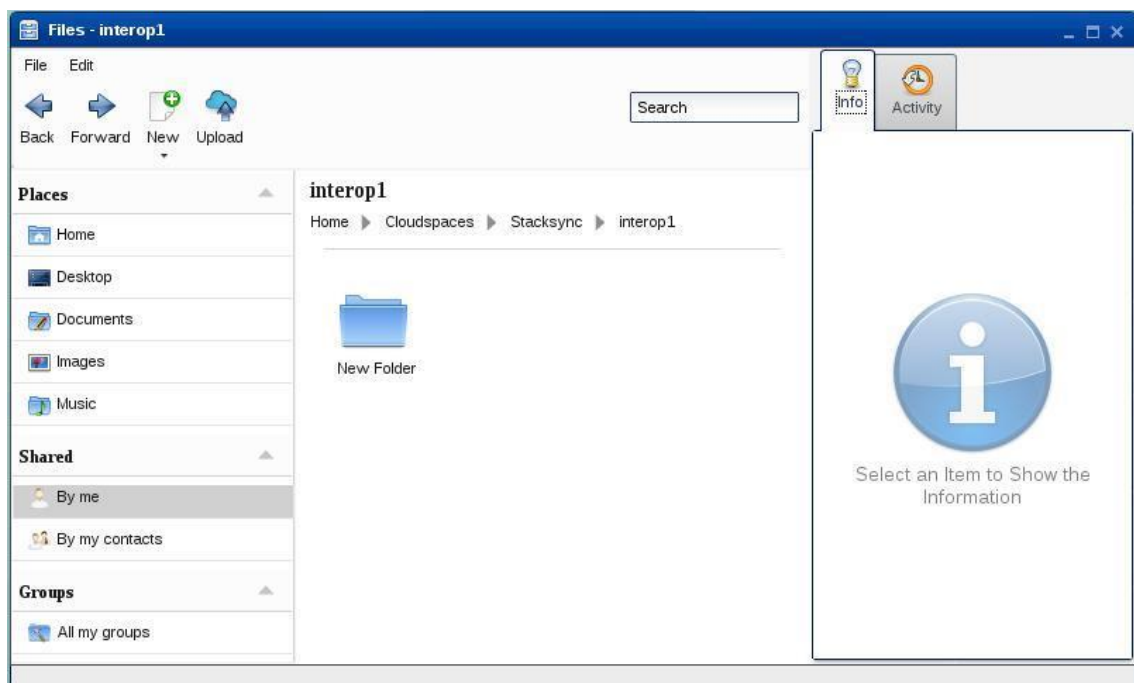
A call is made using the POST method with the token obtained from the first query to create the folder New Folder within the folder interop1 (id: 1972).

URL: <http://csdev.neccloudhub.com:1080/api/cloudspaces/folder>

POST: {"name": "New Folder","parent":1972}

The following metadata confirms that the folder has been created correctly:

```
{"status": "RENAMED", "mimetype": "inode/directory", "checksum": 0,
"modified_at": "2015-04-23 12:11:09.351", "filename": "interop1", "is_root":
false, "parent_id": "null", "version": 2, "is_folder": true, "id": 1972, "contents":
[{"status": "NEW", "mimetype": "inode/directory", "checksum": 0,
"modified_at": "2015-09-06 13:04:03.15", "filename": "New Folder", "is_root":
false, "parent_id": 1972, "version": 1, "is_folder": true, "id": 2063, "size": 0}],
"size": 0}
```



The user can also share or stop sharing any directory with users from the same or different Personal Clouds. Using the options of the context menu and the activity tab of the Social Bar, the user can manage the directories that they share, deciding at each moment which users can or cannot have access to them.

The following example details the process of how the eyeos user shares the folder share with the user tester1 from the same Personal Cloud as them (StackSync):

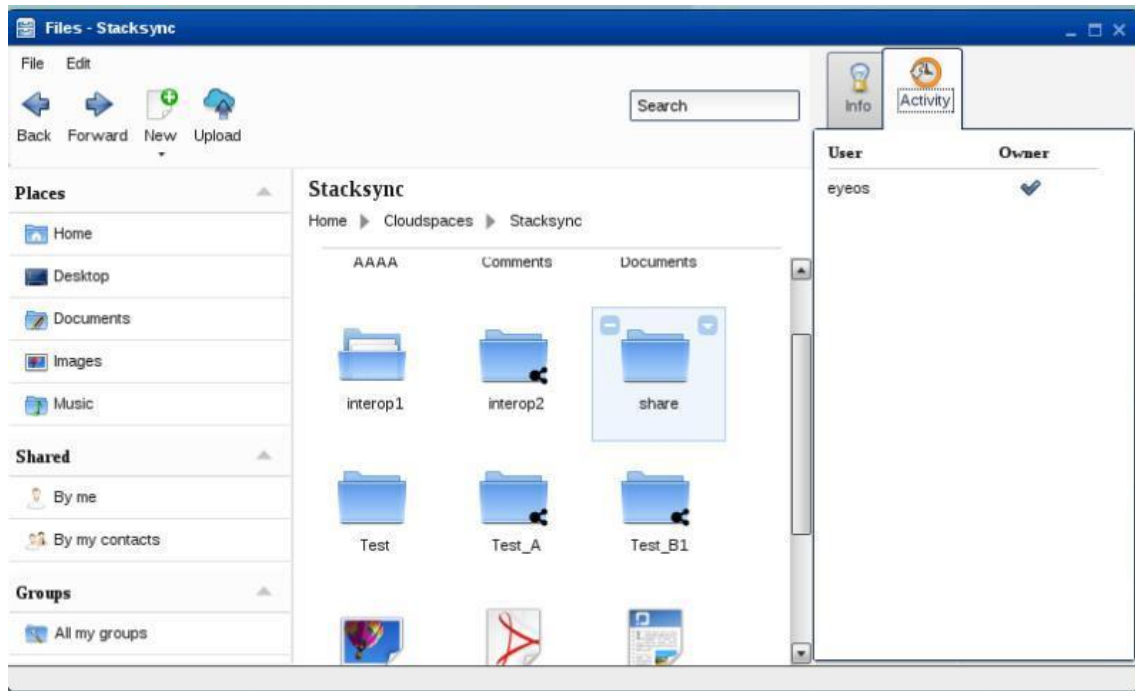
A call is made with the GET method using a valid StackSync token to obtain information of the users who share the folder share (id: 1973):

URL parameters: { "id": "1973" }

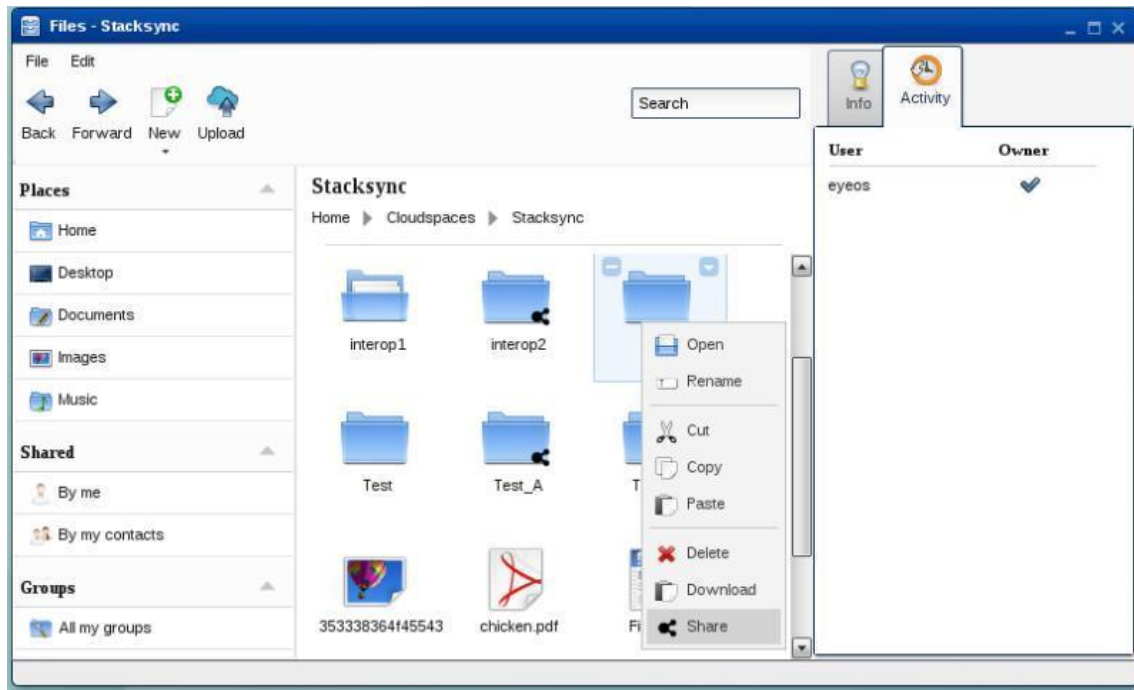
URL: <http://api.stacksync.com:8080/v1/folder/:id/members>

It returns metadata with the users who share that folder, in this case since it is not shared, it only returns the owner user:

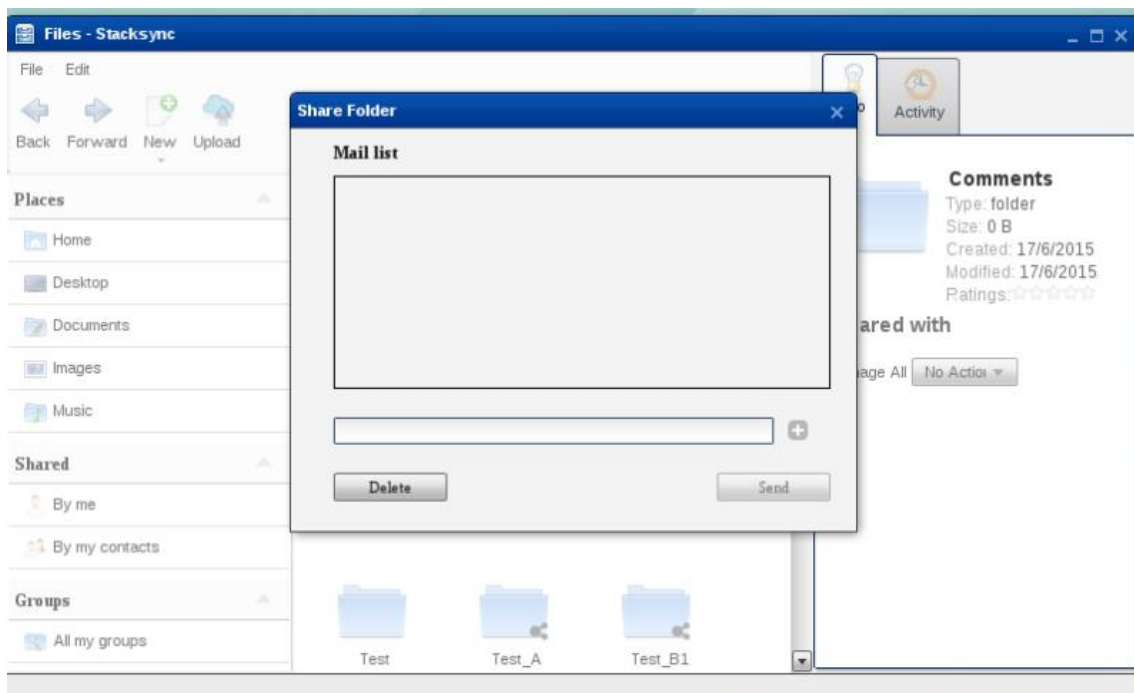
```
[{"joined_at": "2015-03-27", "is_owner": true, "name": "eyeos", "email": "eyeos@test.com"}]
```



- The share option is selected from the context menu to share the folder share with the user tester1:



- The email of the user tester1 is added so that StackSync can share this folder with this user:



In a POST call the list of emails is sent of the users with whom the folder is going to be shared:

URL parameters: { "id": "1973" }

URL: http://api.stacksync.com:8080/v1/folder/:id/share

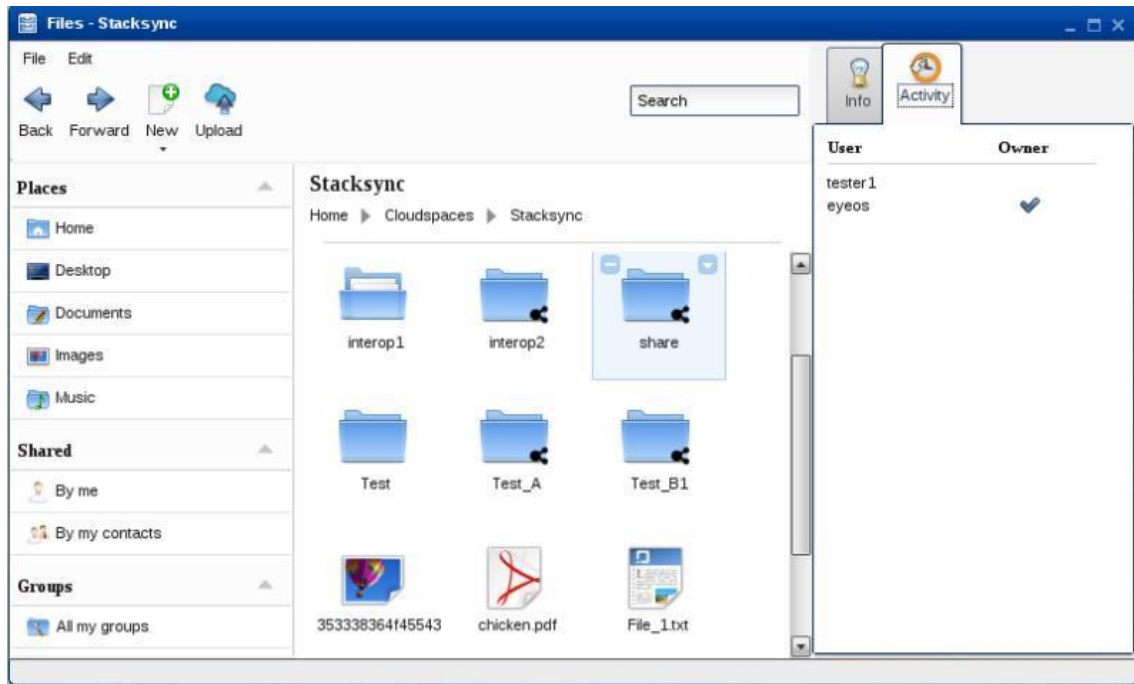
POST: { ["tester1@test.com"] }


When doing another GET call to check which users share the folder, it can be seen that it has been shared with the user tester1 in the tab Activity of the Social Bar:

URL parameters: { "id": "1973" }

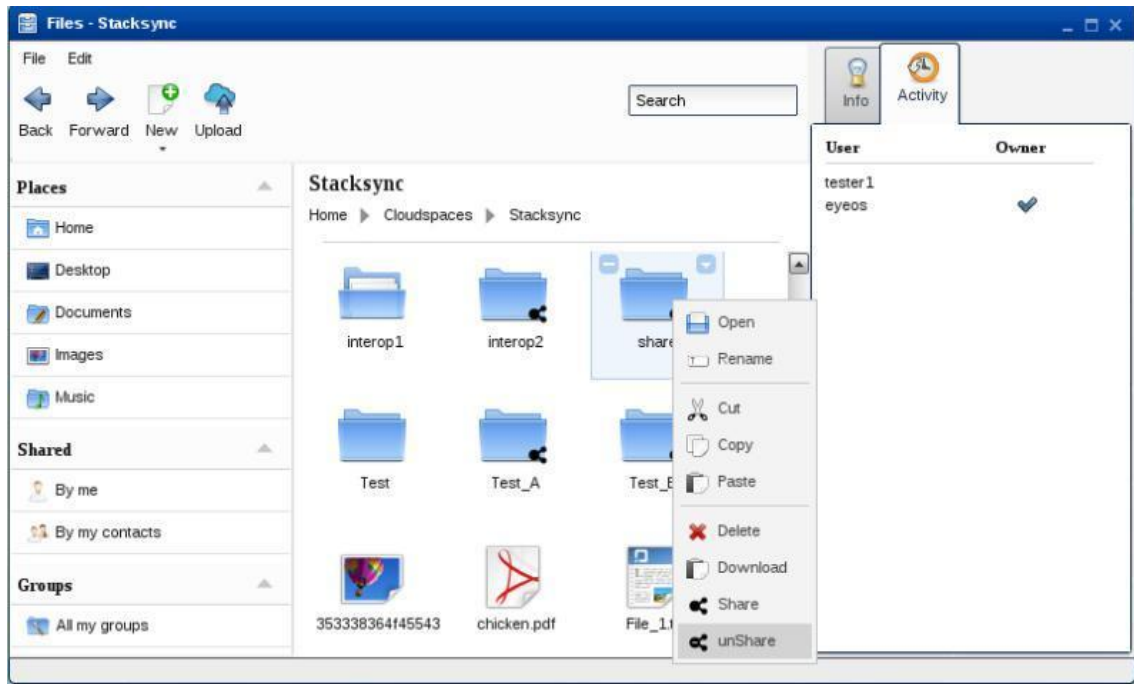
URL: http://api.stacksync.com:8080/v1/folder/:id/members

```
[{"joined_at": "2015-03-27", "is_owner": true, "name": "eyeos", "email": "eyeos@test.com"}, {"joined_at": "2015-03-27", "is_owner": false, "name": "tester1", "email": "tester1@test.com"}]
```

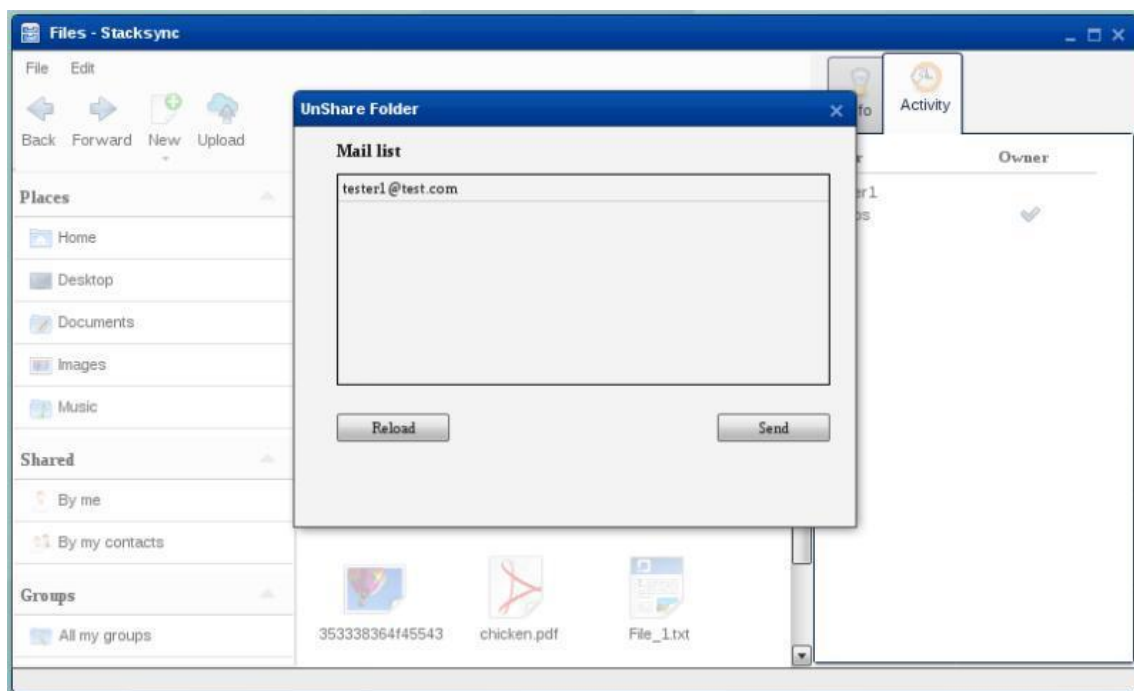


The shared directories are identified in the Files Manager with  the icon

- The unShare option is selected from the context menu to stop sharing the folder share with the user tester1:



- Once “unShare” has been selected, a form is displayed to select the users with whom the directory is no longer going to be shared. The directory cannot be unshared with the owner of said directory, so they are not shown on the list.



In a POST call the list of emails is sent of the users with whom the folder is going to be unshared:

URL parameters: { "id": "1973" }

URL: http://api.stacksync.com:8080/v1/folder/:id/unshare

POST: { ["tester1@test.com"] }

If all of the users are removed from the list, the directory will no longer be available for those users and the "unshare" option in the context menu will disappear.

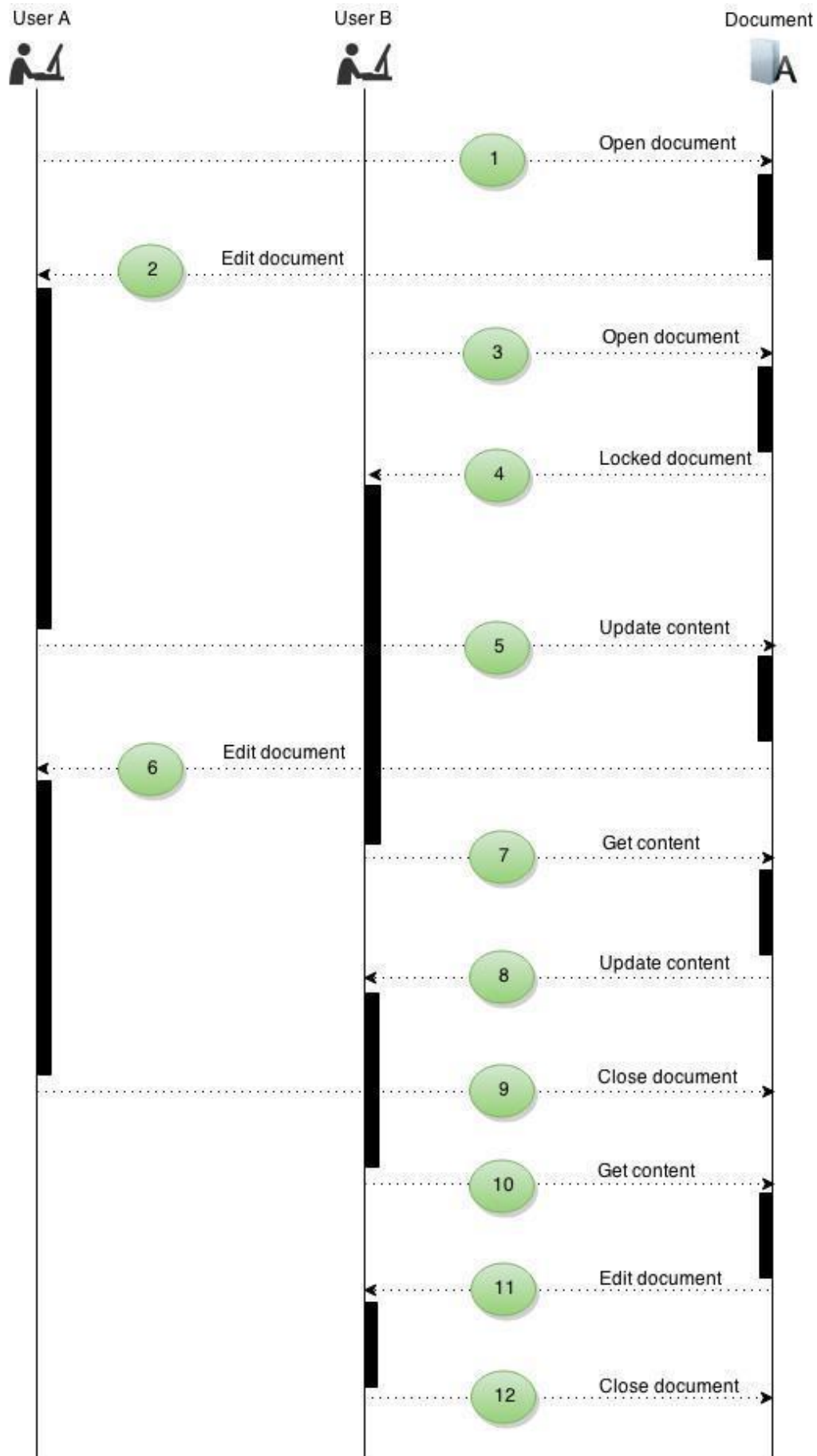
3.4 Collaborative editing tool

The eyeOS platform offers the native eyeDocs app, a word processor which allows users to create and edit documents in the eyeOS format. Since it is a Cloud application, text editing with eyeDocs can be done from any computer without pre-installed local programs. Like other Cloud-based text editors, it does not guarantee that the formatting of documents with sophisticated borders and tables will be preserved. Therefore, far from competing with applications such as Microsoft Word, eyeDocs is optimized to load quickly and be a comfortable tool to create documents and take notes.

It is a tool made for collaborating with functions that allows different users to edit or see the content of a document at the same time. These functions are only available for users of eyeOS who share a single document in the cloud.

The collaboration between users is achieved by linking the data of the editing user to the file, in a server synchronized with other eyeOS platforms. When a user opens a file, a query is sent to a Sync API which collects information from the server to indicate its current state.

The implementation of the collaboration between users to edit a document is detailed in the following diagram:



Step 1:

The user stacksync (User A) opens the document eyeos2.edoc (id: 1980) located in the folder Comments of StackSync. Two checks are made:

- The document is not being used by any user.
- The document is being used by another user, but the date and time of the last update is greater in minutes than the time specified in settings.php in the root of the project.

In both cases it is blocked by the current user.

To configure the maximum blocking time of the file, the constant TIME_LIMIT_BLOCK in settings.php should be modified. In the following example it is set at 10 minutes:

```
If (!defined('TIME_LIMIT_BLOCK') define('TIME_LIMIT_BLOCK',10);
```

To block the file the user stacksync (User A) makes a call with the POST method using a valid StackSync token to the Sync API:

URL: <http://api.stacksync.com:8080/v1/lockFile>

POST: {"id": "1980", "cloud": "Stacksync", "user": "stacksync", "ipserver": "192.68.56.101", "datetime": "2015-05-12 10:50:00", "timelimit": 10}

Step 2:

The user stacksync (User A) gets permission to edit the requested document.

```
{"lockFile": true}
```

Step 3:

The user stacksync2 (User B) opens the same document that the user stacksync (User A) is editing. It is confirmed that the user does not have permission to edit the document.

The same call is made as in Step 1:

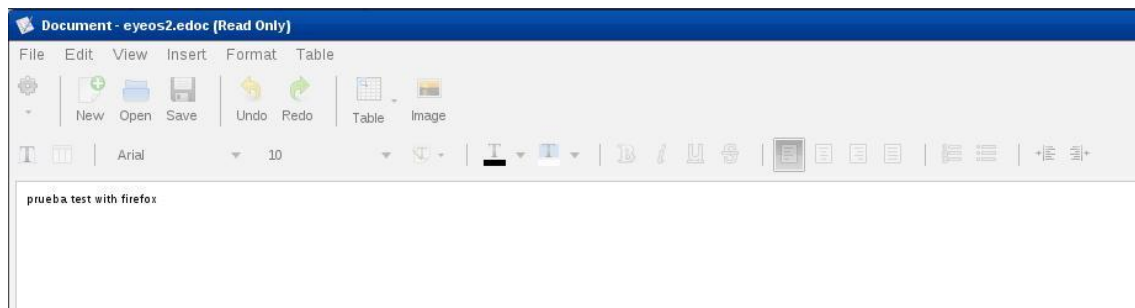
URL: http://api.stacksync.com:8080/v1/lockFile

POST: {"id":"1980", "cloud": "Stacksync", "user": "stacksync2", "ipserver":
"192.68.56.101", "datetime": "2015-05-12 10:55:00", "timelimit":10}

Step 4:

The user stacksync2 (User B) is informed that they do not have permission to edit the document:

{"lockFile":false}



In the eyeDocs application this document will be identified with the label '(Read Only)' in the title bar and the Menu, Toolbar, and document container will be blocked.

Step 5:

The user stacksync (User A) updates the content of the document. Each content update causes the time and date of the last document update to change to current values.

The user stacksync (User A) makes a call using the PUT method to the Sync API to update the time and date of the document:

URL: http://api.stacksync.com:8080/v1/updateTime

PUT: {"id":"1980", "cloud": "Stacksync", "user": "stacksync", "ipserver":
"192.68.56.101", "datetime": "2015-05-12 10:57:00"}

It returns metadata showing that the update has been made:

{"updateFile":true}

Step 6:

The user stacksync (User A) still has permission to edit the document.

Step 7:

Every 10 seconds the user stacksync2 (User B) checks if the document is still being blocked by the user stacksync (User A). As it continues to be blocked the user makes a request for the content of the document.

Step 8:

The content is updated in the eyeDocs document of the user stacksync2 (User B).

Step 9:

The user stacksync (User A) closes the document and frees the editing permissions.

A PUT call is made to the Sync API to remove the document block.

URL: `http://api.stacksync.com:8080/v1/unLockFile`

PUT: `{"id": "1980", "cloud": "Stacksync", "user": "stacksync", "ipserver": "192.68.56.101", "datetime": "2015-05-12 10:59:00"}`

It returns metadata that states that the file has been freed:

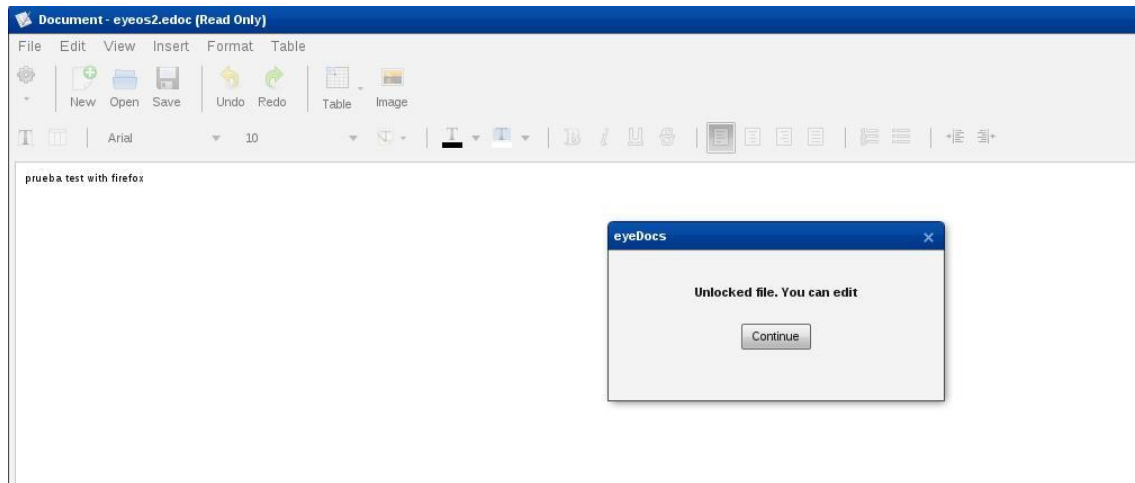
`{"unLockFile": true}`

Step 10:

User B checks if the document is still being blocked by User A. Since it has been freed, User B unblocks it and makes a request to recover the content of the document.

Step 11:

The same procedure is carried out as in Step 8 and the following message appears:



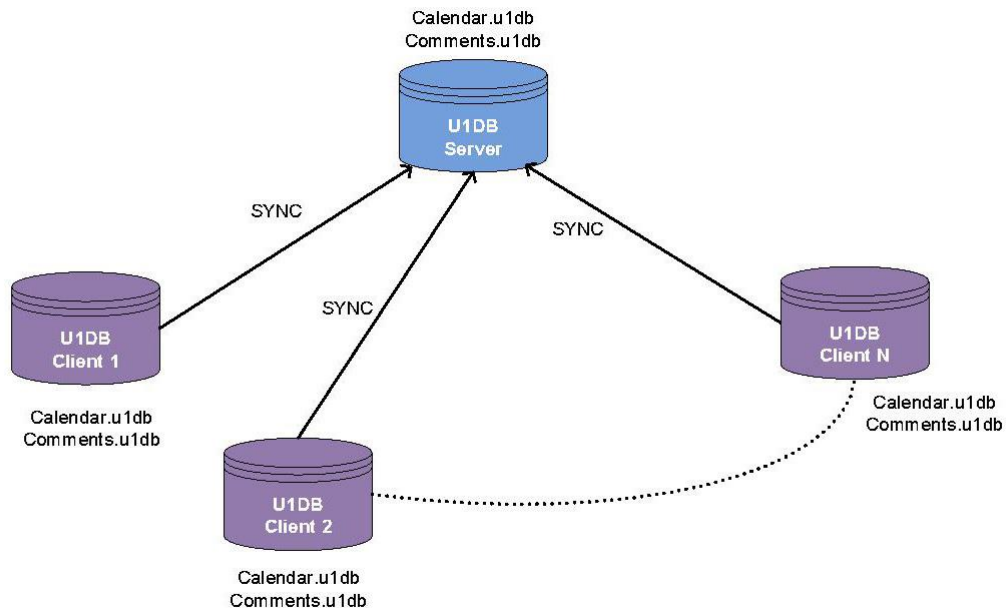
User B closes the document and frees the editing permissions.

The functions used to block and unblock files are detailed in annexes 4 and 5.

3.5 Replacement U1DB to API

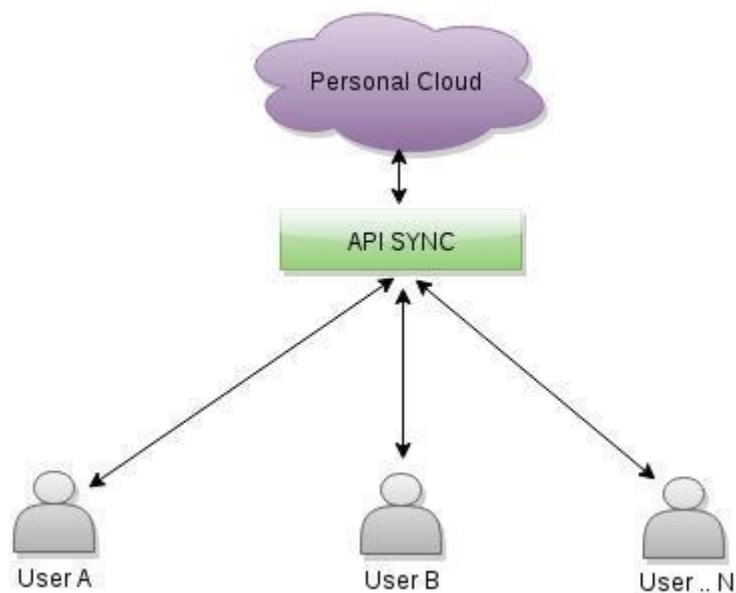
3.5.1 Implementation

Previously, the comments and calendars used U1DB, which is an API to synchronize JSON document databases created by Canonical. It allows applications to store documents and synchronize them between machines and devices. U1DB is a database designed to work everywhere, offering storage backup for the data that is native to the platform. This means that it can be used on different platforms, with different languages, with support and synchronization between all of them.



Currently, the use of the U1DB database has been removed, implementing a new Sync API which stores the comments and calendars of the user on their Personal Cloud.

The following image details how the new Sync API is implemented:

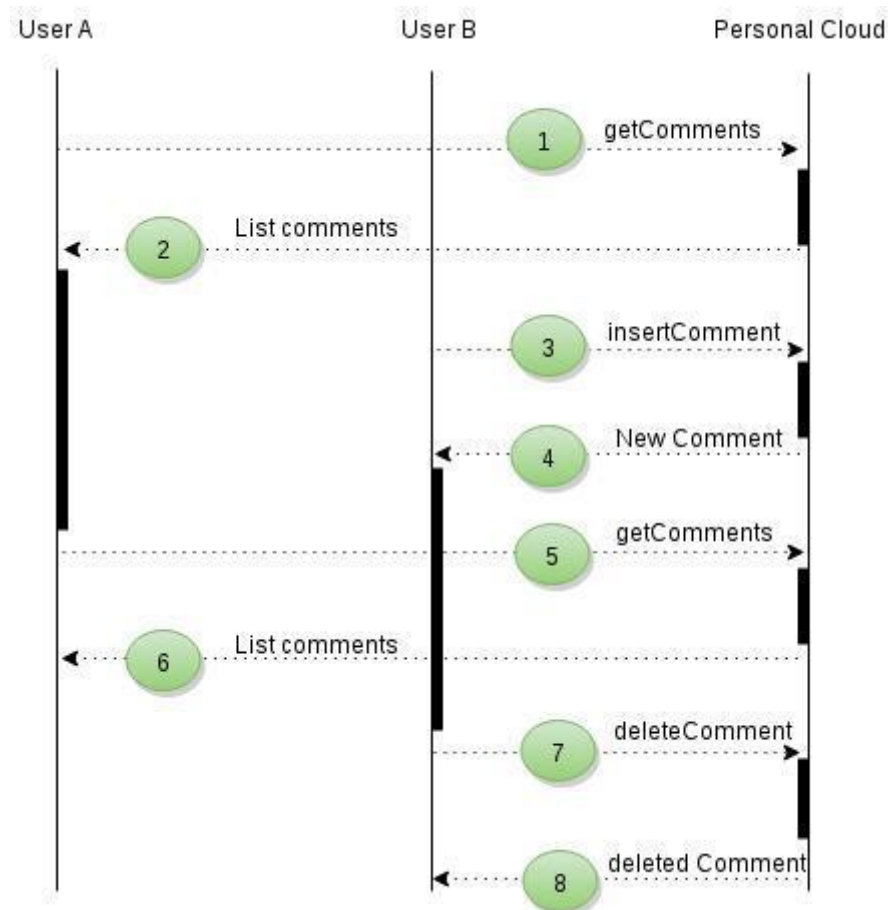


3.5.2 Comments

The eyeOS platform implements a tool that allows users to manage comments on files shared in the Personal Cloud. Comments can be created or deleted, and comments made by other users can also be seen.

In the settings file (See annex 1), there is the key “comments”, which indicates whether comments can be visualized or inserted in the files in the cloud. If it is deactivated (value set at false), the “Comments” tab will be shown but no comments will be listed, and new ones will not be able to be made.

The following diagram establishes a framework for detailing the process involved in checking, creating, and deleting comments based on the collaboration between two users:



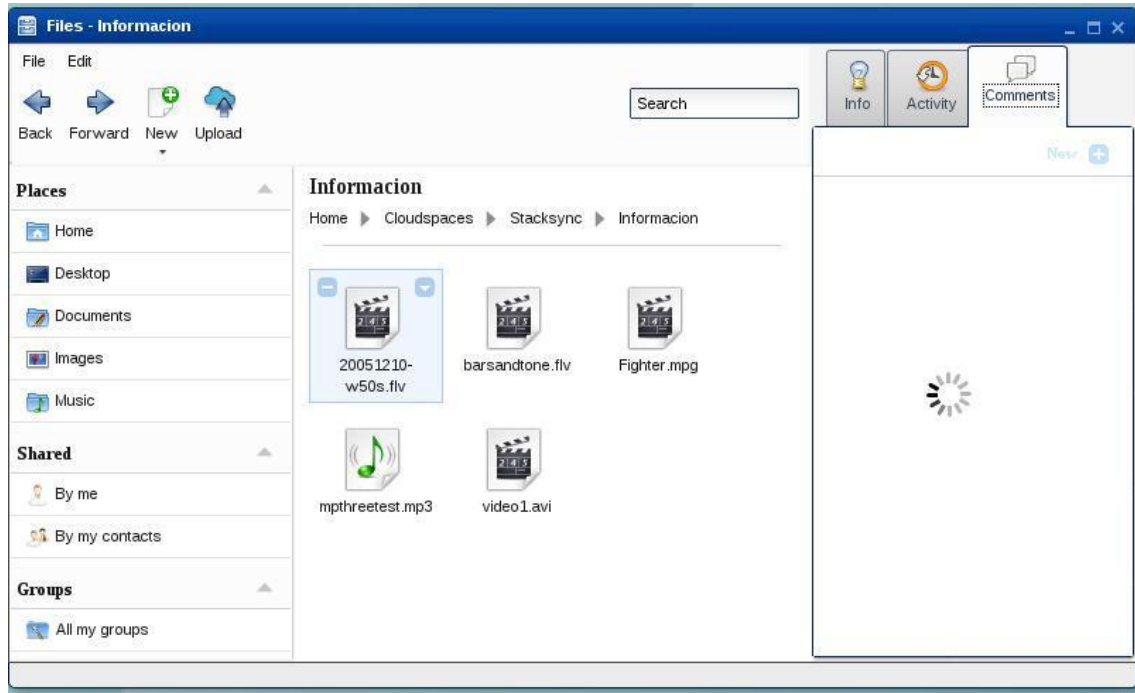
Step 1:

When selecting a file shared with other users from the same or different cloud, when clicking on the “Comments” tab of the file 2005 12 10-w50s.flv (id:1950)

located in the folder Information, the user stacksync will make a GET request using the valid StackSync token to obtain the comments related to the file:

URL parameters: { "id": "1950", "cloud": "Stacksync" }

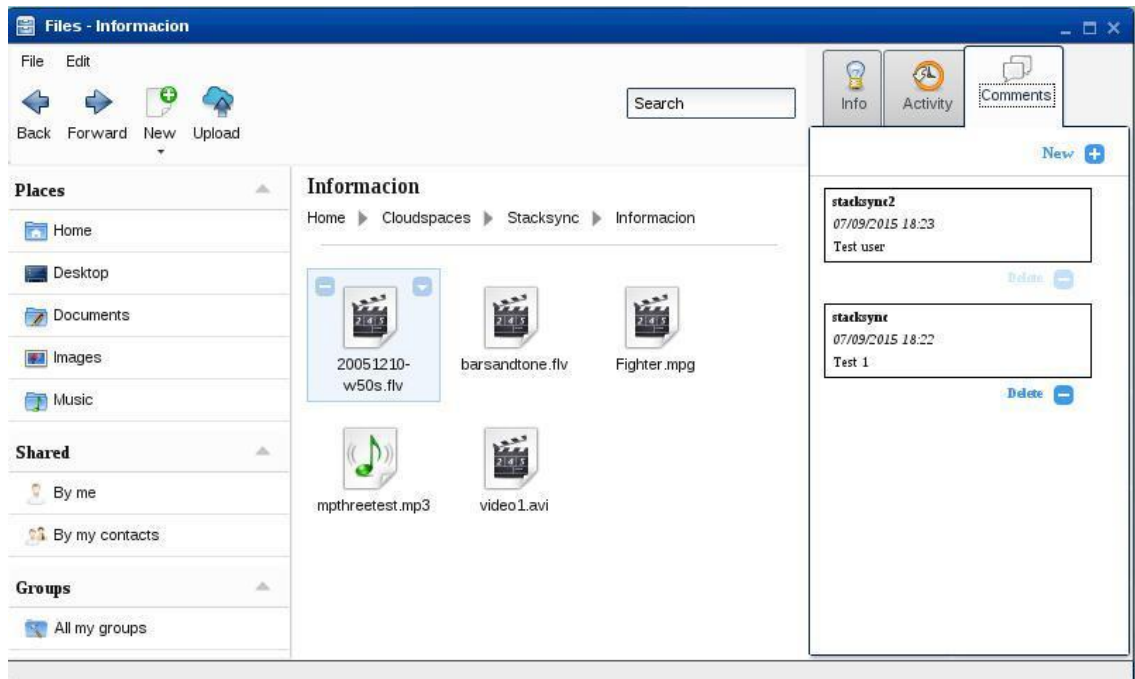
URL: <http://api.stacksync.com:8080/v1/comment/:id/:cloud>



Step 2:

It returns metadata with the list of comments associated to the file:

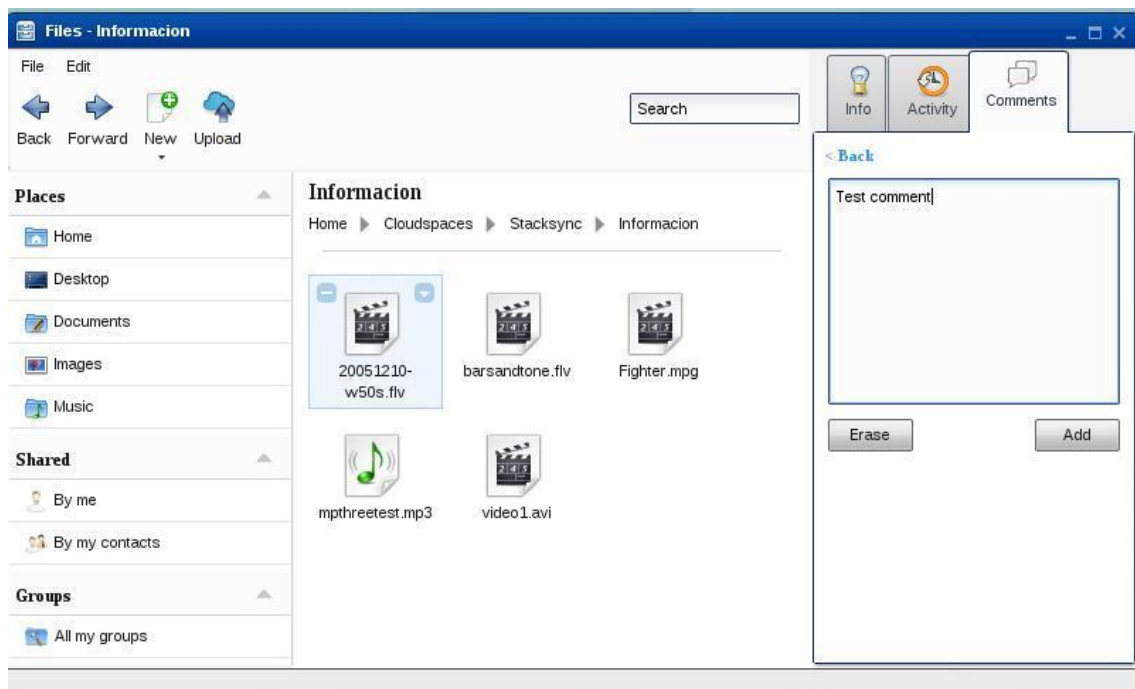
```
[ { "id": "1950", "user": "stacksync", "cloud": "Stacksync", "text": "Test 1",  
  "time_created": "201509071822", "status": "NEW" }, { "id": "1950", "user":  
  "stacksync", "cloud": "Stacksync", "text": "Test user", "time_created":  
  "201509071823", "status": "NEW" } ]
```



The comments are shown in a list which shows the most current comments first.

Step 3:

Clicking on the *New* button will bring up a form which will allow the user stacksync2 to insert a comment associated to the shared file.



A POST call is made to the Sync API to add the comment of the user stacksync2 to the file on StackSync:

URL: <http://api.stacksync.com:8080/v1/comment>

POST: { "id": "1950", "user": "stacksync2", "cloud": "Stacksync", "text": "Test comment" }

Step 4:

It returns metadata with the data of the new comment.

```
{ "id": "1950", "user": "stacksync2", "cloud": "Stacksync", "text": "Test comment", "time_created": "201509071827", "status": "NEW" }
```

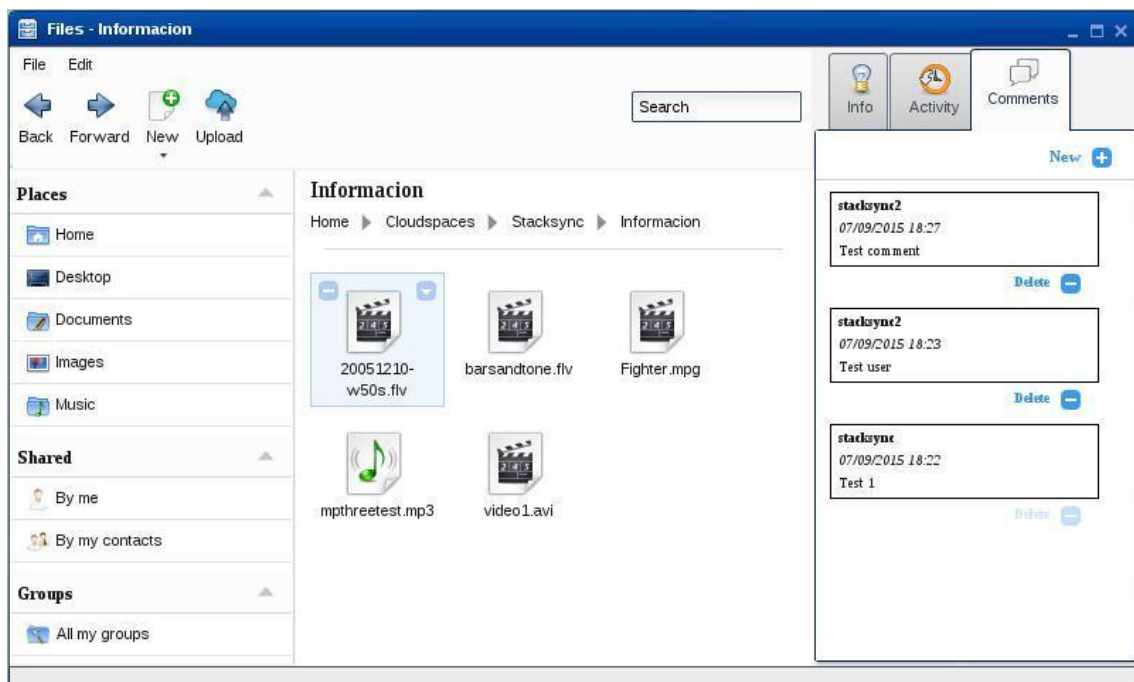
Step 5:

The user stacksync makes a query as in Step 1.

Step 6:

It returns metadata with the comment introduced by the user stacksync2:

```
[{ "id": "1950", "user": "stacksync2", "cloud": "Stacksync", "text": "Test comment", "time_created": "201509171710", "status": "NEW" }, { "id": "1950", "user": "stacksync", "cloud": "Stacksync", "text": "Test 1", "time_created": "201509071822", "status": "NEW" }, { "id": "1950", "user": "stacksync", "cloud": "Stacksync", "text": "Test user", "time_created": "201509071823", "status": "NEW" } ]
```



It is possible to witness one of the more important restrictions of this tool, which is to not allow for comments made previously by another user to be deleted. In the previous image the current user (stacksync2), who lists the comments, is not able to delete comments made by other users, such as the comment made by stacksync.

If at any time the file changes its state (it stops being shared with other users), the list of comments will still be shown to the owner, but the actions of inserting and deleting them will be disabled.

The change of state is reversible, so if the file is shared again, options for inserting and deleting comments will automatically be enabled again.

Step 7:

A call is made using the DELETE method to the Sync API to delete the comment made previously by the user stacksync2:

URL parameters: { "id": "1950", "user": "stacksync2", "cloud": "Stacksync",
"time_created": "201509071827" }

URL: http://api.stacksync.com:8080/v1/comment/:id/:user/:cloud/:time_created

Step 8:

It returns metadata with the data of the deleted comment.

```
{ "id": "1950", "user": "stacksync2", "cloud": "Stacksync", "text": "Test  
comment", "time_created": "201509071827", "status": "DELETED" }
```

For more detailed information of the functions regarding comments implemented by the Sync API, refer to annexes 4 and 5.

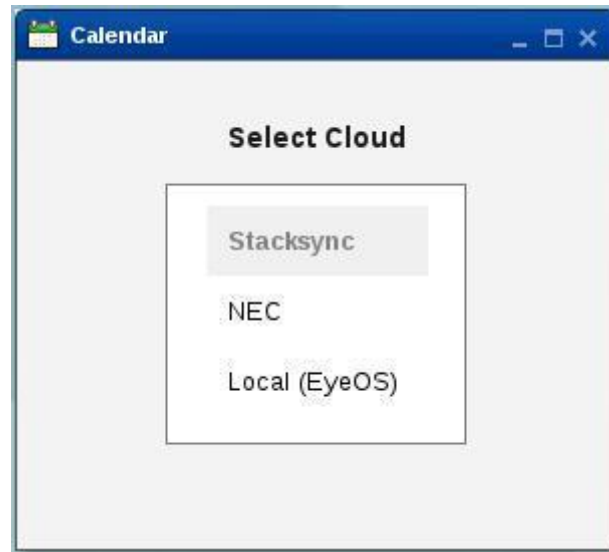
3.5.3 Calendar

Another function which has been affected by the new Sync API is the management of calendars and events. Previously the U1DB database was used in the same way as for comments. This is no longer available, to allow the user a comprehensive management of all of their calendars and events by associating them to a cloud.

With this API the user can carry out the different actions (create/update/delete), both with their calendars and with their events, regardless of the platform used.

When opening the calendar, a list of options is displayed:

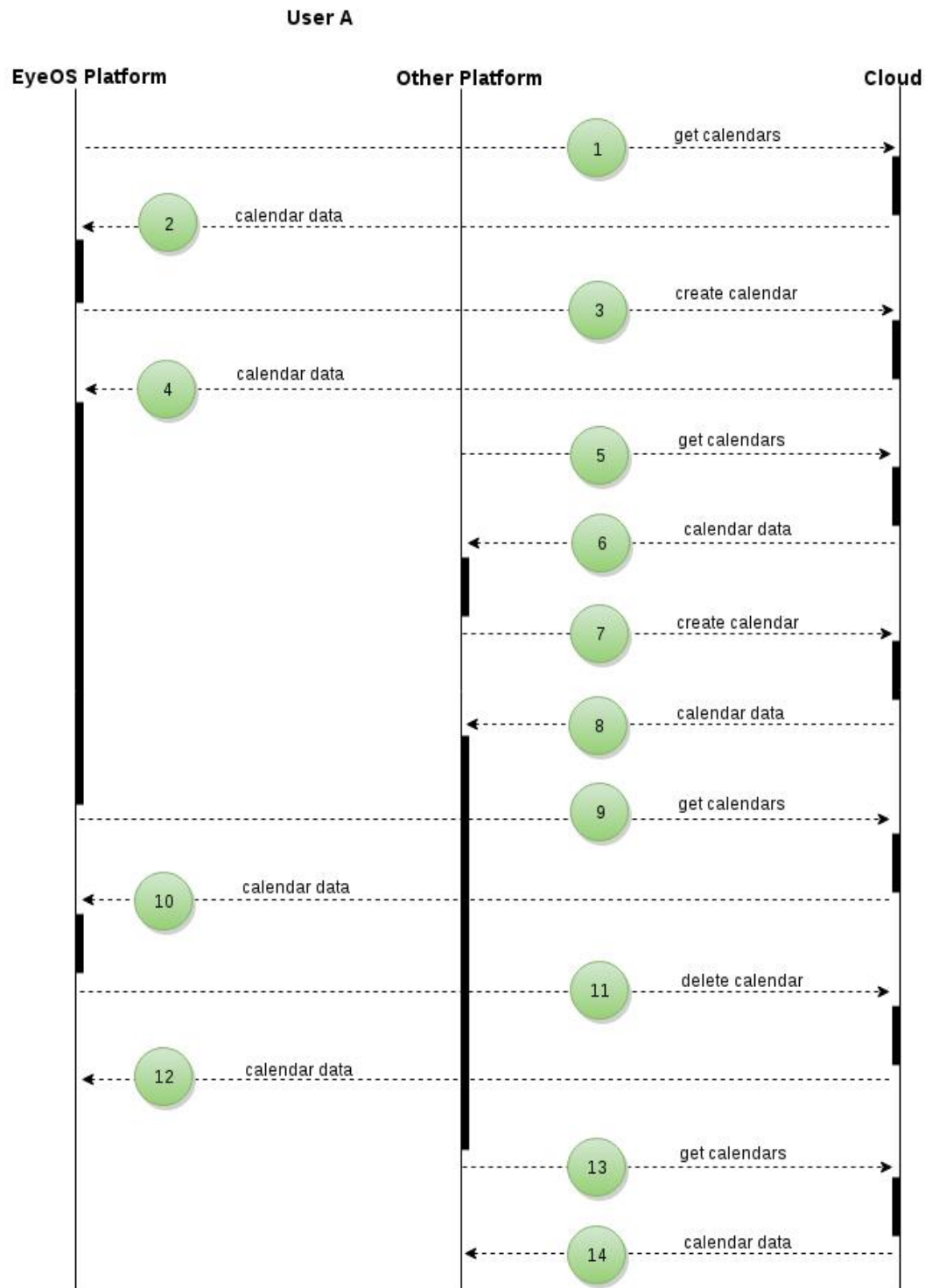
1. Names of the different clouds able to manage calendars and registered with an access_token from the current user
2. Local calendar of the eyeOS platform.



To identify a cloud with calendar and event management privileges in the eyeOS platform, the settings file needs to be accessed (see annex 1), where the “calendar” key can be set at true (active) and false (not active).

When selecting a specific cloud, the user can manage their calendars and events, and see the changes applied in real time through other platforms.

The synchronization of calendars is detailed in the following diagram:



Step 1:

The user stacksync (User A), logged into eyeOS, makes a request to the cloud to obtain a list with all their calendars.

A GET call is made to the Sync API using a valid StackSync token:

URL parameters: { "user": "stacksync", "cloud": "Stacksync" }

URL: http://api.stacksync.com:8080/v1/calendar/:user/:cloud

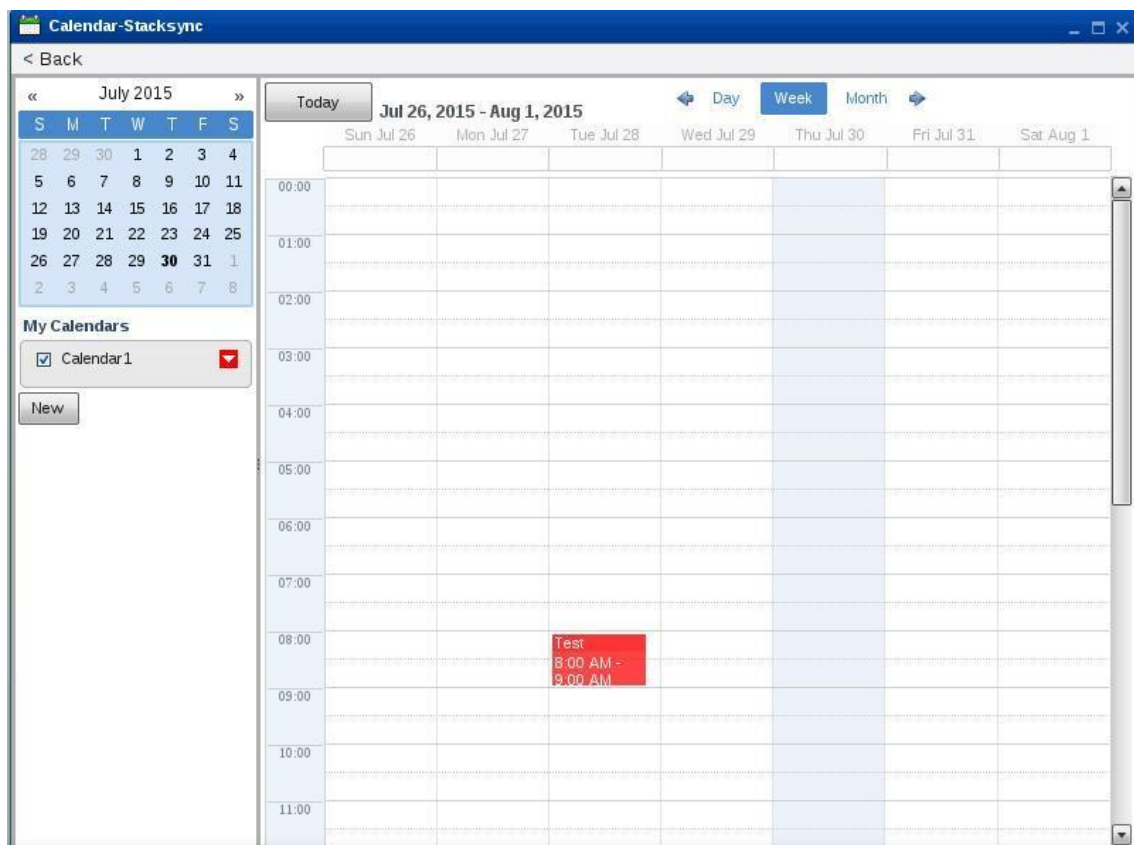
Step 2:

It returns metadata with the list of calendars:

```
[{"status": "NEW", "description": "Test Calendar", "user": "stacksync",  
"timezone": 0, "type": "calendar", "cloud": "Stacksync", "name": "Calendar 1"}]
```

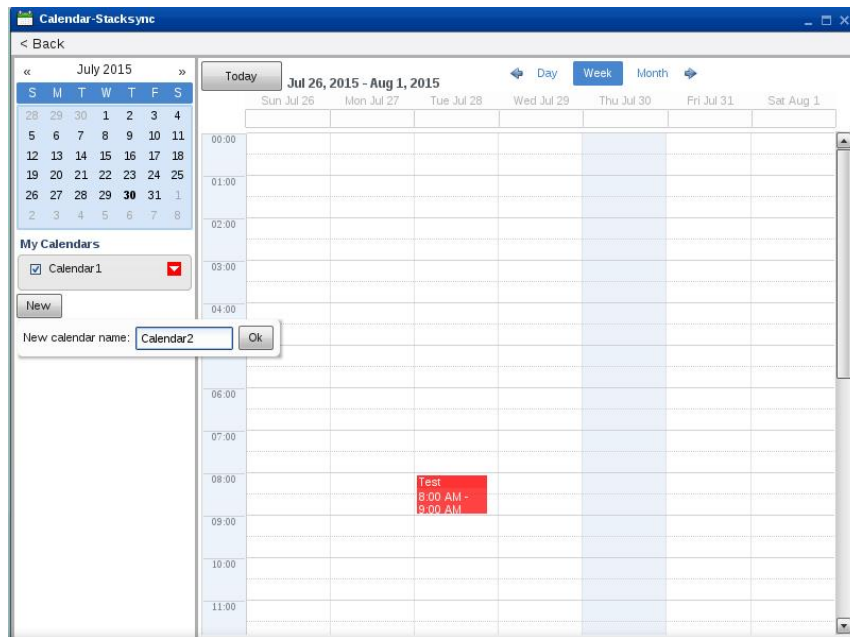
The calendar application adds these calendars to the list on the left side of the screen.

Every 20 seconds it checks whether the user has created a new calendar on another platform. If the answer is affirmative, the new calendar is added to the list.



Step 3:

The user stacksync (User A) creates a new calendar using the *New calendar* button and specifying the name that they want it to have on the cloud:



A POST call is made to the Sync API to create the calendar entered by the user stacksync:

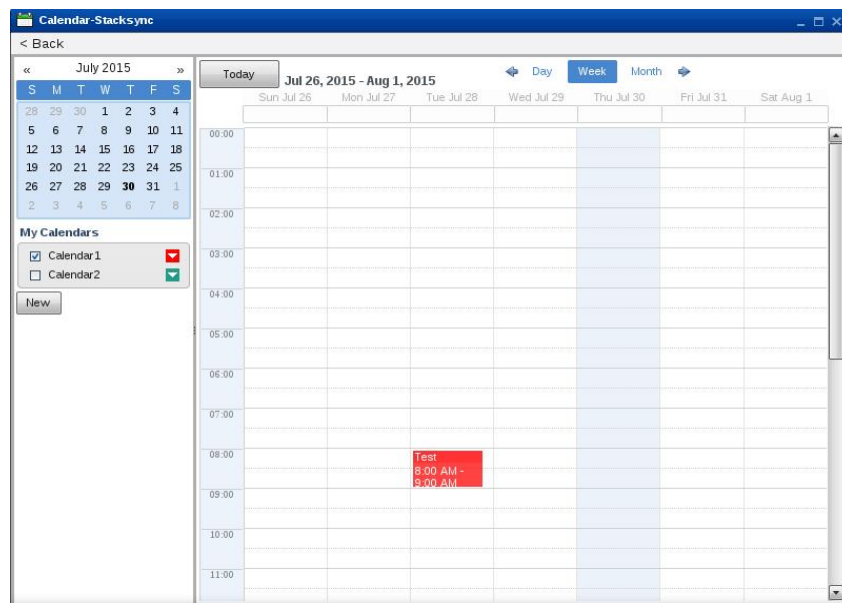
URL: <http://api.stacksync.com:8080/v1/calendar>

POST: {"user": "stacksync", "name": "Calendar 2", "cloud": "Stacksync", "description": "Test Calendar2", "timezone": 0}

Step 4:

It returns metadata with the data of the new calendar.

{"status": "NEW", "description": "Test calendar2", "user": "stacksync", "timezone": 0, "type": "calendar", "cloud": "Stacksync", "name": "Calendar 2"}



The list of calendars is updated with the data of the new calendar:

Step 5:

The user stacksync (User A), logged in on another platform, makes a request to obtain the list of calendars. The calendar introduced previously on eyeOS is included on this list.

Step 6:

The platform will update the calendars of the user, with the data obtained in the response to the request

Step 7:

The user stacksync (User A) creates a new calendar on this platform, identifying with a name on the cloud.

Step 8:

The platform refreshes the list of calendars including the new calendar.

Step 9:

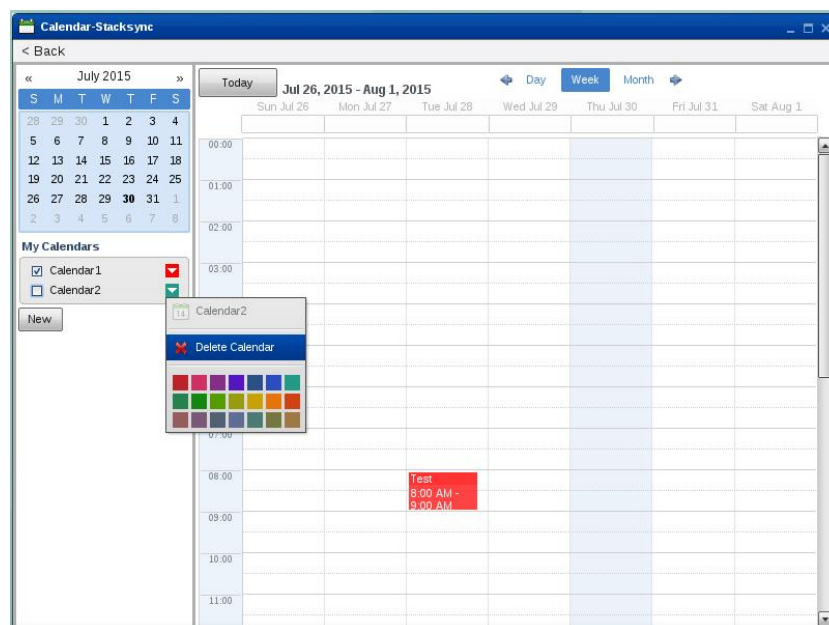
The calendar application checks every 20 seconds to see if a change has been made in the user's calendar. When a request is made to obtain the list of calendars, it checks if a new calendar has been created from another platform.

Step 10:

The list of calendars is updated, adding the calendar created from another platform.

Step 11:

The user stacksync (User A) deletes a calendar from the context menu:



When using this option a DELETE call is made to the Sync API to delete the calendar:

URL parameters: {"user": "stacksync", "name": "Calendar 2", "cloud": "Stacksync"}

URL: <http://api.stacksync.com:8080/v1/calendar/:user/:name/:cloud>

It returns metadata with the data of the deleted calendar.

```
{"type":"calendar","user": "stacksync", "name": "Calendar 2", "cloud":  
"Stacksync", "description": "Test calendar 2, "timezone": 0,"status":  
"DELETED"}
```

Step 12:

The list of calendars is updated, deleting the selected calendar.

Step 13:

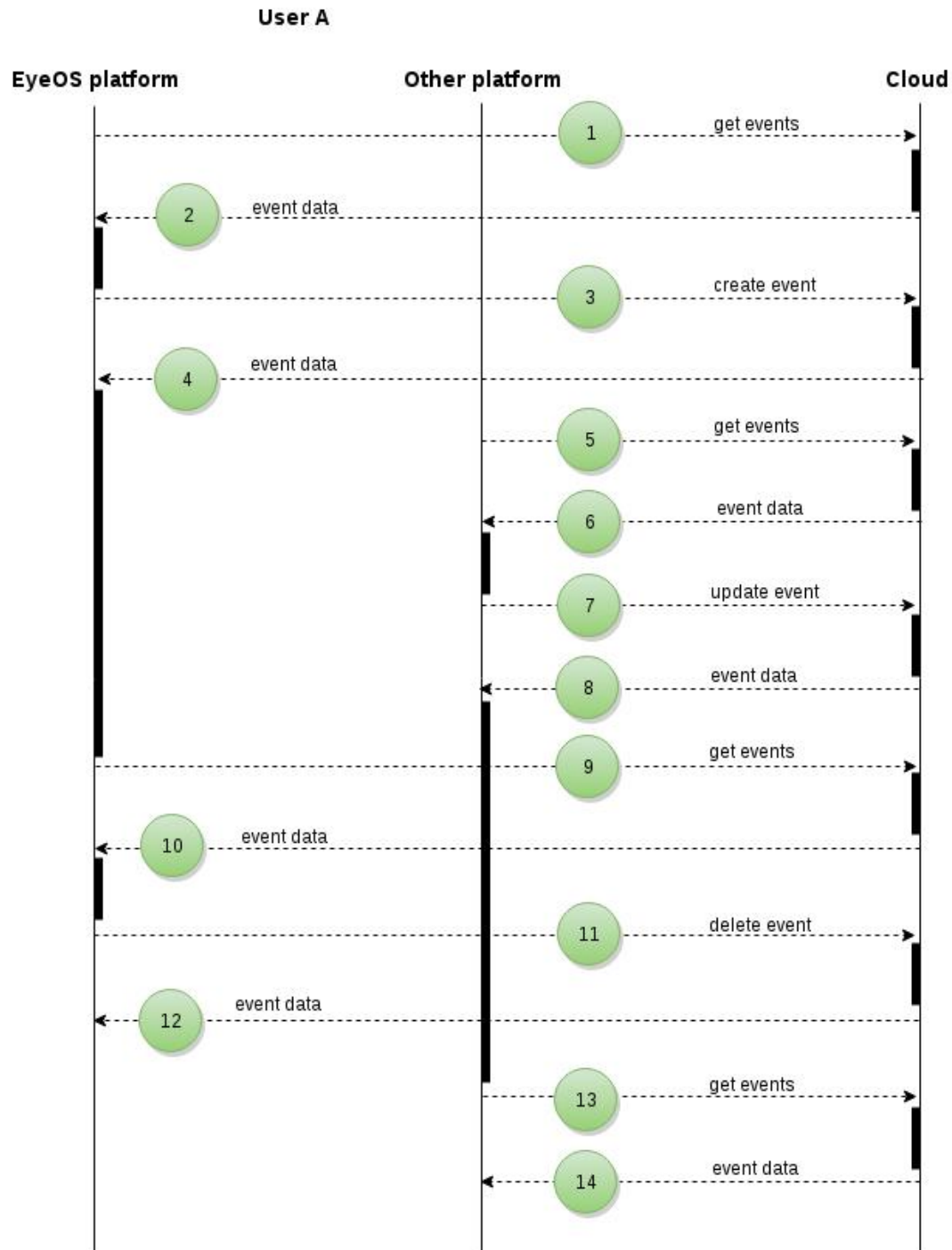
The user stacksync (User A) makes a request on the other platform to obtain a list of their calendars. They check whether any change has been made to the calendars from another platform.

Step 14:

The platform deletes the calendar deleted with eyeOS from the list.

The user can synchronize their events with other platforms, selected the calendars from the list. When a calendar is selected, events can be created/updated/deleted in the cloud.

The synchronization of events is detailed in the following diagram:



Step 1:

The user stacksync (User A), logged into eyeOS, selects the calendars they wish to view and makes a request to obtain all of the events associated to the calendar which are stored in the Personal Cloud.

A GET call is made to the Sync API using a valid StackSync token:

URL parameter: {"user": "stacksync", "calendar": "Calendar 1", "cloud": "Stacksync"}

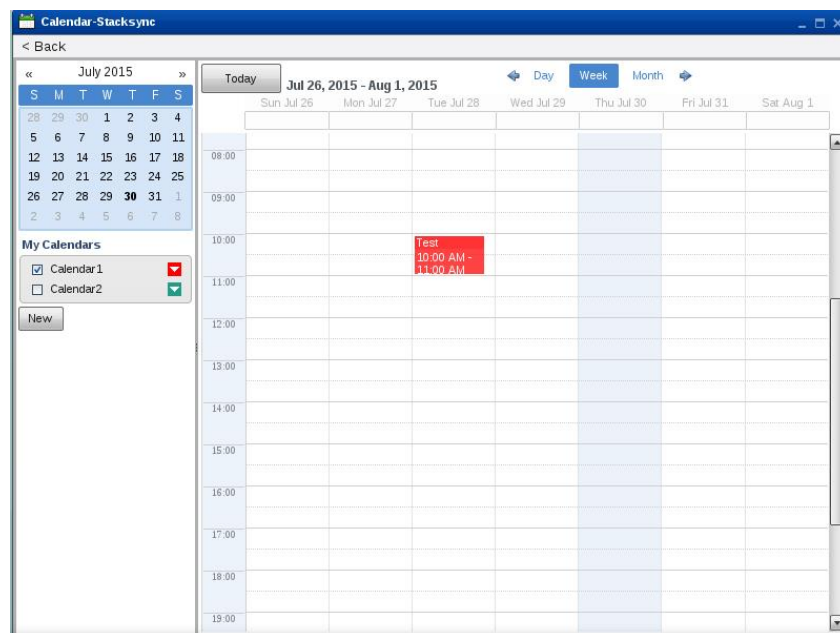
URL: http://api.stacksync.com:8080/v1/event/:user/:calendar/:cloud

Step 2:

It returns metadata with the events associated to Calendar 1:

```
[{"status": "NEW", "description": "Test", "location": "Barcelona", "finalvalue": "0", "timeend": "20150828110000", "finaltype": "1", "timestart": "20150828100000", "isallday": 0, "user": "eyeos", "repeattype": "n", "calendar": "personal", "repetition": "None", "type": "event", "cloud": "Stacksync", "subject": "Test"}]
```

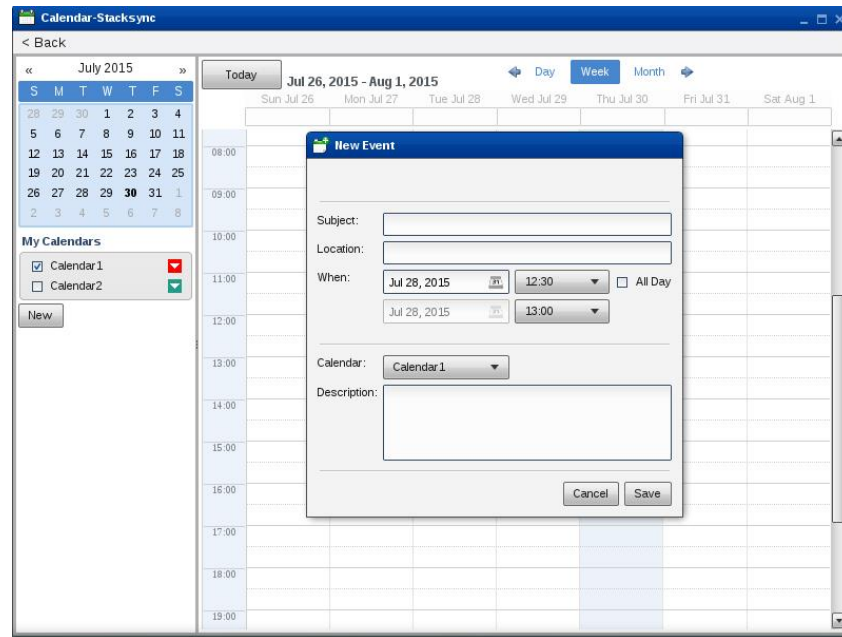
The calendar application shows the events according to the type of calendar selected by the user. Events can be viewed by day, week, or month.



In this case, the user has selected the option to view events by week.

Step 3:

When clicking on a cell in the calendar, a dialog box is shown which allows users to insert an event and associate it with a particular calendar:



A POST call is made to the Sync API to introduce an event in Calendar 1:

URL: <http://api.stacksync.com:8080/v1/event>

POST: {"user": "stacksync", "calendar": "Calendar 1", "cloud": "Stacksync", "isallday": 0, "timestart": "20150828120000", "timeend": "20150828130000", "repetition": "None", "finaltype": "1", "finalvalue": "0", "subject": "Test", "location": "Barcelona", "description": "Test", "repeattype": "n" }

Step 4:

It returns metadata with the data of the event introduced in the calendar.

```
{"type": "event", "user": "stacksync", "calendar": "Calendar 1", "cloud": "Stacksync", "isallday": 0, "timestart": "20150828120000", "timeend": "20150828130000", "repetition": "None", "finaltype": "1", "finalvalue": "0", "subject": "Test", "location": "Barcelona", "description": "Test", "repeattype": "n", "status": "NEW" }
```

Calendar introduces the details of the new events in the calendar cells.

It checks every 10 seconds if any change has been made in the events from another platform. If they have, the calendar cells are refreshed with the updated events.

Step 5:

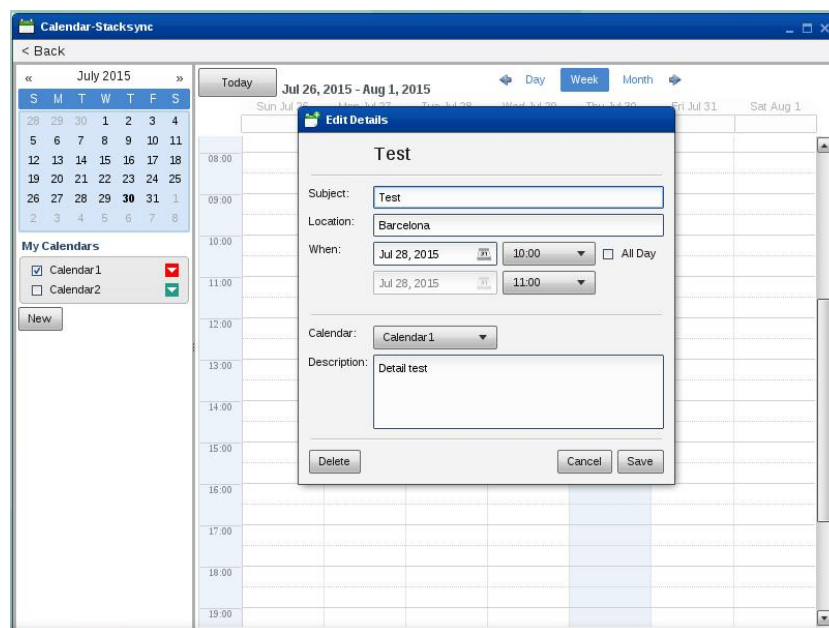
From a different platform, the user stacksync (User A) makes a query to obtain a list of events associated to the selected calendar. This list includes the event created previously from the eyeOS platform.

Step 6:

The platform updates its list of events and displays them on the screen.

Step 7:

The user stacksync (User A) can edit all the information related to the event. If the platform used is eyeOS, they need to click on a previously entered event. The following dialog box is displayed:



A PUT call is made to the Sync API to update the description of the event.

URL: <http://api.stacksync.com:8080/v1/event>

POST: {"user": "stacksync", "calendar": "Calendar 1", "cloud": "Stacksync", "isallday": 0, "timestart": "20150828120000", "timeend": "20150828130000", "repetition": "None", "finaltype": "1", "finalvalue": "0", "subject": "Test", "location": "Barcelona", "description": "Detail Test", "repeattype": "n" }

Step 8:

It returns metadata with the data of the updated calendar.

```
{ "type": "event", "user": "stacksync", "calendar": "Calendar 1", "cloud":  
"Stacksync", "isallday": 0, "timestart": "20150828120000", "timeend":  
"20150828130000", "repetition": "None", "finaltype": "1", "finalvalue": "0",  
"subject": "Test", "location": "Barcelona", "description": "Detail Test",  
"repeattype": "n", "status": "NEW" }
```

The platform updates the event with the data received from the cloud.

Step 9:

The calendar application checks every 10 seconds whether changes have been made to the events associated to the calendar. In this case, it finds that there are updates to an event made from a different platform.

Step 10:

The event is updated locally with the data received from the cloud.

Step 11:

The user stacksync (User A) can delete an event by clicking on the event and selecting delete from the dialog box to edit the event.

A DELETE call is made to the Sync API to delete the selected event:

URL parameter: { "user": "stacksync", "calendar": "Calendar 1", "timestart":
"20150828120000", "timeend": "20150828130000", "isallday": 0 }

URL: http://api.stacksync.com:8080/v1/event/:user/:calendar/:cloud/:timestart/
:timeend/:isallday

Step 12:

The event is deleted from the calendar cell.

Step 13:

The platform makes a query to check if any of the events have been changed. In this case it is seen that an event has been deleted.

Step 14:

The platform deletes the event previously deleted from eyeOS from its calendar.

For more detailed information on the reference functions of the calendars and events that the Sync API implements, refer to annexes 4 and 5.

The synchronization of comments and calendars are associated to a specific cloud, with which the configuration of the Sync API is linked to the configuration of that cloud (annex 1). In the event that synchronization in the cloud is not implemented, an external Rest API can be used, following the same defined contract in our Sync API, which can be configured in settings.php, identifying the URL where our API should connect. The constant is API_SYNC.

4. Tissat

4.1 Validation and Feedback analysis from open Internet trials

4.1.1 General description

We developed a monitoring system to infer in real-time the resource consumption of StackSync users in a cloud infrastructure.

StackSync runs the management of the files, so the current production version allows you to monitor a number of parameters, giving us a detail by user, due to the type of structure used.

The version currently in pre-production, as it includes more advanced functionalities like sharing, gives us different monitoring values, due to the characteristics of the tool.

Despite the traces that we gathered so far capture only a small number of users, they are a proof-of-concept of our monitoring system.

Because the production platform does not include the sharing functionality, the platform is not yet of great use and interest to most users. In the coming months, with the inclusion of the sharing system, it will give us much more realistic traces of use of the platform.

4.1.2 Setting up a metering service for a storage system

It has been implemented system usage several traces for the analysis of the platform, obtaining information using not only the OpenStack platform, but also the StackSync APIs.

Among the objectives of the analysis, it is studied how the implementation of encryption affects to the average load of the platform, or how it affects the folder sharing to the average usage per user of the platform.

These early analysis of the platform help us to start dimensioning cloud platforms, based on preliminary estimates lab, about the use of resources.

We had two separate parts in our setup: the proxy-server and the ceilometer-server.

In the proxy server we had to install the following packages:

```
root@swift-p1-dev:~# dpkg -l | grep ceilometer
ii ceilometer-api          2013.2.3-0ubuntu1~cloud0      ceilometer api
service
ii ceilometer-common      2013.2.3-0ubuntu1~cloud0      ceilometer
common files
ii python-ceilometer       2013.2.3-0ubuntu1~cloud0      ceilometer
python libraries
ii python-ceilometerclient 1.0.5-0ubuntu1~cloud0         Client library for
Openstack ceilometer server.
```

Then we had to add a filter to our proxy-server in order for the ceilometer agents to grab service usage data.

```
[filter:ceilometer]
use = egg:ceilometer#swift
```

And configuring the /etc/ceilometer/ceilometer.conf with the same parameters as the server.

And in the ceilometer-server we installed, these are the minimal tools to gather info from swift:

```
root@stacksync-log1-dev:~# dpkg -l | grep ceilometer
ii ceilometer-agent-central 2013.2.3-0ubuntu1             all
ceilometer central agent
ii ceilometer-api           2013.2.3-0ubuntu1             all      ceilometer
api service
ii ceilometer-collector     2013.2.3-0ubuntu1             all      ceilometer
collector service
ii ceilometer-common        2013.2.3-0ubuntu1             all
ceilometer common files
ii python-ceilometer        2013.2.3-0ubuntu1             all
ceilometer python libraries
ii python-ceilometerclient  1.0.5-0ubuntu1                all      Client
library for Openstack ceilometer server.
```

- ceilometer-agent-central: polls for resource utilization, in this case queries the swift proxy-server
- ceilometer-collector: monitor the message queues, publishes messages in a queue. Notification messages are processed and turned into metering

messages and sent back out onto the message bus using the appropriate topic. Telemetry messages are written to the data store without modification.

- ceilometer-api: we use it as an entry point to retrieve our data

The MongoDB was chosen because of the nature of a lot of concurrent writes when gathering usage statics.

Service	Meter	Description
Swift_meters	storage.api.request	Number of API requests against swift
Swift_meters	storage.objects.containers	Number of containers
Swift_meters	storage.objects.incoming.bytes	Number of incoming bytes
Swift_meters	storage.objects	Number of objects
Swift_meters	storage.objects.outgoing.bytes	Number of outgoing bytes
Swift_meters	storage.objects.size	Total size of stored objects

MongoDB is used as a storage backend for Ceilometer, all of our data collected by ceilometer will be stored in this database. The metering service creates so much data that MongoDB was their choice for production environments.

We have Openstack standard dashboard to print out a metering report per tenant:

admin

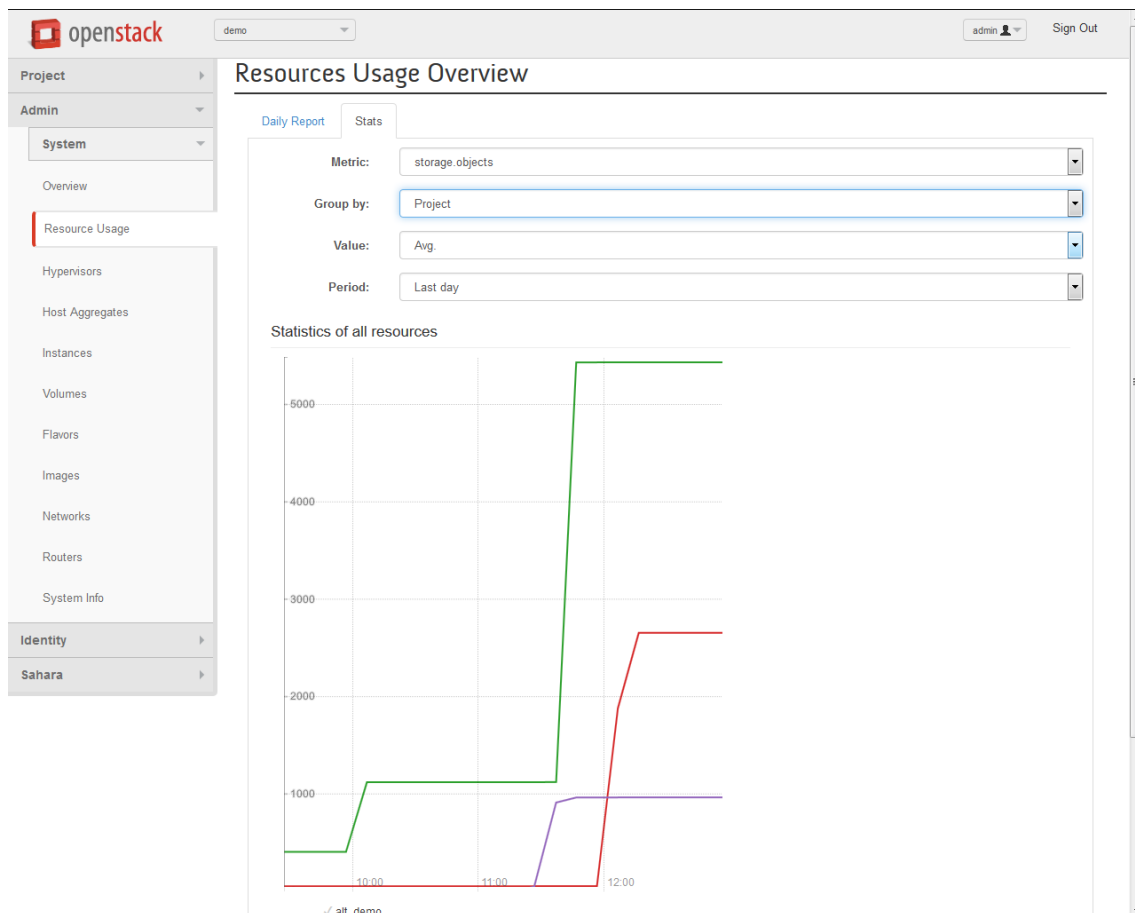
Service	Meter	Description	Day	Value (Avg)
Glance	image.size	Uploaded image size	2014-09-17	60,877,166.25
Swift_meters	storage.api.request	Number of API requests against swift	2014-09-17	1.0
Swift_meters	storage.objects.containers	Number of containers	2014-09-17	4.0
Swift_meters	storage.objects.incoming.bytes	Number of incoming bytes	2014-09-17	76,182.5808081
Swift_meters	storage.objects	Number of objects	2014-09-17	406.0
Swift_meters	storage.objects.outgoing.bytes	Number of outgoing bytes	2014-09-17	31,572.1388889
Glance	image	Image existence check	2014-09-17	1.0
Swift_meters	storage.objects.size	Total size of stored objects	2014-09-17	30,168,302.0
Displaying 8 items				

demo

Service	Meter	Description	Day	Value (Avg)
Swift_meters	storage.api.request	Number of API requests against swift	2014-09-17	1.0
Swift_meters	storage.objects.containers	Number of containers	2014-09-17	4.0
Swift_meters	storage.objects.incoming.bytes	Number of incoming bytes	2014-09-17	1,499,152.89091
Swift_meters	storage.objects	Number of objects	2014-09-17	52.0
Swift_meters	storage.objects.outgoing.bytes	Number of outgoing bytes	2014-09-17	1,026.36842105
Swift_meters	storage.objects.size	Total size of stored objects	2014-09-17	55,681,364.0
Displaying 6 items				

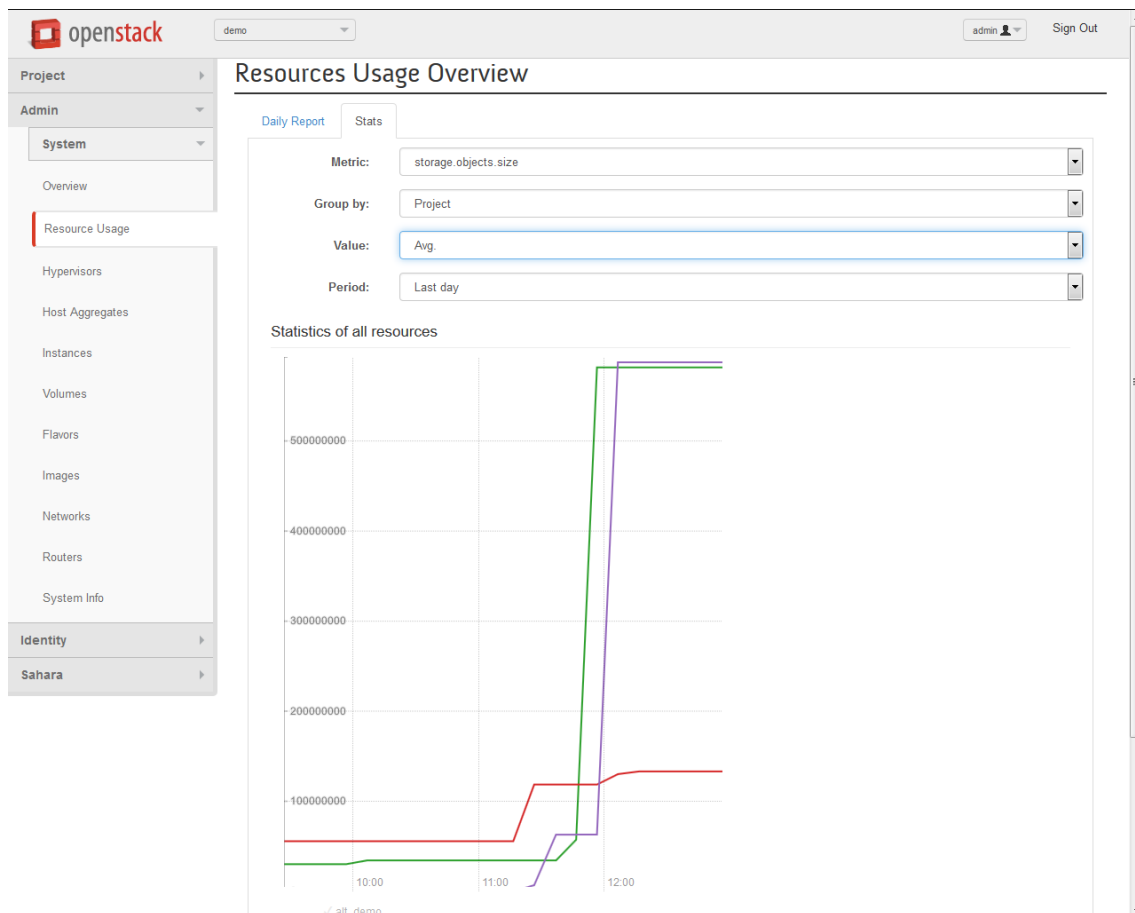
We can show the average number of files that a tenant has on any given day.

It will be used for general auditing, control of the use storage, and for billing purposes.



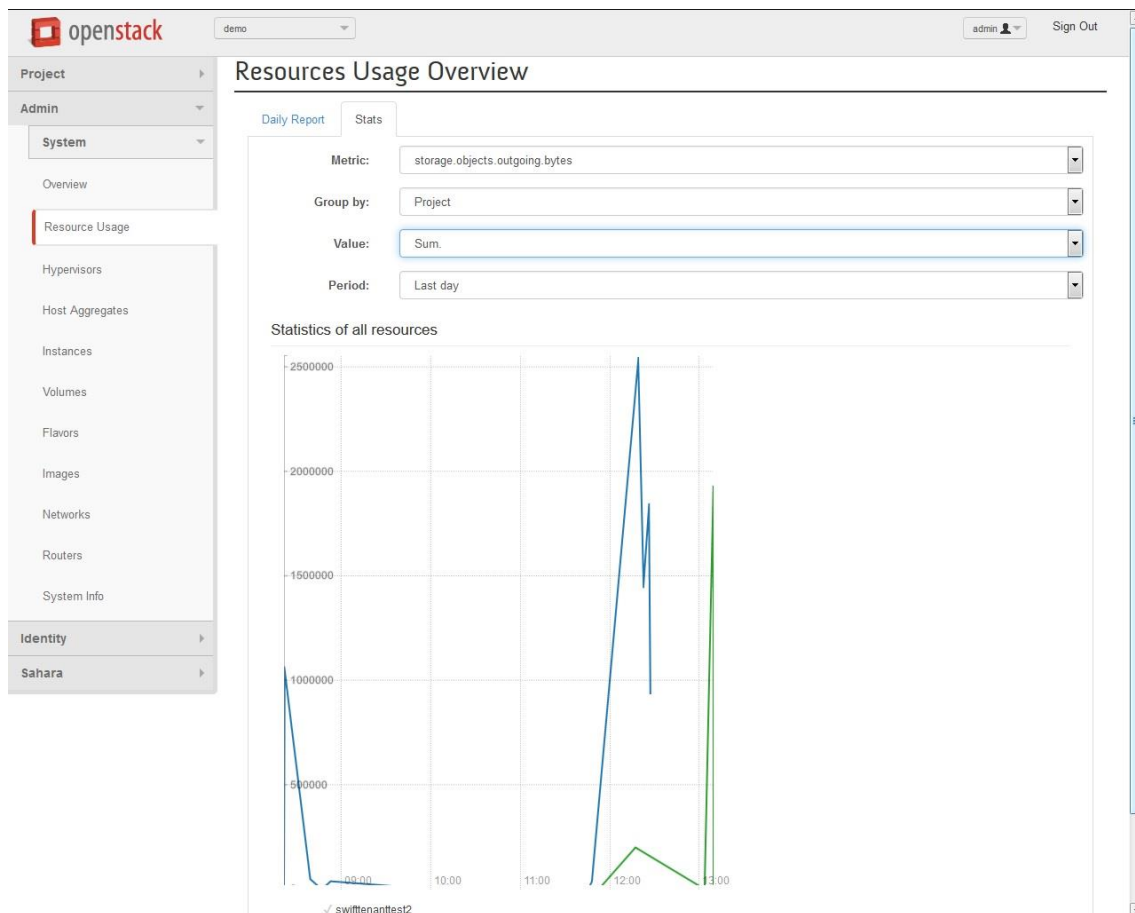
We can check the files average size that a tenant has.

Make a realistic assessment of the performance is difficult at this stage of the project, because of multiple platforms that are analyzed, considering the different released versions of software, and the implemented functionality at each one (sharing, encryption, exclusive StackSync use, Openstack platform shared with StackSync users...). However, these measures are useful at this stage, in order to validate the correct operation of the platform and also for the dimensioning of the new platforms.



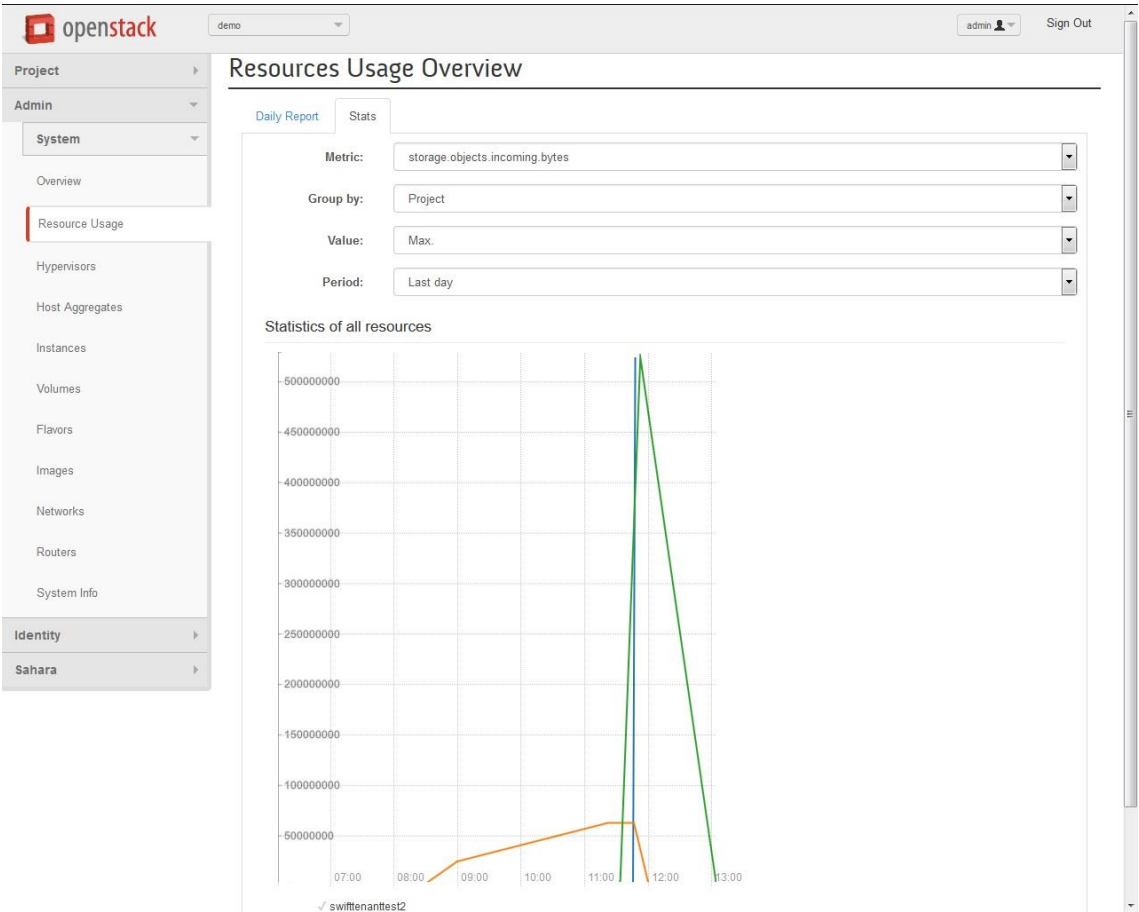
Amount of bytes that users have downloaded.

This metric is particularly interesting in order to know the bandwidth required to operate the platform, and the required load balance, according to the number of users and the size of the files.

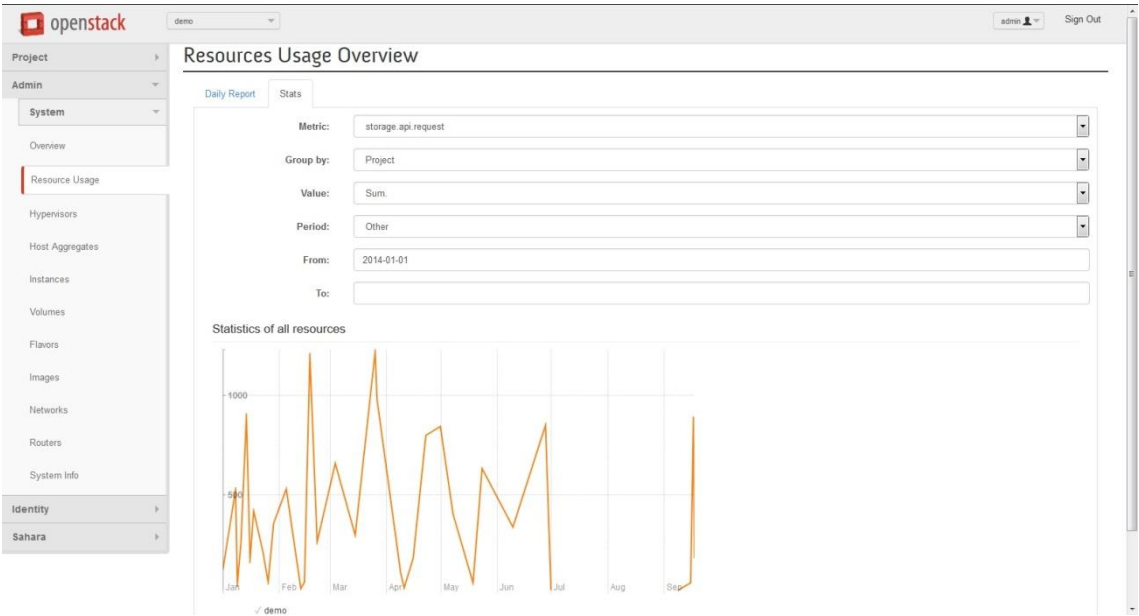


Amount of bytes that users have uploaded.

The File upload to the platform is a company goal, due that the company will rent disk space to the user, so it is important to know this type of action of the users and how affects to the used size of the platform, the encryption processes.



The graph allows us to see the use load of the platform at a specific time slot:



In this diagram, we can appreciate the number of request that our users made during a 9 month period, we are showing swift requests, which means that a for a stacksync file request can spawn into multiple swift file requests, depending on the number of chunks that a file has.

We can see non activity during holiday period, we can see spikes every month meaning that, every month our software is being used.

The identification of several users that represent the average use of the platform, will provide a detailed analysis and values for the forecasting of the user behavior, although massive data analysis would be periodically performed.

4.2 Service Platform reference prototype

These are the tasks performed by Tissat, which has resulted in improved safety and efficiency as well as the increase of the functionality, and the number of potential devices on which runs StackSync.

- Migrate Keystone v2.0 to Keystone v3
- Secure StackSync platform with SSL
- Development of group membership for StackSync users
- Development of a group-based storage quota system
- Migrate web interface back-end from PHP to Python
- Refactoring of StackSync web client
- Implemented a folder sharing functionality for StackSync users
- StackSync support for the ownCloud application
- Development of iOS App

4.2.1 Migrate Keystone v2.0 to Keystone v3

Given the fact that Tissat is a cloud provider we had to be able to grant administrative privileges to our customers so that they could manage their own resources (containers, tenants, user accounts).

Regretfully, under Keystone v2.0, the ‘admin’ role had a global scope which allowed that a user admin with admin privileges on a specific tenant could erase other tenants.

Since Tissat has active customers that need to have admin privileges to create their own users, we had a situation where a customer could delete other customer's data. Obviously this was a security issue.

Even though this issue was solved in later releases of StackSync Server through a change in stacksnc's user data mapping scheme. The change involved going from a tuple 'tenant: user' to a 'stacksync_tenant:user_container' scheme, where all users have a container under the same tenant.

Another challenge that we faced was the fact that system administrators didn't want to grant a 'stacksync admin user' with a 'ResellerAdmin' role even though it was necessary change quota or swift attributes on StackSync tenants because that could lead to a situation where an 'stacksync admin user' could actually delete data store in other users Swift system.

At that point in time, Tissat solved the problem, by adopting Keystone V3 for our cloud infrastructure, and use a domain-based approach.

This way we could create separate entities where administrative users could not interfere with other resources belonging to other users/customers, at least on the keystone level.

So we are currently using keystone v3 coupled with StackSync Server v0.4.4 on our main environment, allowing cloudspaces disk capacity could reach 100% of the company's current platform.

Keystone:

```
[ssl]
enable = True
certfile = /etc/keystone/ssl/certs/signing_cert.pem
keyfile = /etc/keystone/ssl/private/signing_key.pem
ca_certs = /etc/keystone/ssl/certs/ca.pem
ca_key = /etc/keystone/ssl/certs/cakey.pem
key_size = 1024
valid_days = 3650
ca_password = None
cert_required = False
cert_subject = /C=US/ST=Unset/L=Unset/O=Unset/CN= XXXXXXXXXXXX

[signing]
token_format = PKI
certfile = /etc/keystone/ssl/certs/signing_cert.pem
keyfile = /etc/keystone/ssl/private/signing_key.pem
ca_certs = /etc/keystone/ssl/certs/ca.pem
ca_key = /etc/keystone/ssl/private/cakey.pem
key_size = 2048
valid_days = 3650
cert_subject = /C=US/ST=Unset/L=Unset/O=Unset/CN=XXXXXXXXXXXX
```

4.2.2 Secure our StackSync platform with SSL

Since we were developing a web interface, we had in mind from the get-go that our web server would force all its connections over SSL, being implemented secure communications between client and server.

But what happens with the rest of the clients?

Given the fact StackSync clients transfer potentially private data, we must ensure that data is transmitted safely over an insecure network such as the Internet.

We can identify three phases on the StackSync workflow where data could be sent in plaintext and therefore exposed to attack. These steps are the keystone authentication process, the swift file transfer and the RabbitMQ metadata transfer.

Our solution was to set up SSL on all services over the Internet, currently we have operating Keystone and Swift.

We have currently setup RabbitMQ working with and without SSL, but have now recognized the parts of the StackSync codebase that need to be changed.

It is necessary to create a rabbitmq.config in /etc/rabbitmq and restart the queue service.

```
[
  {rabbit, [
    {ssl_listeners, [5671]},
    {ssl_options, [{cacertfile, "/path/to/testca/cacert.pem"},
                  {certfile, "/path/to/server/cert.pem"},
                  {keyfile, "/path/to/server/key.pem"},
                  {verify, verify_peer},
                  {fail_if_no_peer_cert, false}]}]}
].
```

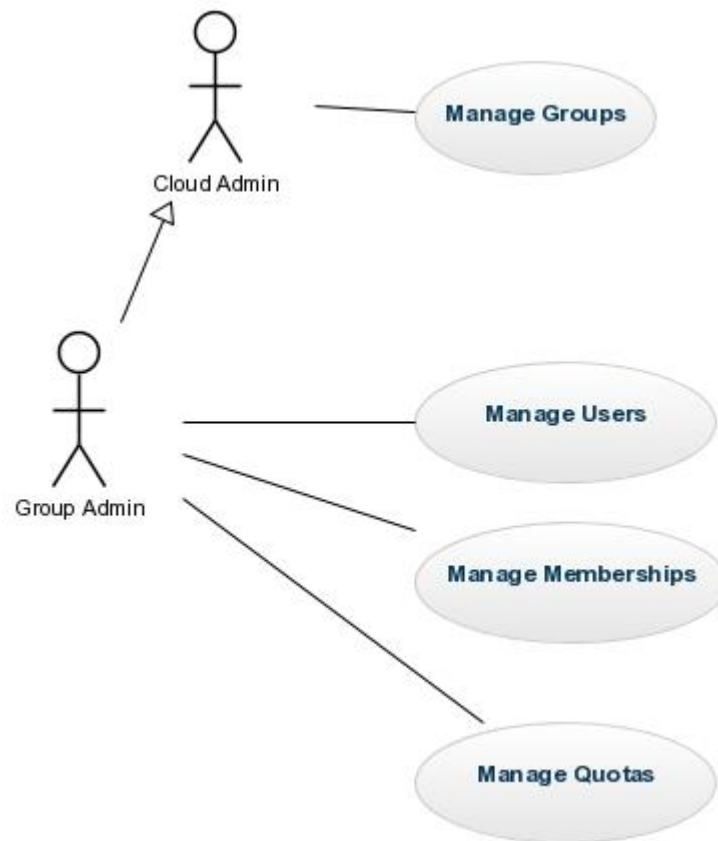
First, we studied the parts of the broker OMQ, since it's a wrapper form RabbitMQ we were confident we could setup SSL easily. Now we noticed that the StackSync-server didn't configure itself to be using a SSL connection.

4.2.3 Development of group-based membership for StackSync users

The ability to group users into groups or departments makes easy the quota management allowing the existence of an administrators group, which can manage users and their available space, using StackSync.

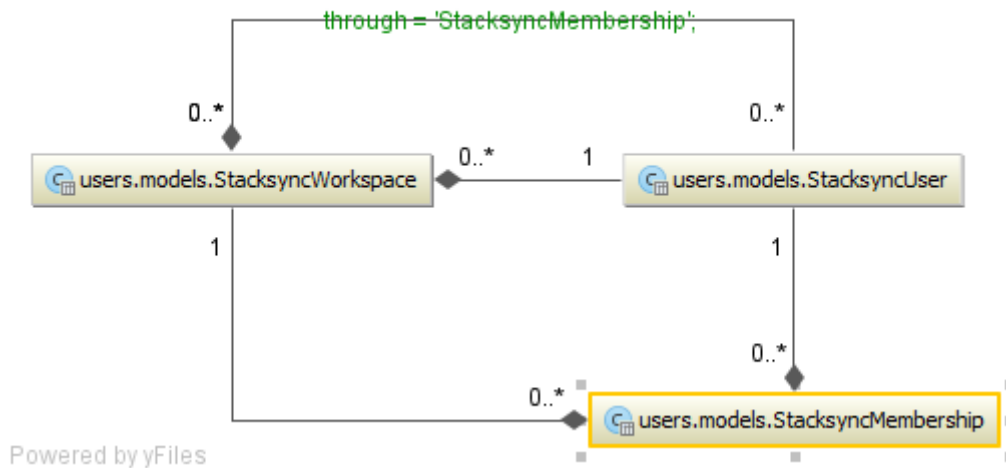
A group like URV would manage a certain amount of storage space, and this organization could have some administrative user that can create and delete users, and assign quota to its users.

The following use case presents a situation where a 'Cloud admin' user creates groups and grants 'Group admin' privileges to a user.



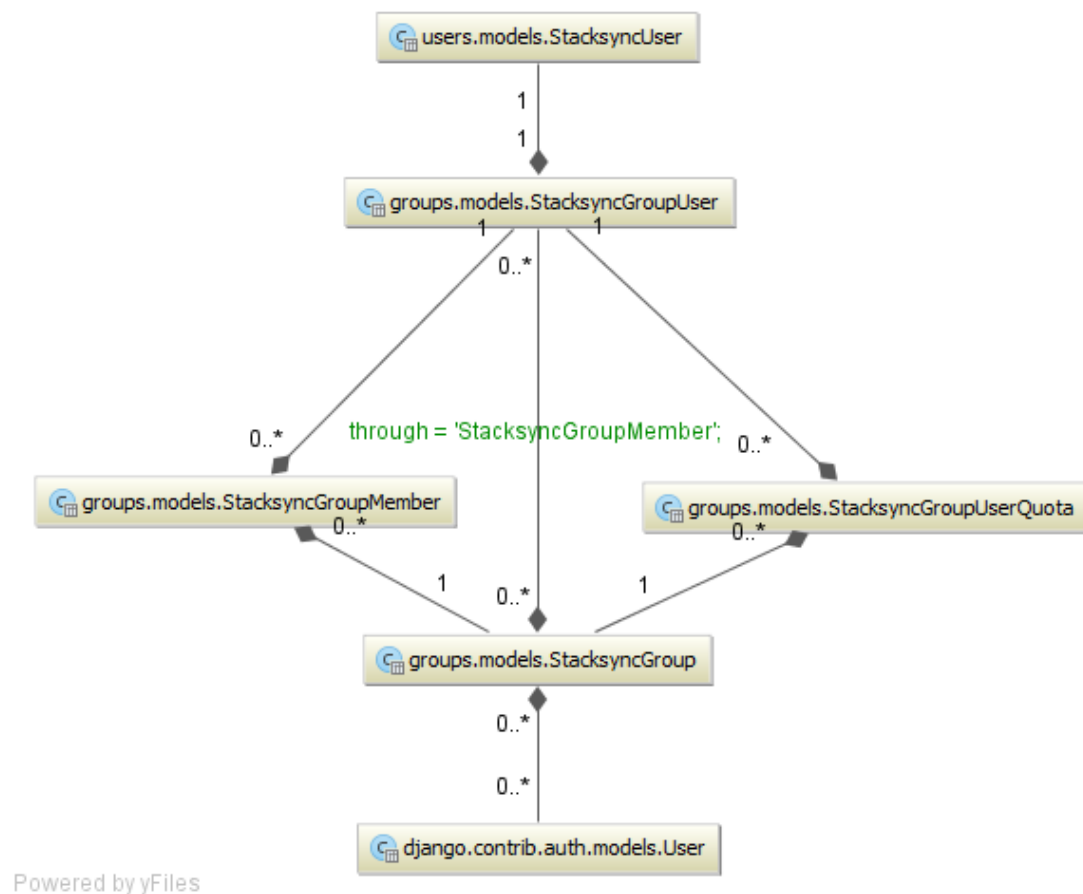
We understand the word manage to mean add/delete/edit capabilities.

We decided to create a new application named "group" in Django.



Obviously since StackSync is an evolving platform we tried to decouple the current user scheme as much as possible from our group development.

In the following diagram, we present the StackSync group application model.



4.2.4 Development of a group-based quota web application

Segmentation of users into groups enables the creation and management of companies with common storage needs.

With this premise in mind we have developed a group management tool, which makes quotas easy to manage by a group administrator, making it easier for entities the management of their own users and disk space.

User management and StackSync groups are managed through the admin panel.

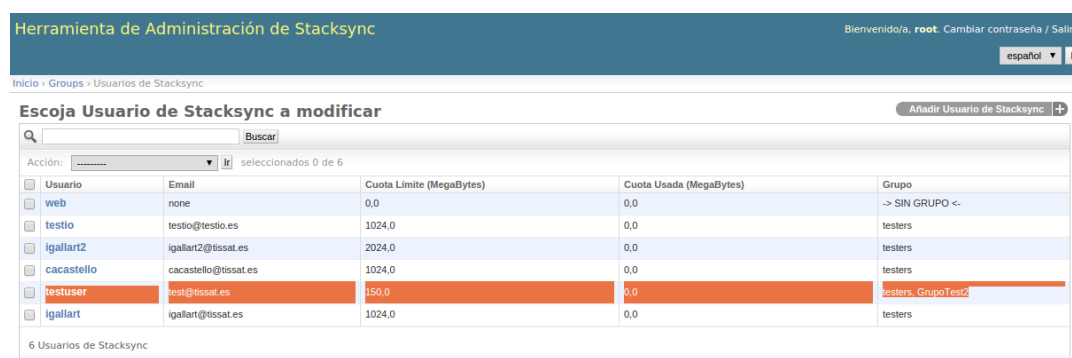


Because generating admin sites for our 'group managers' to add, change and delete users is a repetitive work, we want to provide an easy way to do it.

For that reason, we have chosen Django to automate the user creation using StackSync, through admin interface.

The Django admin panel allows the management of users with a profile of 'group manager', those users will have permissions on the entity that are assigned, in order to manage the creation of its subgroups, users and quotas.

To each Django user can be assigned several permissions (read, edit, delete).



Herramienta de Administración de Stacksync

Bienvenido/a, root. Cambiar contraseña / Salir

español Ir

Inicio > Groups > Usuarios de Stacksync

Escoja Usuario de Stacksync a modificar

Añadir Usuario de Stacksync +

Acción: Ir seleccionados 0 de 6

Usuario	Email	Cuota Limite (MegaBytes)	Cuota Usada (MegaBytes)	Grupo
<input type="checkbox"/> web	none	0,0	0,0	-> SIN GRUPO <-
<input type="checkbox"/> testio	testio@testio.es	1024,0	0,0	testers
<input type="checkbox"/> igallart2	igallart2@tissat.es	2024,0	0,0	testers
<input type="checkbox"/> cacastello	cacastello@tissat.es	1024,0	0,0	testers
<input type="checkbox"/> testuser	test@tissat.es	150,0	0,0	testers, GrupoTest2
<input type="checkbox"/> igallart	igallart@tissat.es	1024,0	0,0	testers

6 Usuarios de Stacksync

Once the users are created in Django, the user can be logged with sufficient permissions, allowing him to can create groups and users for StackSync.

Herramienta de Administración de Stacksync

Bienvenido/a, root. Cambiar contraseña / Salir

español Ir

Inicio > Groups > Usuarios de Stacksync > test@tissat.es

Modificar Usuario de Stacksync

Histórico Ver en el sitio +

Usuario: testuser
25 caracteres max.

Email: test@tissat.es

Password:

Grupos Del Usuario

Grupo Del Usuario: GrupoTest2 - MBY: 50.0 ☐ Eliminar

Grupo: GrupoTest2 +

Cuota (MegaBytes): 50

Grupo Del Usuario: testers - MBY: 100.0 ☐ Eliminar

Grupo: testers +

Cuota (MegaBytes): 100

Grupo Del Usuario: #3

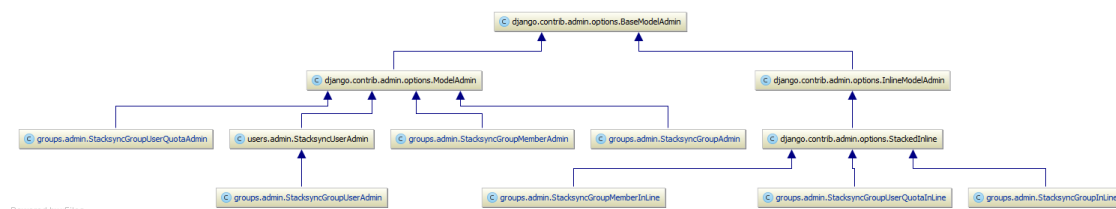
Grupo: +

Cuota (MegaBytes):

+ Agregar Grupo Del Usuario adicional.

Our work was mainly focused on restricting that each group could not interfere with other group's users.

For each model class we have implemented the corresponding ModelAdmin class, in order for the Django admin interface to work.



If we go back to our use case where "A group/company would use certain amount of storage space. They could have some administrative user that can create and delete users, and assign quota to its users.

We need to follow these steps to complete this task.

1. Create a Group Admin with the StackSync permissions

Change user History View on site

Username:
Required: 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: **algorithm:** pbkdf2_sha256 **iterations:** 12000 **salt:** cEmDO8***** **hash:** az6gQ1*****
Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Permissions

☒ **Active**
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☒ **Staff status**
Designates whether the user can log into this admin site.

☐ **Superuser status**
Designates that this user has all permissions without explicitly assigning them.

Groups: The groups this user belongs to. A user will get all permissions granted to each of his/her group. Hold down "Control", or "Command" on a Mac, to select more than one.

Available groups

Chosen groups
stacksync

[Delete](#) [Save and add another](#) [Save and continue editing](#) [Save](#)

2. Create StackSync Group with quota

Stacksync administration Welcome, **admin**. Change password / Log out

Home > Groups > Stacksync groups > Add stacksync group

Add stacksync group

Name:

Quota (GB): **Quota used:** 0

Admins:

Hold down "Control", or "Command" on a Mac, to select more than one.

[Save and add another](#) [Save and continue editing](#) [Save](#)

3. Now the group/customer manager can access as an admin to create his own users, memberships, and quotas.

Stacksync administration Welcome, **customer1**. Change password / Log out

Site administration

Groups

Stacksync group members	Add Change
Stacksync group user quotas	Add Change
Stacksync groups	Change
Stacksync users	Add Change

Recent Actions

My Actions
None available

At this use case, the manager creates a user, then creates a membership and at last creates a quota.

A Django user can be an administrator of one or more groups.

A group can have multiple administrators.

A group can be assigned a quota limit on MB, being an advantage for the entity being able to manage more efficiently the way it allocates disk space. So we can control that a user has not an excessive fee.

The quota limit is the maximum quota that can be assigned to the user of that group (but to each user can be assigned a quota lower than the maximum quota of the group to which it belongs)

For instance:

Group: test1
max quota = 1000 MB.

User: user1
quota: 512 MB.

User: user2
quota: 1000 MB.

Users in this group may have different quotas, but if you try to assign more than the maximum allowable quota, the system will not let you assign it.

A user can belong to one or more groups.

- Each user can have different quotas in each group.
- The maximum user quota is the sum of its maximum quotas.

For instance:

If a user is assigned to 2 groups:

Group 1: quota allocated 512 MB.

Group 2: quota allocated 256 MB.

His total quota is the sum of $512 + 256 = 768$ MB.

4.2.5 Migrate web interface back-end from PHP to Python

Because OpenStack is programmed in Python, although the prototype initial web interface was generated in PHP for training reasons, it was considered appropriate to use a single programming language for all development environments, making easy to the community, expanding and participating at the project by limiting the number of programming languages.

During our development with PHP, we encountered the issue that we often had to use 3rd party frameworks or develop our own classes to interact with the Openstack platform, therefore we noticed that a change to Python could help gain a more in-depth knowledge about the underlying Openstack platform, and in that way apply the current good developing practices and in doing so improving the quality of our code.

It made sense to make a change on the project before advancing more with new applications.

Other point was the fact that there was a previous satisfactory experience using the Django framework in the developer team, so that could ease the transition. Also, since the previous user interface was being developed in bootstrap we were confident that the end result, would the same.

The Django framework enables us to use a powerful scaffolding system that could help us develop simple applications very easy and rapidly.

In order to interact with StackSync we developed a web client and an administrative interface, using freely distributed components.

On the same train of thought, we recognized that if there was a moment of doubt, we could take the Horizon dashboard as an example to follow.

Now we have an easier codebase with lesser dependencies and more tested since we are using standard tools.

4.2.6 Refactoring of StackSync web client

Here we show the new user interface, where you can see the space of your physical quota, in case it had been defined.

The screenshot shows the Stack Sync web interface. At the top, there's a header with the Stack Sync logo, user information (User: al@al.com), and links for Contact us and Logout. Below the header is a progress bar at 60% and two buttons: 'add files' and 'Create new folder'. The main content area displays a table with columns 'Name', 'Size', and 'Last modified'. The table lists three items: 'Carpeta' (Size: --, Last modified: 04/07/2014 12:07), 'Aa' (Size: --, Last modified: 21/07/2014 01:07), and 'Create Consumer_Oauth.Txt' (Size: 968 bytes, Last modified: 04/07/2014 12:07). Below the table, there's a code diff showing changes to a file. The diff highlights several lines of Python code, including the definition of the 'Api' class, the 'metadata' method, and the 'metadata_focus' method. The code is shown in a side-by-side comparison, with the original code on the left and the refactored code on the right. The refactored code is more concise and includes additional functionality for folder sharing.

Here we show a refactoring gone from an original file of 247 lines to 185 lines, while adding new functionality such as folder sharing.

4.2.7 Implemented a folder sharing functionality for StackSync users

We developed a web functionality to add share folder functionality to the web client.

The screenshot shows the Stack Sync web interface. At the top, there's a header with the Stack Sync logo and a progress bar at 60%. Below the header is a table with columns 'Name' and 'Size'. The table lists three items: 'Carpeta' (Size: --), 'Aa' (Size: --), and 'Create Consumer_Oauth.Txt' (Size: 968 bytes). A context menu is open over the 'Carpeta' item, showing options: 'Delete', 'Rename', 'Move', and 'Share'. The 'Share' option is highlighted.

```
def share_folder(self, folder_id, allowed_user_emails=[],
access_token_key=None, access_token_secret=None):
    headeroauth, headers = self.get_oauth_headers(access_token_key,
access_token_secret)
    url = self.DEFAULT_FOLDER_URL + str(folder_id) + '/share'
```

```
json_payload = json.dumps(allowed_user_emails)

r = requests.post(url, data=json_payload, auth=headerauth,
headers=headers)
return r
```

Share folder

×

Please enter the email addresses that the folder will be shared with.

Close

Save changes

```
def get_members_of_folder(self, folder_id, access_token_key=None,
access_token_secret=None):

    headerauth, headers = self.get_oauth_headers(access_token_key,
access_token_secret)
    url = self.DEFAULT_FOLDER_URL + str(folder_id) + '/members'

    response = requests.get(url, auth=headerauth, headers=headers)

    if response.status_code == 200:
        return response.json()
    else:
        response.reason = response.reason + ". " + response.content
        response.raise_for_status()
```

Share folder

×

Please enter the email addresses that the folder will be shared with.

al@al.com x john.doe@yahoo.com x

Close

Save changes

User: al@al.com

You have shared this folder with the x following users: john.doe@yahoo.com

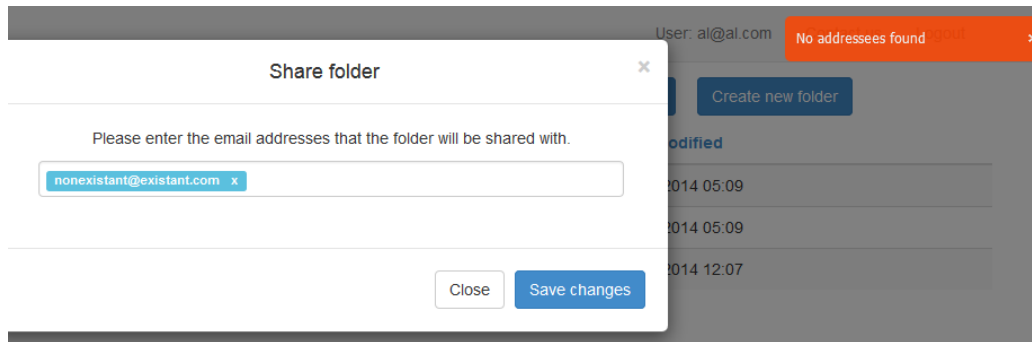
Create new folder

modified

2014 05:09

2014 05:09

2014 12:07



We can verify the correct behavior of the functionality with this functional test.

```
def test_get_folder_members(self):
    name_of_folder = self.create_folder()
    folder_td_element = self.find_folder(name_of_folder)[0]
    folder_members = self.get_folder_members(folder_td_element)

    self.assertEqual(0, len(folder_members))

def test_share_a_folder(self):
    users = ['al@al.com', 'walter.smith@stacksync.com']
    name_of_folder = self.create_folder()
    folder_td_element = self.find_folder(name_of_folder)[0]

    self.show_context_menu(folder_td_element)

    menu_share_option = self.browser.find_element_by_css_selector('#jqContextMenu > ul > li#share')
    menu_share_option.click()
    folder_members = self.browser.find_elements_by_css_selector('#folder-members option')

    for user in users:
        self.input_members_email_in_folder(user)

    # Saves current members
    self.browser.find_element_by_id('save-member-button').click()

    # Closes modal window
    close_button = '#share-folder-modal > div.modal-dialog > div.modal-content > div.modal-footer > button'
    self.browser.find_element_by_css_selector(close_button).click()

    folder_members = self.get_folder_members(folder_td_element)
    self.assertEqual(3, len(folder_members))

    self.browser.find_element_by_css_selector(close_button).click()
```


4.2.8 StackSync support for the ownCloud application

One goal of the project was to demonstrate the interoperability between different networks, within the objectives of analyzing and promoting standardization in communication of data, between different clouds. It was selected as mockup for it, one of the best known free software cloud products: ownCloud.

We have started the integration studies for sharing between ownCloud and StackSync platforms.

We have setup an ownCloud server 7.0.2 to use Openstack Swift storage, we tested it in order to analyze if it worked properly with cloudspaces infrastructure, and it was determined that in order for ownCloud's synchronization, to update its database we had to use a development version (daily build).

When using ownCloud, we first uploaded a file to a web server running ownCloud, and after that, the server uploaded our file to our swift container.

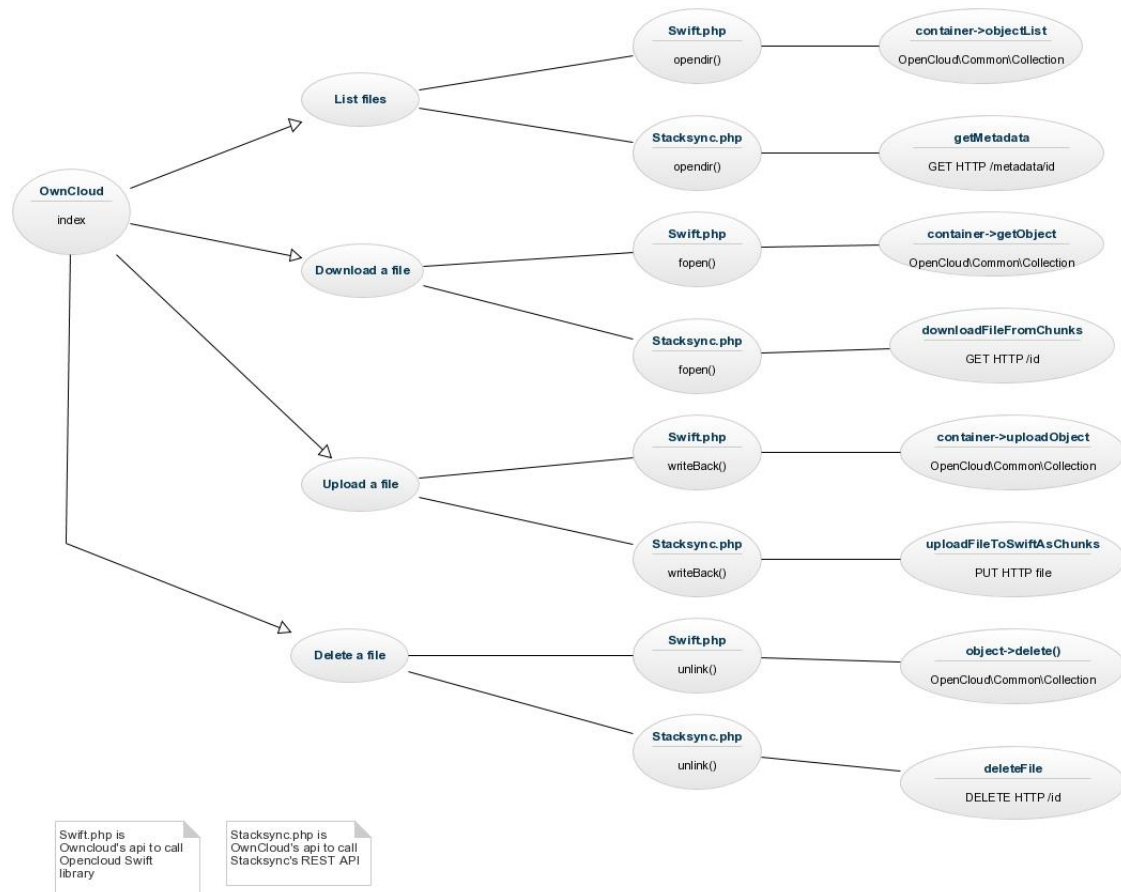
When you download a file from swift, you first download onto the webserver's temporal folder and then you download it from the web server.

In order to create a StackSync implementation we can take notes of how OwnCloud is interacting with Openstack Swift; we only need to replace the native upload/download/delete /list file operations for the corresponding StackSync calls.

We would have to create a \OC\Files\Storage\Stacksync.php service, its physical location would be in "owncloud\apps\files_external\lib" which contained all three basic function calls PUT, GET, GET METADATA, DELETE.

We've chosen to compare with Opencloud Swift, which is the current Openstack library for OwnCloud, we have documented how to replace regular Swift calls with their StackSync equivalents.

In the following diagram, we compare the current functionalities of Owncloud, and how it would look when running a StackSync application.



Then we would have to register our StackSync application with OwnCloud kernel as shown in owncloud/apps/files_external/appinfo/app.php

```
OC::$CLASSPATH['OC\Files\Storage\Stacksync']
'files_external/lib/stacksync.php';

OC_Mount_Config::registerBackend('\OC\Files\Storage\Stacksync', array(
    'backend' => (string)$l->t('Stacksync Storage'),
    'priority' => 100,
    'configuration' => array(
        'user' => (string)$l->t('Username'),
        'bucket' => (string)$l->t('Bucket'),
        'region' => '&'.$l->t('Region (optional for OpenStack Object Storage)'),
        'tenant' => '&'.$l->t('Tenantname (required for OpenStack Object Storage)'),
        'password' => '&*'.$l->t('Password (required for OpenStack Object Storage)'),
        'service_name' => '&'.$l->t('Service Name (required for OpenStack Object Storage)'),
        'url' => '&'.$l->t('URL of identity endpoint (required for OpenStack Object Storage)'),
        'timeout' => '&'.$l->t('Timeout of HTTP requests in seconds'),
    ),
    'has_dependencies' => true));
```

Now, we can start by showing differences of how ownCloud is using Opencloud to operate with an Openstack system, and how it would be different when working with a StackSync system.

For example:

Basically all file operations use the basicOperation functionality, inside files/view.php, we notice that \$storage->\$operation, storage is an OC\Files\Storage\Swift object, and \$operation (file_exists) means that we have to execute the 'file_exists' operation.

<https://github.com/owncloud/core/blob/d15c3e4030d1ed6fad6e258b758a79c520d6bd39/lib/private/files/view.php#L750>

```
private function basicOperation($operation, $path, $hooks = array(),
$extraParam = null) {
    *****omitted*****
    $run = $this->runHooks($hooks, $path);
    list($storage, $internalPath) = Filesystem::resolvePath($absolutePath .
$postFix);
    if ($run and $storage) {
        if (!is_null($extraParam)) {
            $result = $storage->$operation($internalPath, $extraParam);
        } else {
            $result = $storage->$operation($internalPath);
        }

        $result = \OC_FileProxy::runPostProxies($operation, $this-
>getAbsolutePath($path), $result);
        *****omitted*****
    }
}
```

- Upload a file

https://github.com/owncloud/core/blob/c88d517e8879c56755bc26f604d515d9772b35b3/apps/files_external/lib/swift.php#L503

Inside the ownCloud's Swift library the software executes the writeBack function when you need to upload a file.

```
public function writeBack($tmpFile) {
    if (!isset(self::$tmpFiles[$tmpFile])) {
        return false;
    }
    $fileData = fopen($tmpFile, 'r');
```

```
$this->container->uploadObject(self::$tmpFiles[$tmpFile], $fileData);  
unlink($tmpFile);  
}
```

This line `$this->container->uploadObject` does the heavy work of actually uploading a file into a container.

So we can conclude that if we manage to create a service class `Stacksync.php` that will require a `writeBack` function overridden like this.

```
public function writeBack($tmpFile) {  
    if (!isset(self::$tmpFiles[$tmpFile])) {  
        return false;  
    }  
    $fileData = fopen($tmpFile, 'r');  
    $this->uploadFileToSwiftAsChunks($tmpFile, $tmpFile->name, $tmpFile->parent);  
    unlink($tmpFile);  
}
```

(This example was based on `stacksync-server 0.4.4`)

```
public function uploadFileToSwiftAsChunks($localFilePath, $fileName,  
    $parent=null)  
{  
  
    $fd = fopen($localFilePath, "r");  
    $options = array(  
        CURLOPT_URL => $this->url . '/' . $this->container . '/files?file_name=' .  
$fileName . '&parent=' . $parent,  
        CURLOPT_RETURNTRANSFER => true,  
        CURLOPT_CUSTOMREQUEST => "PUT",  
        CURLOPT_VERBOSE => true,  
        CURLOPT_INFILE => $fd,  
        CURLOPT_INFILESIZE => filesize($localFilePath),  
        CURLOPT_CAINFO => dirname(__FILE__) . '/certs/ca-swift.pem',  
        CURLOPT_SSL_VERIFYHOST => false,  
        CURLOPT_UPLOAD => true,  
        CURLOPT_HTTPHEADER => array('stacksync-api:true',  
            'X-Auth-Token:' . $this->tokenId),  
    );  
  
    $message = $this->httpClient->exec($options);  
    fclose($fd);  
  
    return json_decode($message, false);  
}
```

```
}
```

- Download a file:

Owncloud executes a function called `fopen()` when you need to download a file:

https://github.com/owncloud/core/blob/ebf7758d1027709e29038540d6dc267015f45296/apps/files_external/lib/swift.php#L316

We would create the equivalent of `fopen()` and substitute the line `$object = $this->container->getObject($path);` by a function similar to this:

```
public function downloadFileFromChunks($id) {
    $fd = fopen($id, "w+");
    $options = array(
        CURLOPT_URL => $this->url . '/' . $this->container . '/files?file_id=' .
    $id,
        CURLOPT_FILE => $fd,
        CURLOPT_CAINFO => dirname(__FILE__) . '/certs/ca-swift.pem',
        CURLOPT_SSL_VERIFYHOST => false,
        CURLOPT_HTTPHEADER => array('stacksync-api:true',
            'X-Auth-Token:' . $this->tokenId),
    );

    $contents = $this->httpClient->exec($options);
    fclose($fd);

    return $contents;
}
```

- Delete a file:

Owncloud executes the following `unlink` function when you need to delete a file:

https://github.com/owncloud/core/blob/ebf7758d1027709e29038540d6dc267015f45296/apps/files_external/lib/swift.php#L299

```
public function deleteFile($id) {

    $options = array(
        CURLOPT_URL => $this->url . '/' . $this->container . '/files?file_id=' .
    $id,
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_CUSTOMREQUEST => "DELETE",
        CURLOPT_VERBOSE => true,
        CURLOPT_CAINFO => dirname(__FILE__) . '/certs/ca-swift.pem',
        CURLOPT_SSL_VERIFYHOST => false,
```

```
CURLOPT_HTTPHEADER => array('stacksync-api:true',  
    'X-Auth-Token:' . $this->tokenId),  
);  
  
$message = $this->httpClient->exec($options);  
  
return json_decode($message, false);  
}
```

- List folder:

As we can see in view.php:965, here we get the data from SQL, inside the function of public function getFileInfo. We want to stress this part because Owncloud reads its folder information from SQL.

```
if ($cache->getStatus($internalPath) < Cache\Cache::COMPLETE) {  
    $scanner = $storage->getScanner($internalPath);  
    $scanner->scan($internalPath, Cache\Scanner::SCAN_SHALLOW);  
} else {  
    $watcher = $storage->getWatcher($internalPath);  
    $watcher->checkUpdate($internalPath);  
}
```

And if you need to check for updates, must be used scanner.php:255, OC\Files\Cache\Scanner->scanChildren(), then we can start reading files from our resource(Swift in our case).

An interesting functionality is the way that a regular swift folder is listed:

https://github.com/owncloud/core/blob/ebf7758d1027709e29038540d6dc267015f45296/apps/files_external/lib/swift.php#L210

```
public function opendir($path) {  
    *****ommitted*****  
    try {  
        $files = array();  
        /** @var OpenCloud\Common\Collection $objects */  
        $objects = $this->container->objectList(array(  
            'prefix' => $path,  
            'delimiter' => '/'  
        ));  
  
        /** @var OpenCloud\ObjectStore\Resource\DataObject $object */  
        foreach ($objects as $object) {  
            $file = basename($object->getName());  
            if ($file !== basename($path)) {
```

```

        $files[] = $file;
    }
}

\OC\Files\Stream\Dir::register('swift' . $path, $files);
return opendir('fakedir://swift' . $path);
} catch (\Exception $e) {
    \OCP\Util::writeLog('files_external', $e->getMessage(),
\OCP\Util::ERROR);
    return false;
}
}

```

As we can see in the container calls object list which makes the proper HTTP call to the proxy server. Now we could replace the container object list with our own StackSync call.

Here's a prototype for a getter metadata function in StackSync.

```

public function getMetadata($fileId=null){
    $options = array(
        CURLOPT_URL => $this->url . "/" . $this->container .
"/metadata?file_id=". $fileId .'&list=true',
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_HTTPHEADER => array('stacksync-api:true',
        'X-Auth-Token:' . $this->tokenId),
        CURLOPT_CAINFO => dirname(__FILE__) . '/certs/ca-swift.pem',
        CURLOPT_SSL_VERIFYHOST => false,
    );

    $contents = $this->httpClient->exec($options);

    if (defined('JSON_BIGINT_AS_STRING')){
        $contents = json_decode($contents, false, 512,
JSON_BIGINT_AS_STRING);
    } else {
        $contents = json_decode($contents, false, 512);
    }

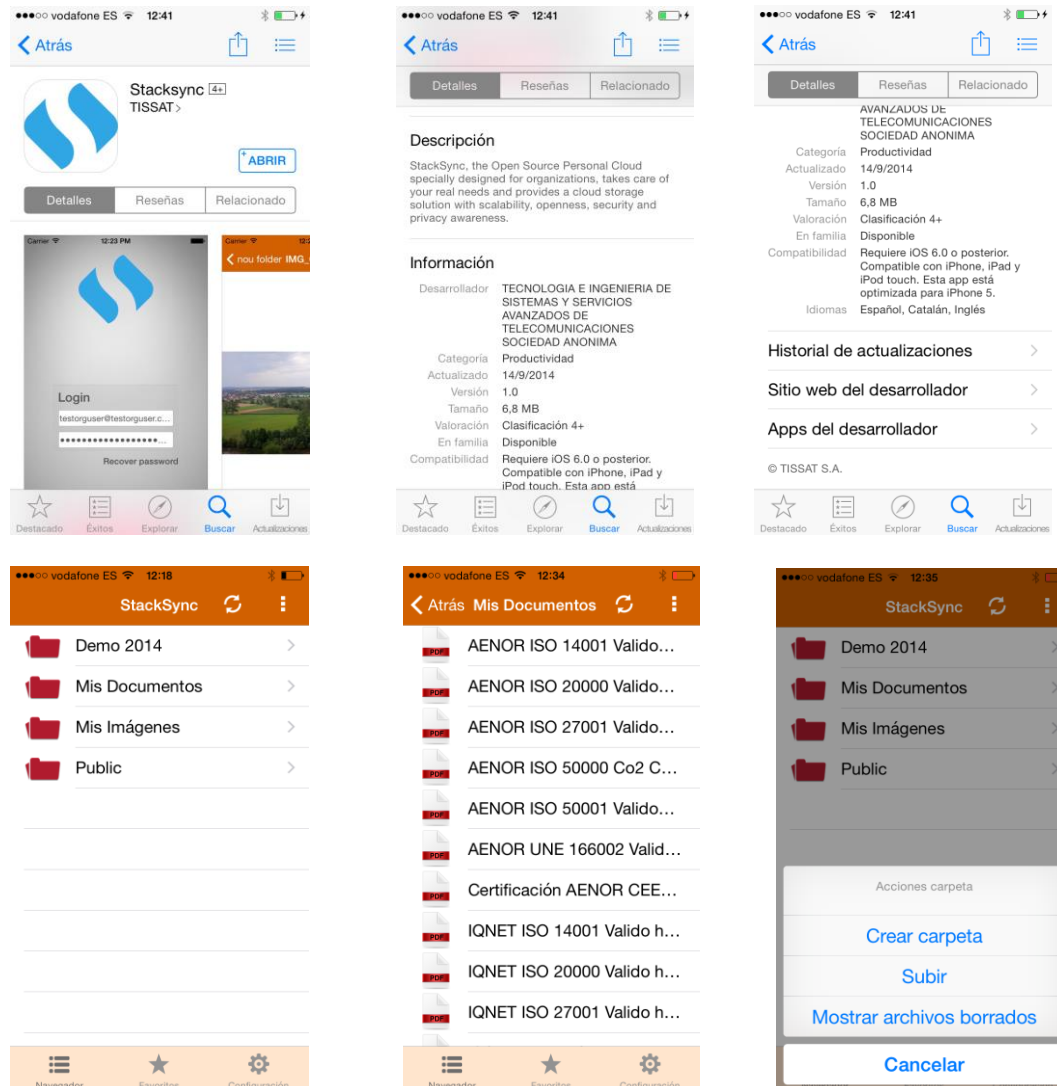
    usort($contents->contents, array($this,
"compareSortOrderByFoldersFirst"));
    return $contents;
}

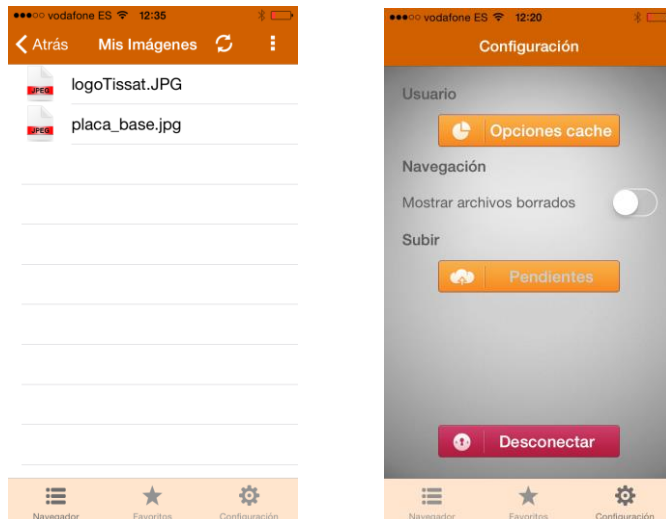
```

4.2.9 Development of IOS App

We have developed and published in App Store, a native application to manage files and folders in a secure way, through StackSync.

The development of this application provides mobile access to the platform, increasing the number of operating systems supported by StackSync.





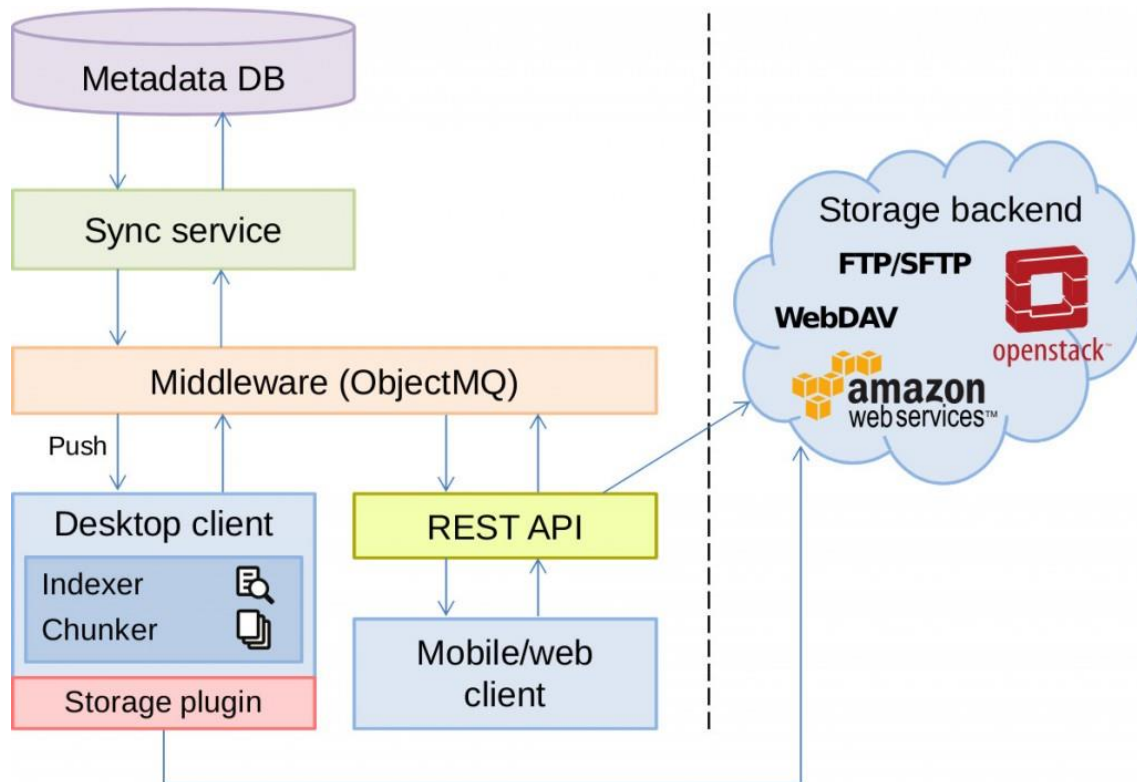
This prototype application extends the growth potential of the solution, in addition to serve as a basis for developing more advanced versions incorporating file sharing and also new functionalities and advanced services on the platform.

4.3 Adding Security Features to StackSync Components

The cloud encryption process requires encryption of the existing communications between the components of the platform. On one hand, some of them are public and encryption is essential, as is the case of communication of the storage components, StackSync servers and web servers. On the other hand, within the subnet management platform, the transmission of information is also encrypted for greater security, in the event of an unlikely access to information, from within the internal subnet of the platform.

Finally, a further aspect that can be implemented, is full encryption of the information within the storage spaces. This has been approached from the standpoint of unique key (a private key for all users of the platform, for encrypting users data on the disks), which allows the regeneration of the users password if they forget it, but guarantees the encryption of data on the server. The alternative is to encrypt the data on disk using the user's password. This would be the highest level of security, but has been considered as an option for private networks that require high security, because of the risk of loss of user information, in case the password is lost.

In order to understand better the needs for encryption between the components of the platform, we will describe the architecture of the platform:



Metadata DB: Database (PostgreSQL) where are saved the metadata files that the storage backend can not store. Example: file size, file name, user, creation date, file version, modified date...

Sync service: An entity that is responsible for processing the metadata. Responsible for interacting with the database server and StackSync.

Middleware (ObjectMQ): A software responsible for the exchange of information between applications. Is in charge interact with the desktop client and the REST API and the SackSync server.

We use RabbitMQ Middleware that is an open source software, used for message negotiations, and falls within the category of messaging middleware.

Desktop client: It is the application that interacts with the user, the application is able to interact with the backend storage to store the files and with the StackSync server to store the metadata.

REST API: This module allows us to create services and applications that can be used by any device or client that uses HTTP.

The REST API is stored in the Storage backend, in this project it is a plugin that is installed on OpenStack.

Mobile / Web client: Mobile applications (Android, IOS) and web clients capable of interacting with the REST API in order to communicate with StackSync.

Storage backend: It is responsible of the storage of files, in this use case we use OpenStack as a platform for data storage.

4.3.1 Encryption. Android version

The communication of the mobile application Android is done through API StackSync. To ensure encryption of data sent, it has implemented a system of encryption "AES / CBC / PKCS5Padding".

The block encryption algorithms such as AES spread the message in fixed-size blocks for processing. The way in which these blocks are managed is called "encryption mode", in our case we used the CBC.

CBC mode encryption (Cipher Block Chaining Mode) is an ECB extension that adds safety (using an initialization vector IV). It is the encryption mode by blocks more used.

The simplest ECB is the electronic codebook (ECB) mode, in which messages are divided into blocks and each is coded separately using the same key K.

Both the mobile application and the web client has an option to encrypt / decrypt files.

Each application must keep the same key, to decrypt what has been encrypted with another application.

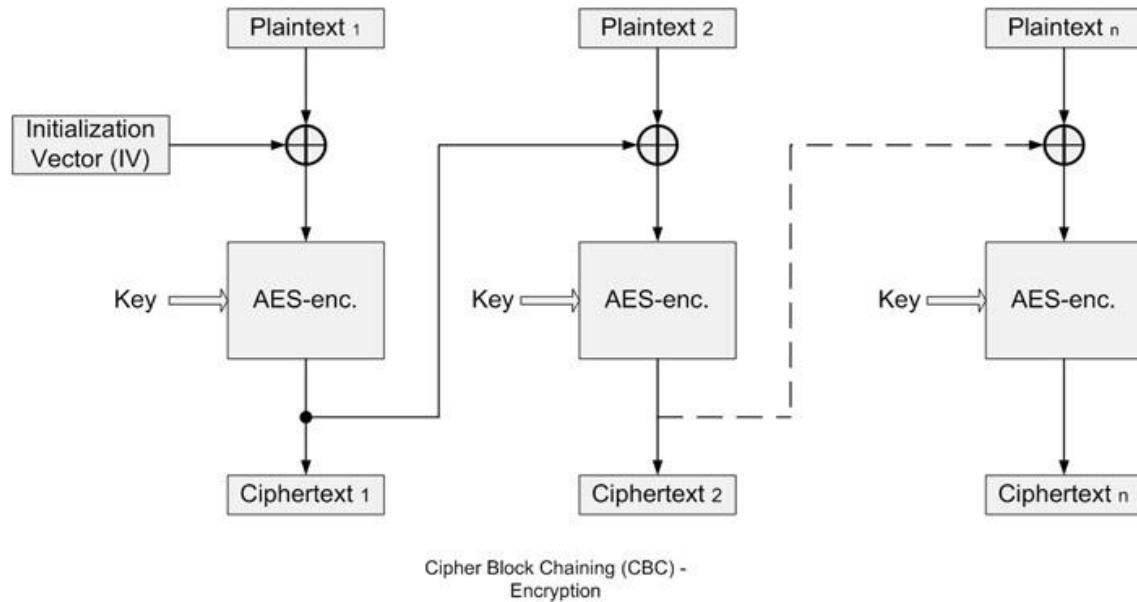
Therefore storage of the keys is performed on each application, both mobile and desktop applications must have the same key K to encrypt and decrypt, the files uploaded or downloaded.

In the webclient we have included also an icon (lock icon) to allow the user to enter the password with which he wants to encrypt / decrypt.

In the mobile client there is an option to enter the encryption key and stored in the mobile application itself.

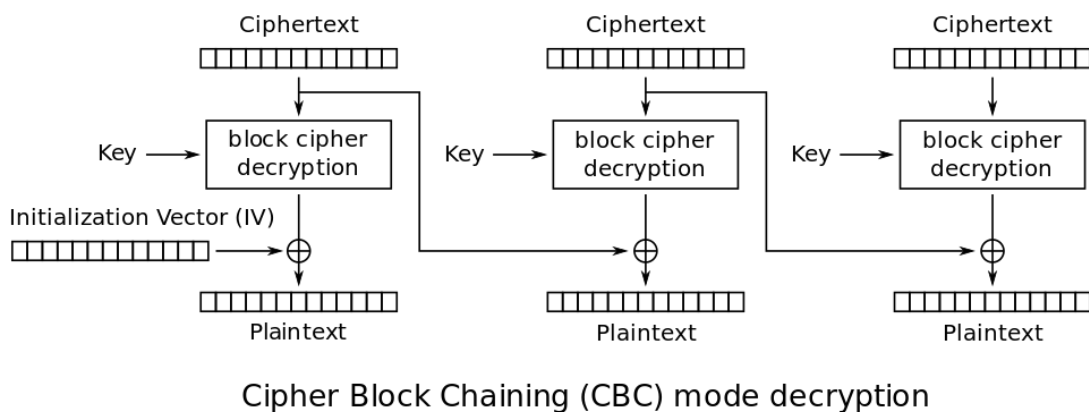
This additional encryption, gives a user the chance to be sure not only that the communication is encrypted (using https), but also that the encryption of the file itself is done.

In the cipher-block chaining mode (CBC), to each block of plaintext is applied XOR operation with the previous encrypted block before being encrypted.



In this way, each ciphertext block depends on all the plaintext processed up to this point. In order to make unique each single message, it is also used an initialization vector.

PKCS5Padding is a filler mechanism to define 8 bytes chains



4.3.2 Encryption. StackSync Mobile App for IOS

The communication of the mobile application iOS is done through API StackSync. To ensure encryption of data sent, it has implemented a system of encryption by using the library RNCryptor/RNCryptor.

Encryption can be used in almost all applications but is especially important in network applications for ensure the security of the information transmitted between two devices. In the IOS version we include RNCryptor library that allows to exchange information securely.

RNCrypto is a library used for development in Objective-C (IOS) that is also available in other platforms such as C ++, JAVA, etc.

RNCrypto includes:

- AES-256 encryption
- CBC mode
- Password stretching with PBKDF2
- Password salting
- Random IV
- Encrypt-then-hash HMAC

RNCryptor supports asynchronous use, specifically designed to work with NSURLConnection. This is also useful for cases where the encrypted or decrypted data will not comfortably fit in memory. If the data will comfortably fit in memory, asynchronous operation is best achieved using `dispatch_async()`.

4.3.3 Implementation of encryption. Web Client (Python)

The implementation of encryption for the StackSync Web client, is based on digital content encryption using HTTPS certificate, ensuring the secure transfer of files using AES in CBC mode, by using the python cryptography tools "pycrypto".

Pycrypto is a collection of cryptographic modules that implements various algorithms and protocols for Python programming language. The encryption toolkit Python aims to provide a reliable and stable base to write Python programs that require cryptographic functions.

A central aim has been to provide a simple interface consisting of similar classes of algorithms. For example, all block objects have the same encryption methods and return values, and support the same functions of information.

Hash functions have a different interface, but is also constant in all hash functions available.

Some of these interfaces are encoded as documents of proposals for improving Python, PEP 247, "API for cryptographic hash functions" and PEP 272, "API block encryption algorithms."

The aim is to make it easy the replacement of old algorithms with the new ones, which are safest.

4.3.4 Encryption. Admin Web Client (Python)

The implementation of encryption for the StackSync Admin Web client, is based on digital content encryption using HTTPS certificate, ensuring the secure transfer of files using AES in CBC mode, by using the python cryptography tools "pycrypto".

Pycrypto is a collection of cryptographic modules that implements various algorithms and protocols for Python programming language. The encryption toolkit Python aims to provide a reliable and stable base to write Python programs that require cryptographic functions.

A central aim has been to provide a simple interface consisting of similar classes of algorithms. For example, all block objects have the same encryption methods and return values, and support the same functions of information.

Hash functions have a different interface, but is also constant in all hash functions available.

Some of these interfaces are encoded as documents of proposals for improving Python, PEP 247, "API for cryptographic hash functions" and PEP 272, "API block encryption algorithms."

The aim is to make it easy the replacement of old algorithms with the new ones, which are safest.

4.3.5 Encryption. StackSync Desktop Client

The implementation encryption of desktop applications was performed using HTTPS. The encryption of the applications, written in Java, has been implemented for the latest versions of Windows and Linux.

In order to grant the security of the information at the desktop client, it is required encryption between the client and two different components, since it is

necessary to encrypt the connection between the client and the storage environment (OpenStack Swift) for the transmission of information, and between the client and metadata manager (StackSync).

To encrypt the connection between the StackSync server and the client desktop, RabbitMQ server configuration is required to accept secure SSL connections.

SSL (Secure Sockets Layer) are cryptographic protocols that provide secure communications over a network.

In the SSL X.509 certificates are used and therefore is used asymmetric cryptography to authenticate the partner with whom you are communicating, and to exchange a symmetric key. This session is then used to encrypt the data stream between both parts.

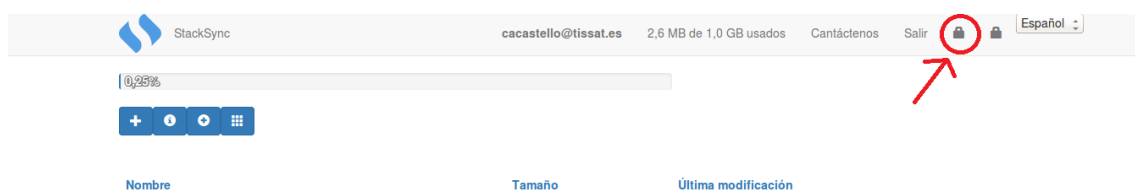
This ensures confidentiality for data and message, and the control via message authentication code, for integrity and message authentication.

4.4 Interoperability between clouds. Integration with ownCloud

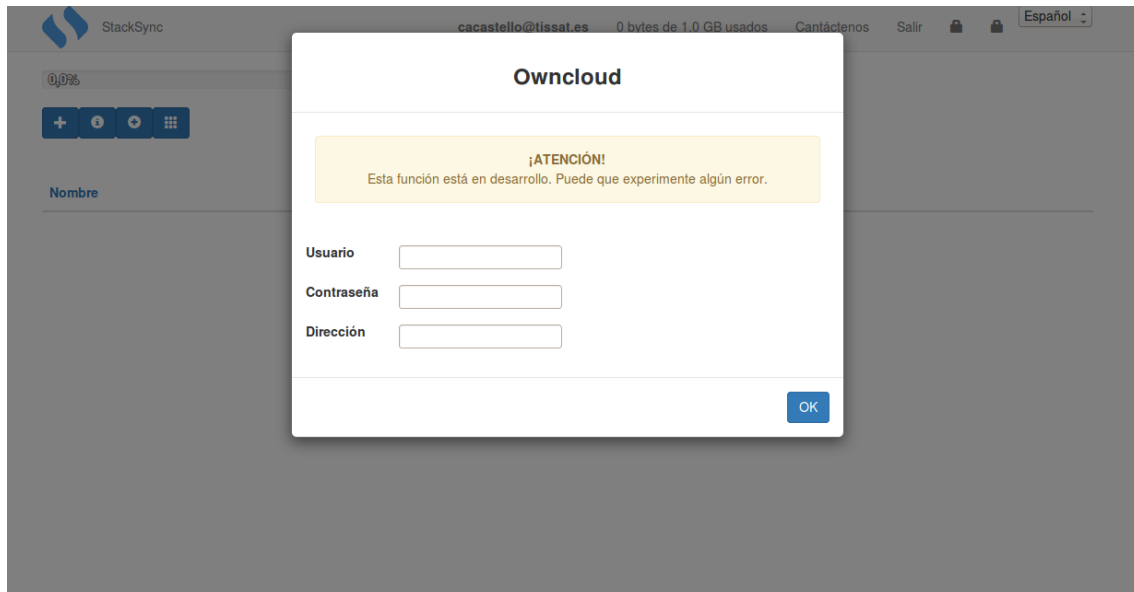
In July 2015, Google announced its support for OpenStack Foundation as a corporate sponsor, which gives an important impulse to the cloud platform project, and expands opportunities for interoperability.

While the ways of integration with the Google platform are defined, and in order to demonstrate the ability to integrate with other platforms, ownCloud was selected as a target for integration, for its open source nature.

A user from the web interface can be interconnected with an ownCloud platform, the StackSync web interface includes a new icon for interconnection, which allows the user to input its credentials, to allow interoperability between clouds.



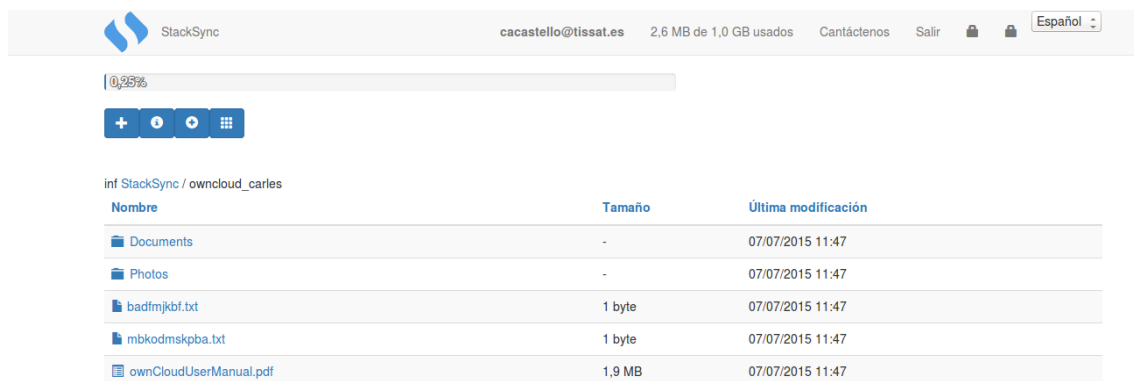
By clicking on the icon, a window requesting the identification for owncloud is shown, where the user can define the user name, password and address of the ownCloud platform.



The following figure shows the interface of a user, and his data, created on an ownCloud platform.



The StackSync user, through its web platform, and after entering the data connection information to the owncloud network, allows to grant access not only to their information at StackSync, but also to the files and folders inside the ownCloud network, as if he was on a single platform.



The application has the restriction of not allowing upload files with the same name in the same directory, but it is allowed with different directories.

Technically, the connection with owncloud is performed by an external library by using the WebDAV protocol implemented in ownCloud: easywebdav.

This library allows to freely move the user navigation through WebDAV file structure (ownCloud) from the StackSync platform.

4.5 Traces Analysis

Several platforms have been deployed, each one containing different test users, in order to properly test the peculiarities of each configuration.

Although the number of users per platform, and therefore the number of traces obtained, are small, the significance of this separation justifies a separate analysis of the platforms.

One of the platforms of the project incorporates the recent updates of StackSync, including sharing. The analysis of this information allows us to draw conclusions, for example, that users with the highest number of shared folders tend to store more files, and to use the platform more frequently.

Another platform incorporates encryption systems in communications between the storage platform and the mobile applications, desktop and web. The analysis of use of this platform serves to analyze the file load times, and to get indicators for different file sizes. It has been useful for the debug of the software, by improving the average load times and capacity of the platform.

It has been also used a more powerful platform for load testing of concurrent users, by forcing from different users and applications, a heavy use of upload and download of files, which has helped us to improve the functioning of the platform and start the study of its scalability.

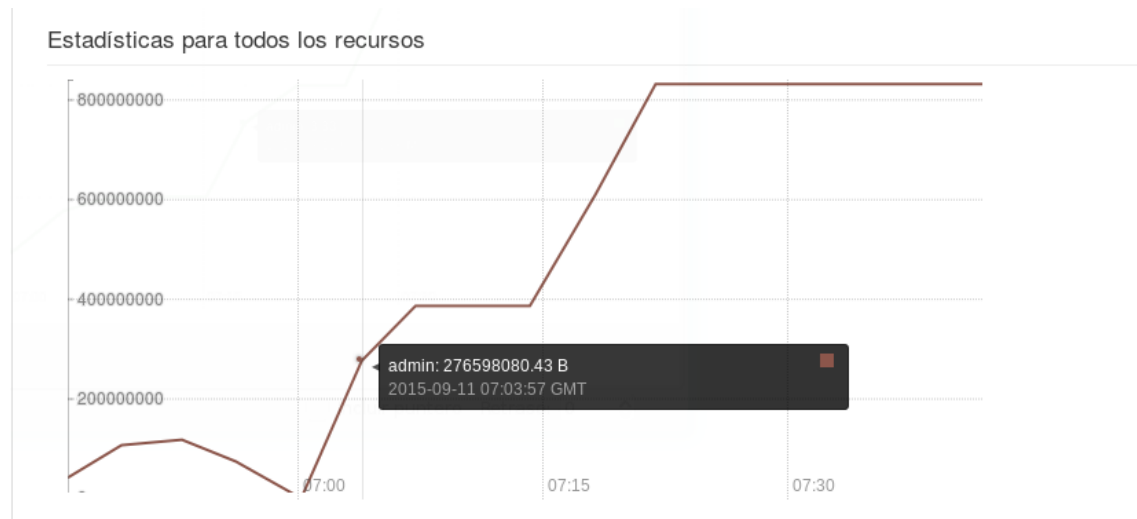
Currently there is a test version platform used by actual customers of the legal environment, Which requires compliance with the law in the most demanding level of data protection, in a demo environment. If the test results are satisfactory, the tool will be sold through a licensing system, with cost proportional to the number of users and disk space.

Next, we draft some examples of the indicators of measurement taken.

First of the indicators shows the mean number of objects, that a user is storing at a certain time.



Another indicator shows for a concrete user the size of the files that are stored in a given time. This is shown in bytes. At this example the user is using approximately 263MB of storage space.



A third indicator shows the mean number of containers of the user at a given time. The graph show that the user had an average of four containers created.



Annexes

Annex 1. Configuration file

The cloud configuration file is found at the address “/var/www/eyeos/eyeos/extern/u1db/” and is called “settings.py”.

```
"Clouds":{
  "Stacksync": {
    "urls": {
      "REQUEST_TOKEN_URL": "http://api.stacksync.com:8080/oauth/request_token",
      "ACCESS_TOKEN_URL": "http://api.stacksync.com:8080/oauth/access_token",
      "CALLBACK_URL": "http://192.168.100.50",
      "RESOURCE_URL": "http://api.stacksync.com:8080/v1/",
      "OAUTH_URL": "http://api.stacksync.com:8080/oauth/authorize?oauth_token="
    },
    "consumer": {
      "key": "8224c4148302d09e287ba35fe96f214e0ac5b3c5c",
      "secret": "f88a48225a02542ec720dc18cba36023"
    },
    "version": "v2",
    "controlVersion": "true",
    "comments": "true",
    "calendar": "true"
  },
  "NEC": {
    "urls": {
      "REQUEST_TOKEN_URL": "http://csdev.necccloudhub.com:1080/oauth/request_token",
      "ACCESS_TOKEN_URL": "http://csdev.necccloudhub.com:1080/oauth/access_token",
      "CALLBACK_URL": "http://192.168.100.50",
      "RESOURCE_URL": "http://csdev.necccloudhub.com:1080/api/cloudspaces/",
      "OAUTH_URL": "http://csdev.necccloudhub.com:1080/oauth/Authorize.aspx?oauth_"
    },
    "consumer": {
      "key": "8224c4148302d09e287ba35fe96f214e0ac5b3c5c",
      "secret": "f88a48225a02542ec720dc18cba36023"
    },
    "version": "v2",
    "controlVersion": "false",
    "comments": "true",
    "calendar": "true"
  }
}
```

Below is a list of components that make up the configuration of a specific cloud, in our case “Stacksync” or “NEC”:

version	Version of the Oauth API (v2)
controlVersion	Activates or deactivates the option to recover file versions
comments	Activates or deactivates the option to share comments between the users of a file
calendar	Activates or deactivates the option to synchronize calendars.

Urls	
REQUEST_TOKEN_URL	Oauth API where the request token is requested.
ACCES_TOKEN_URL	Oauth API to request the Access token
CALLBACK_URL	Redirection to eyeOS once the user authorizes access to their private area
RESOURCE_URL	Oauth API to access the user's protected resources
OAUTH_URL	Oauth API to request user verification
Consumer	
key	Provided by the cloud to identify eyeOS
secret	Provided by the cloud to identify eyeOS

Annex 2. Oauth Manager

getRequestToken(cloud)

Request the request token from a specific cloud for the eyeOS consumer.

Parameters:	cloud – Contains the name of the cloud
Script call:	Example: <pre>{ "config": { "cloud": "Stacksync" } }</pre>
Return:	Token object or null in case of error. Example: <pre>{ "key" : "token1234", "secret": "secret1234" }</pre>

getAccessToken(cloud, token, verifier)

Request the Access token from a specific cloud for the eyeOS consumer from a request token.

Parameters:	<ul style="list-style-type: none"> • cloud – Contains the name of the cloud • token – Contains the request token and user authorization • verifier – Contains the authorization given by the user for the eyeOS consumer
Script call:	<p>Example:</p> <pre>{ "token": { "key": "token1234", "secret": "secret1234" }, "verifier": "userVerified" }</pre>
Return:	<p>Token object or null in case of error.</p> <p>Example:</p> <pre>{ "key": "access1234", "secret": "access1234" }</pre>

Annex 3. Oauth API

The configuration file of the Oauth API is detailed in Annex 1.

getRequestToken(oauth)

Request the request token of the eyeOS consumer.

Url:	Use REQUEST_TOKEN_URL of the configuration file
Method:	GET
Signature:	Plaintext
Parameters:	oauth – OAuthRequest object. Contains the values of the consumer key, consumer secret, and CALLBACK_url of the configuration file.

Return:	<p>Key and secret of the request token or in case of an error it returns an error structure:</p> <ul style="list-style-type: none"> -error: Error number -description: Error description. <p>Example:</p> <pre>{ "oauth_token" : "token1234", "oauth_token_secret" : "secret1234" } { "error" : "401", "description" : "Authorization required" }</pre>
---------	--

getAccessToken(oauth)

Request the Access token of the eyeOS consumer from the saved request token.

Url:	Use ACCESS_TOKEN_URL of the configuration file
Method:	GET
Signature:	Plaintext
Parameters:	<p>oauth – OAuthRequest object. Contains the values of the consumer key and consumer secret of the configuration file. In addition to the request token and verifier received from StackSync.</p>
Return:	<p>Key and secret of the access token or in case of an error it returns an error structure:</p> <ul style="list-style-type: none"> - error: Error number - description: Error description. <p>Example:</p> <pre>{ "oauth_token" : "token1234", "oauth_token_secret" : "secret1234" } { "error" : "401", "description" : "Authorization required" }</pre>

Annex 4. Storage Manager

getMetadata(cloud, token, id, path, user, resourceUrl)

Obtain the metadata of the current element. Generate its structure from files and/or directories in eyeOS.

Parameters:	<ul style="list-style-type: none"> • cloud – Contains the name of the cloud • token – Contains the key and secret of the access token • id – Identifying number of the element in the specific cloud • path – eyeOS route • user – Identifier of the eyeOS user • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "get", "file": false, "id": "1653", "contents": true } }</pre>

Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Example:</p> <pre>{ "status": "NEW", "mimetype": "index/directory", "checknum": 0, "modified_at": "2015-04-07 16:08:52.449" "filename": "Test_B1" "is_root": false, "parent_id": "null", "version": 1, "is_folder": true, "id": 1653, "size": 0, "contents":[{ "status": "NEW", "mimetype": "inode/directory", "checksum": 0, "modified_at": "2015-04-07 16:08:53.397" "filename": "AAA", "is_root": false, "parent_id": 1653, "version": 1, "is_folder": true, "id": 1654, "size":0 }] }</pre> <p>{“error”: 401}</p>
---------	---

getSkel(cloud, token, file, id, metadatas, path, pathAbsolute, pathEyeos, resourceUrl)

Recursively obtain the metadata that depends on the current element. Used in the action of copying and moving in eyeOS.

Parameter s:	<ul style="list-style-type: none"> • cloud – Contains the name of the cloud. • token – Contains the key and secret of the access token. • file – True if it is a file, or False if it is a directory. • id – Identifying number of the element in the specific cloud. • metadatas – Metadata accumulating vector. • path – Path of the current element. • pathAbsolute – eyeOS path • pathEyeos – eyeOS path, only used when the destination of the action is outside the Personal Cloud • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "get", "file": false, "id": 1653, "contents": true } }</pre>

Return:	<p>Metadata vector or in case of an error it returns an error structure.</p> <ul style="list-style-type: none">- error. Error number. <p>Example:</p> <pre>[{ "status": "NEW", "mimetype": "inode/pdf", "checksum": 2230714779, "modified_at": "2015-03-27 16:46:33.243" "filename": "chicken.pdf", "parent_id": 1653, "version": 1, "is_folder": false, "chunks": [], "id": 1654, "size": 51500, "pathAbsolute": "/var/www/eyeos/eyeos/users/a/Cloudspaces/Stacksync/Test_B1/chicken.pdf", "path": "/Test_B1/", "pathEyeos": "home://~/Cloudspaces/Stacksync/Test_B1/chicken.pdf"}, { "status": "NEW", "mimetype": "index/directory", "checknum": 0, "modified_at": "2015-04-07 16:08:52.449" "filename": "Test_B1" "is_root": false, "parent_id": "null", "version": 1, "is_folder": true, "id": 1653, "size": 0, "pathAbsolute": "/var/www/eyeos/eyeos/users/a/Cloudspaces/Stacksync/Test_B1", "path": "/", "pathEyeos": "home://~/Cloudspaces/Stacksync/Test_B1" }]</pre>
---------	--

createMetadata(cloud, token, user, file, name, parent_id, path, pathAbsolute, resourceUrl)

Create a new file or directory.

Parameters:	<ul style="list-style-type: none"> • cloud – Contains the name of the cloud. • token – Contains the key and secret of the access token. • user – Identifier of the eyeOS user. • file – True if it is a file, or False if it is a directory. • name – Name of the element. • parent_id – Id of the destination directory. • path – Path of the current element. • pathAbsolute – Absolute path. Mandatory if the element is a file. • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "create", "file": true, "filename": "File_1.txt", "parent_id": "1653", "path": "/var/www/eyeos/.../Cloudspaces/Stacksync/Test_B1/File_1.txt" } }</pre>
Return:	<p><i>Structure of the result.</i></p> <ul style="list-style-type: none"> - <i>status</i>. 'OK' if correct 'KO' in case of an error. - <i>error</i>: Error number. Only exists in case of an error. <p>Example:</p> <pre>{"status": "OK" } { "status": "KO", "error": 401 }</pre>

downloadMetadata(token, id, path, user, isTmp, cloud, resourceUrl)

Download the content of a file.

Parameters:	<ul style="list-style-type: none"> • token – Contains the key and secret of the access token. • id – Identifying number of the element in the specific cloud. • path – Absolute path. • user – Identifier of the eyeOS user. • isTmp – False, updates the version table of the files. True, no update is carried out. • cloud– Name of the cloud • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "download", "id": 32565632111, "path": "/home/eyeos/Documents/Client.pdf" } }</pre>
Return:	<p><i>Structure of the result.</i></p> <ul style="list-style-type: none"> - <i>status</i>. 'OK' if correct 'KO' in case of an error. - <i>error</i>: Error number. Only exists in case of an error. <p><i>Example:</i></p> <pre>{ "status": "OK" } { "status": "KO", "error": -1 }</pre>

deleteMetadata(cloud, token, file, id, user, path, resourceUrl)

Delete an existing file or directory.

Parameters:	<ul style="list-style-type: none"> • cloud – Contains the name of the cloud • token – Contains the key and secret of the access token. • file – True if it is a file, or False if it is a directory. • id – Identifying number of the element in the specific cloud. • user – Identifier of the eyeOS user. • path – Absolute path. • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "delete", "file": true, "id": 32565632111 } }</pre>
Return:	<p><i>Structure of the result.</i></p> <ul style="list-style-type: none"> - <i>status</i>. 'OK' if correct 'KO' in case of an error. - <i>error</i>: Error number. Only exists in case of an error. <p><i>Example:</i></p> <pre>{ "status": "OK" } { "status": "KO", "error": -1 }</pre>

renameMetadata(cloud, token, file, id, name, path, user, parent, resourceUrl)

Rename a file or directory.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • file – True if it is a file, or False if it is a directory. • id – Identifying number of the element in the specific cloud. • name – New name of the element. • path – Path of the current element. • user – Identifier of the eyeOS user. • parent – Id of the destination directory. (Optional) • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync" , "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "update", "file": true, "id": 32565632156 "filename": "Client2.pdf" "parent_id": 155241412 } }</pre>
Return:	<p><i>Structure of the result.</i></p> <ul style="list-style-type: none"> - <i>status. 'OK' if correct 'KO' in case of an error.</i> - <i>error: Error number. Only exists in case of an error.</i> <p>Example:</p> <pre>{ "status": "OK" } { "status": "KO", "error": -1 }</pre>

moveMetadata(cloud, token, file, id, pathOrig, pathDest, user, parent, filenameOld, filenameNew, resourceUrl)

Move a file or directory.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • file – True if it is a file, or False if it is a directory. • id – Identifying number of the element in the specific cloud. • pathOrig – eyeOS path at origin. • pathDest – eyeOS path at destination. • user – Identifier of the eyeOS user. • parent – Id of the destination directory. • filenameOld – Name of the element in the origin path. • filenameNew – Name of the element in the destination path if different from the origin. (Optional). • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync" , "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "update", "file": true, "id": 32565632156 "filename": "Client2.pdf" "parent_id": "null" } }</pre>

Return:	<p><i>Structure of the result.</i></p> <ul style="list-style-type: none"> - <i>status. 'OK' if correct 'KO' in case of an error.</i> - <i>error: Error number. Only exists in case of an error.</i> <p><i>Example:</i></p> <pre>{ "status": "OK" } { "status": "KO", "error": -1 }</pre>
---------	--

listVersions(cloud, token, id, user, resourceUrl)

Obtain the list of versions of a specific file.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id – Identifying number of the element in the specific cloud. • user – Identifier of the eyeOS user. • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
Script call:	<p><i>Example:</i></p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "listVersions", "id": 32565632156 } }</pre>

Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Example:</p> <pre>{ "status": "CHANGED", "mimetype": "text/plain", "versions": [{ "status": "CHANGED", "mimetype": "text/plain", "checksum": 2499810342, "modified_at": "2014-06-20 10:11:11.031", "filename": "welcome.txt", "parent_id": null, "version": 4, "is_folder": false, "chunks": [], "id": 155, "size": 61 }, { "status": "RENAMED", "mimetype": "text/plain", "checksum": 1825838054, "modified_at": "2014-06-20 10:11:11.031", "filename": "welcome.txt", "parent_id": null, "version": 3, "is_folder": false, "chunks": [], "id": 155, "size": 59 }, { "status": "RENAMED", "mimetype": "text/plain", "checksum": 1825838054, "modified_at": "2014-06-20 10:11:11.031", "filename": "welcome.txt", "parent_id": null, "version": 2, "is_folder": false, "chunks": [], "id": 155, "size": 59 }, { "status": "NEW", "mimetype": "text/plain", "checksum": 1825838054, "modified_at": "2014-06-20 10:11:11.031", "filename": "welcome.txt", "parent_id": null, "version": 1, "is_folder": false, "chunks": [], "id": 155, "size": 59 }], "checksum": 2499810342, "modified_at": "2014-06-20 10:11:11.031", "filename": "welcome.txt", "parent_id": "null", "version": 4, "is_folder": false, "chunks": [], "id": 155, "size": 61 }</pre> <p><i>{“error”: 401}</i></p>
---------	---

getFileVersionData(cloud, token, id, version, path, user, resourceUrl)

Download the content of a specific version of an existing file.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id – Identifying number of the element in the specific cloud. • version – Version pending download. • path – Absolute path. • user – Identifier of the eyeOS user. • resourceUrl: API to access the user’s protected resources from the external cloud (Optional)
-------------	---

Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync" , "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "getFileVersion", "id": 32565632156, "version": 2, "path": "/home/eyeos/welcome.pdf" } }</pre>
Return:	<p><i>Structure of the result.</i></p> <ul style="list-style-type: none"> - <i>status.</i> 'OK' if correct 'KO' in case of an error. - <i>error:</i> Error number. Only exists in case of an error. <p>Example:</p> <pre>{ "status": "OK" } { "status": "KO", "error": -1 }</pre>

getListUsersShare(cloud, token, id, resourceUrl)

Obtain the list of users who share the directory.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id – Identifying number of the directory in the specific cloud. • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
-------------	--

Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "listUsersShare", "id": 32565632156 } }</pre>
Return:	<p>Metadata vector or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Example:</p> <pre>[{"joined_at": "2014-05-27", "is_owner": true, "name": "tester1", "email": "tester1@test.com"}] {"error": 401}</pre>

shareFolder(cloud, token, id, list, shared, resourceUrl)

Share or stop sharing a directory with a list of users.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id – Identifying number of the directory in the specific cloud. • list – List of users. • shared – (true- unshare, false - share) • resourceUrl: API to access the user's protected resources from the external cloud (Optional)
-------------	--

Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync" , "resource_url": "http://ast3-deim.urv.cat/v1/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "shareFolder", "id": 32565632156, "list": ["a@a.com", "b@b.com"], "shared": false } }</pre>
Return:	<p><i>Structure of the result.</i></p> <ul style="list-style-type: none"> - status. 'OK' if correct 'KO' in case of an error. error: Error number. Only exists in case of an error. <p>Example:</p> <pre>{ "status": "OK" } { "status": "KO", "error": -1 }</pre>

insertComment(cloud, token, id, user, text, resourceUrl)

Create a new comment associated to a file shared on the cloud

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id – Identifying number of the file in the specific cloud. • user - eyeOS user. • text – Text of the comment. • resourceUrl: API to access resources that allow for comments to be managed (optional).
-------------	--

Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync" , "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "insertComment", "id": "2401", "user": "tester1", "text": "Test coments" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "id": "2401", "user": "tester1", "text": "Test comments", "cloud": "Stacksync", "status": "NEW", "time_created": "201506101548" } { "status": "KO", "error": 400 }</pre>

deleteComment(cloud, token, id, user, timeCreated, resourceUrl)

Eliminar un comentario asociado a un fichero compartido en el cloud.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • Id – Identifying number of the file in the specific cloud. • user - eyeOS user. • timeCreated – Time and date of creation. • resourceUrl: API to access resources that allow for comments to be managed (optional).
-------------	---

Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "deleteComment", "id": "2401", "user": "tester1", "time_created": "201506101548" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "id": "2401", "user": "tester1", "text": "Test comments", "cloud": "NEC", "status": "DELETED", "time_created": "201506101548" } { "status": "KO", "error": 400 }</pre>

getComments(cloud, token, id, resourceUrl)

Obtain a list of comments associated to a file shared on the cloud

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • Id – Identifying number of the file in the specific cloud. • resourceUrl: API to access resources that allow for comments to be managed (optional).
-------------	---

Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "getComments", "id": "2401" } }</pre>
Return:	<p>Metadata vector or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>[{"id": "2401", "user": "tester1", "text": "Test comments", "cloud": "Stacksync", "status": "NEW", "time_created": "201506101548"}] { "status": "KO", "error": 400 }</pre>

insertEvent(cloud, token, user, calendar, isallday, timestart, timeend, repetition, finaltype, finalvalue, subject, location, description, repeattype, resourceUrl)

Create a new event in the calendar.

Parameters:	<ul style="list-style-type: none">• cloud –Contains the name of the cloud• token – Contains the key and secret of the access token.• user - eyeOS user• calendar – Calendar identifier.• isallday – Specifies if the event takes up the whole day.• timestart - Start date and time of the event.• timeend – End date and time of the event.• repetition – Specifies if the event is repeated on different days.• finaltype – Final type of event.• finalvalue – End date and time of the event if it goes on for several days.• subject - Identifier of the event.• location - Location of the event.• description - Complementary information of the event.• repeattype – How the event is repeated.• resourceUrl: API to access resources that allow for calendars to be managed (optional).
-------------	--

Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "insertEvent", "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "NEW", "type": "event" } { "status": "KO", "error": 400 }</pre>

deleteEvent(cloud, token, user, calendar, timestart, timeend, isallday, resourceUrl)

Delete an event from the calendar.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • user - eyeOS user • calendar – Calendar identifier. • timestart - Start date and time of the event. • timeend – End date and time of the event. • isallday – Specifies if the event takes up the whole day. • resourceUrl: API to access resources that allow for calendars to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "deleteEvent", "user": "eyeos", "calendar": "personal", "timestart": "201419160000", "timeend": "201419170000", "isallday": 0, } }</pre>

Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "DELETED", "type": "event" }</pre> <pre>{ "status": "KO", "error": 400 }</pre>
---------	--

updateEvent(cloud, token, user, calendar, isallday, timestart, timeend, repetition, finaltype, finalvalue, subject, location, description, repeattype, resourceUrl)

Update the event data in the calendar.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • user - eyeOS user • calendar – Calendar identifier. • isallday – Specifies if the event takes up the whole day. • timestart - Start date and time of the event. • timeend – End date and time of the event. • repetition – Specifies if the event is repeated on different days. • finaltype – Final type of event. • finalvalue – End date and time of the event if it goes on for several days. • subject - Identifier of the event. • location - Location of the event. • description - Complementary information of the event. • repeattype – How the event is repeated. • resourceUrl: API to access resources that allow for calendars to be managed (optional).
-------------	--

Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "updateEvent", "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "CHANGED", "type": "event" } { "status": "KO", "error": 400 }</pre>

getEvents(cloud, token, user, calendar, resourceUrl)

Obtain all the events from a calendar.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • user - eyeOS user • calendar – Calendar identifier. • resourceUrl: API to access resources that allow for calendars to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "getEvents", "user": "eyeos", "calendar": "personal" } }</pre>
Return:	<p>Metadata vector or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>[{"user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "NEW", "type": "event"}] { "status": "KO", "error": 400 }</pre>

insertCalendar(cloud, token, user, name, description, timezone, resourceUrl)

Create a new calendar.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • user - eyeOS user • name – Calendar identifier. • description - Complementary information of the calendar. • timezone – Timezone of the calendar. • resourceUrl: API to access resources that allow for calendars to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "insertCalendar", "user": "eyeos", "name": "personal", "description": "detail", "timezone": "0" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "NEW", "type": "calendar" } { "status": "KO", "error": 400 }</pre>

deleteCalendar(cloud, token, user, name, resourceUrl)

Delete a calendar.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • user - eyeOS user • name – Calendar identifier. • resourceUrl: API to access resources that allow for calendars to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "deleteCalendar", "user": "eyeos", "name": "personal" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "DELETED", "type": "calendar" } { "status": "KO", "error": 400 }</pre>

updateCalendar(cloud, token, user, name, description, timezone, resourceUrl)

Update calendar data.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • user - eyeOS user • name – Calendar identifier. • description - Complementary information of the calendar. • timezone – Timezone of the calendar. • resourceUrl: API to access resources that allow for calendars to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "updateCalendar", "user": "eyeos", "name": "personal", "description": "detail", "timezone": "0" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "CHANGED", "type": "calendar" } { "status": "KO", "error": 400 }</pre>

getCalendars(cloud, token, user, resourceUrl)

Obtain a list with all of the user's calendars.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • user - eyeOS user • resourceUrl: API to access resources that allow for calendars to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "getCalendars", "user": "eyeos", } }</pre>
Return:	<p>Metadata vector or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>[{"user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "NEW", "type": "calendar"}] { "status": "KO", "error": 400 }</pre>

getCalendarsAndEvents(cloud, token, user, resourceUrl)

Obtain a list with all of the user's calendars and events.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • user - eyeOS user • resourceUrl: API to access resources that allow for calendars to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "getCalendarsAndEvents", "user": "eyeos", } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>[{"user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "NEW", "type": "calendar"}, {"user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "NEW", "type": "event"}] { "status": "KO", "error": 400 }</pre>

deleteCalendarsUser(cloud, token, user, resourceUrl)

Delete all the calendars and events of the user.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • user - eyeOS user • resourceUrl: API to access resources that allow for calendars to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "deleteCalendarsUser", "user": "eyeos" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "delete": "true" } { "status": "KO", "error": 400 }</pre>

unLockedFile(cloud, token, id, user, ipserver, timeLimit, dt_now, resourceUrl)

Check if the file is blocked by another user.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id - Identifying number of the element in the specific cloud. • user - eyeOS user. • ipserver – IP address of the eyeOS server. • timeLimit – Maximum time in minutes to block a file. • dt_now – Current date and time • resourceUrl: API to access resources that allow for documents to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "getMetadataFile", "id": "2150" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "id": "2150", "cloud": "Stacksync", "user": "eyeos", "ipserver": "192.168.56.101", "datetime": "2015-05-12 11:50:00", "status": "close" } { "status": "KO", "error": 400 }</pre>

lockFile(cloud, token, id, user, ipserver, timeLimit, dt_now, resourceUrl)

Unblock an eyeDocs file.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id - Identifying number of the element in the specific cloud. • user - eyeOS user. • ipserver – IP address of the eyeOS server. • timeLimit – Maximum time in minutes to block a file. • dt_now – Current date and time • resourceUrl: API to access resources that allow for documents to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "lockFile", "id": "2150", "user": "eyeos", "ipserver": "192.168.56.101", "datetime": "2015-05-12 11:50:00", "timelimit": 10 } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "lockFile": true } { "status": "KO", "error": 400 }</pre>

updateDateTime(cloud, token, id, user, ipserver, dt_now, resourceUrl)

Update the metadata with the date and time of the latest change.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id - Identifying number of the element in the specific cloud. • user - eyeOS user. • ipserver – IP address of the eyeOS server. • dt_now – Current date and time • resourceUrl: API to access resources that allow for documents to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync" , "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "updateDateTime", "id": "2150", "user": "eyeos", "ipserver": "192.168.56.101", "datetime": "2015-05-12 11:50:00" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "updateFile": true } { "status": "KO", "error": 400 }</pre>

unLockFile(cloud, token, id, user, ipserver, dt_now, resourceUrl)

Unblock an eyeDocs file.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id - Identifying number of the element in the specific cloud. • user - eyeOS user. • ipserver – IP address of the eyeOS server. • dt_now – Current date and time • resourceUrl: API to access resources that allow for documents to be managed (optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync" , "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "unLockFile", "id": "2150", "user": "eyeos", "ipserver": "192.168.56.101", "datetime": "2015-05-12 11:50:00" } }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "unLockFile": true } { "status": "KO", "error": 400 }</pre>

getMetadataFolder(cloud, token, id, resourceUrl)

Obtain the file structure of a file in the cloud.

Parameters:	<ul style="list-style-type: none"> • cloud –Contains the name of the cloud • token – Contains the key and secret of the access token. • id - Identifying number of the element in the specific cloud. • resourceUrl: API to access the user's protected resources from the user in the cloud (Optional).
Script call:	<p>Example:</p> <pre>{ "config": { "cloud": "Stacksync", "resource_url": "http://192.168.56.101:9000/" }, "token": { "key": "token1234", "secret": "secret1234" }, "metadata": { "type": "get", "file": false, "id": "1653", "contents": true } }</pre>

Return:	<p>Metadata or in case of an error it returns an error structure.</p> <ul style="list-style-type: none"> - error. Error number. <p>Examples:</p> <pre>{ "status": "NEW", "mimetype": "index/directory", "checknum": 0, "modified_at": "2015-04-07 16:08:52.449" "filename": "Test_B1" "is_root": false, "parent_id": "null", "version": 1, "is_folder": true, "id": 1653, "size": 0, "contents": [{ "status": "NEW", "mimetype": "inode/directory", "checksum": 0, "modified_at": "2015-04-07 16:08:53.397" "filename": "AAA", "is_root": false, "parent_id": 1653, "version": 1, "is_folder": true, "id": 2150, "size": 0 }] }</pre> <pre>{ "status": "KO", "error": 400 }</pre>
---------	--

Annex 5. Storage API

The configuration file of the Storage API is detailed in Annex 1.

getMetadata(oauth, file, id, contents)

Obtain the metadata of a directory and/or files.

Url:	Use RESOURCE_URL of the configuration file
------	--

Method:	GET
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none">• oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the accesstoken.• file – True if it is a file, or False if it is a directory.• id - Identifying number of the element (directory or file).• contents – True, list the metadata that depend on the element identified with “Id”, or None, does not activate the list. Used when “Id” is a directory. (Optional)

Return:	<p>Metadata of the element(s) or in case of error returns an error structure:</p> <ul style="list-style-type: none"> - error: Error number - description: Error description. <p>Examples:</p> <pre>{ "filename": "clients", "id": 9873615, "mimetype": "inode/directory", "checknum": 0, "status": "NEW", "version": 1, "parent_id": "null", "modified_at": "2013-03-08 10:36:41.997", "is_root": false, "is_folder": true, "contents": [{ "filename": "Client1.pdf", "id": 32565632156, "size": 775412, "mimetype": "application/pdf", "checksum": 714671479, "status": "NEW", "version": 1, "parent_id": -348534824681, "modified_at": "2013-03-08 10:36:41.997", "is_folder": false, "chunks": [] }] }</pre> <p><code>{"error": 403, "description": "Forbidden"}</code></p>
---------	--

updateMetadata(oauth, file, id, name, parent)

Update the metadata of the element when renaming and moving.

Url:	Use RESOURCE_URL of the configuration file
Method:	PUT

Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • file – True if it is a file, or False if it is a directory. • id - Identifying number of the element (directory or file). • name – Name of the element • parent – Id of the destination directory. (Optional)
Return:	<p>Metadata of the element or in case of an error it returns an error structure:</p> <ul style="list-style-type: none"> - error: Error number - description: Error description. <p>Examples:</p> <pre>{ "status": "RENAMED", "parent_file_version": "", "parent_file_id": "", "checksum": 151519872, "is_folder": false, "modified_at": "Wed Apr 15 09:51:41 CEST 2015", "id": 1705, "size": 6, "mimetype": "text/plain", "filename": "File_A.txt", "parent_id": "null", "version": 2 "chunks": ["711383A59FDA05336FD2C70C8059D1523EB41A"] }</pre> <pre>{"error":403, "description": "Forbidden"}</pre>

createMetadata(oauth, file, name, parent, path)

Create a new element (file or directory).

Url:	Use RESOURCE_URL of the configuration file
Method:	POST

Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none">• oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the accesstoken.• file – True if it is a file, or False if it is a directory.• name – Name of the element• parent – Id of the destination directory. (Optional)• path – Absolute path of the file. (Optional)

Return:	<p>Metadata of the element or in case of an error it returns an error structure:</p> <ul style="list-style-type: none"> - error: Error number - description: Error description. <p>Examples:</p> <ul style="list-style-type: none"> - Metadata new directory <pre> {"status": "NEW", "parent_file_version": 2, "parent_file_id": 1636, "checksum": 0, "is_folder": true, "modified_at": "Wed Apr 15 10:00:54 CEST 2015", "id": 1706, "size": 0, "mimetype": "inode/directory", "filename": "New Folder", "parent_id": 1636, "version": 1 } </pre> - Metadata new file <pre> {"status": "NEW", "parent_file_version": 2, "parent_file_id": 1636, "checksum": 2159423794, "is_folder": false, "modified_at": "Wed Apr 15 10:04:34 CEST 2015", "id": 1706, "size": 0, "mimetype": "application/zip", "filename": "New Document.edoc", "parent_id": 1636, "version": 1, "chunks": ["1C2D8F868958D654484980A347C9E417B"] } </pre> <p><code>{"error":403, "description": "Forbidden"}</code></p>
---------	--

uploadFile(oauth, id, path)

Upload the content of an existing file.

Url:	Use RESOURCE_URL of the configuration file
Method:	PUT
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id – Identifying number of the file. • path – Absolute path of the file. (Optional)
Return:	True, false, or in case of 403 error it returns the error number Examples: true false {"error":403, "description": "Forbidden"}

downloadFile(oauth, id, path)

Download the content of an existing file.

Url:	Use RESOURCE_URL of the configuration file
Method:	GET
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id – Identifying number of the file. • path – Absolute path of the file. (Optional)
Return:	True, false, or in case of 403 error it returns the error number Examples: true false {"error":403, "description": "Forbidden"}

deleteMetadata(oauth, file, id)

Delete an element (file or directory).

Url:	Use RESOURCE_URL of the configuration file
Method:	DELETE
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none">• oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token.• file – True if it is a file, or False if it is a directory.• id - Identifying number of the element (directory or file).

Return:	<p><i>Metadata of the element or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <ul style="list-style-type: none"> - <i>Delete a directory</i> <pre>{ "status": "DELETED", "parent_file_version": 2, "parent_file_id": 1636, "checksum": 0, "is_folder": true, "modified_at": "Wed Apr 15 10:37:51 CEST 2015", "id": 1706, "size": 0, "mimetype": "inode/directory", "filename": "New Folder", "parent_id": 1636, "version": 2 }</pre> <ul style="list-style-type: none"> - <i>Delete a file</i> <pre>{ "status": "DELETED", "parent_file_version": 2, "parent_file_id": 1636, "checksum": 111038472, "is_folder": false, "modified_at": "Wed Apr 15 10:40:46 CEST 2015", "id": 1707, "size": 493, "mimetype": "application/zip", "filename": "New Document.edoc", "parent_id": 1636, "version": 4, "chunks": [] }</pre> <pre>{"error": 403, "description": "Forbidden"}</pre>
---------	---

getFileVersions(oauth, id)

Obtain the list of versions of a specific file.

Url:	Use RESOURCE_URL of the configuration file
Method:	GET
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none">• oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token.• id – Identifying number of the file.

Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "status": "CHANGED", "is_folder": false, "chunks": [], "id": "155", "size": 61, "mimetype": "text/plain", "versions": [{ "status": "CHANGED", "is_folder": false, "chunks": [], "id": "155", "size": 61, "mimetype": "text/plain", "checksum": 2499810342, "modified_at": "2014-06-20 10:11:11.031", "filename": "welcome.txt", "parent_id": "null", "version": 2}, { "status": "RENAMED", "is_folder": false, "chunks": [], "id": "155", "size": 59, "mimetype": "text/plain", "checksum": 1825838054, "modified_at": "2014-06-20 10:11:11.031", "filename": "welcome.txt", "parent_id": "null", "version": 1}], "checksum": 2499810342, "modified_at": "2014-06-20 10:11:11.031", "filename": "welcome.txt", "parent_id": "null", "version": 2 }</pre> <p><i>{ "error": 403, "description": "Forbidden" }</i></p>
---------	--

getFileVersionData(oauth, id, version, path)

Download the content of a specific version of an existing file.

Url:	Use RESOURCE_URL of the configuration file
Method:	GET
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id – Identifying number of the file. • version – Version pending download. • path – Absolute path of the file.
Return:	<p><i>True or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Example:</i></p> <pre><i>{"error":403, "description": "Forbidden"}</i></pre>

getListUsersShare(oauth, id)

Obtain the list of users who share the directory.

Url:	Use RESOURCE_URL of the configuration file
Method:	GET
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id – Identifying number of the directory.

Return:	<p><i>List of users or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Example:</i></p> <pre>[{"joined_at": "2014-05-27", "is_owner": true, "name": "tester1", "email": "tester1@test.com"}] {"error":403, "description": "Forbidden"}</pre>
---------	--

shareFolder(oauth, id, list, isShared)

Share or stop sharing a directory with a list of users.

Url:	Use RESOURCE_URL of the configuration file
Method:	POST
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id – Identifying number of the directory. • list – List of users • isShared – (true - unshare, False - share)
Return:	<p><i>True or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Example:</i></p> <pre>{"error":403, "description": "Forbidden"}</pre>

insertComment(oauth, id, user, text, cloud)

Create a new comment associated to a file shared on the cloud

Url:	RESOURCE_URL of the configuration file.
Method:	POST
Signature:	HMAC-SHA1

Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id – Identifying number of the file • user – Name of user • text –Text of the comment. • cloud – Identifier of the cloud
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "id": "2401", "user": "tester1", "text": "Test comments", "cloud": "Stacksync", "status": "NEW", "time_created": "201506101548" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>

Create new comment in the cloud through the use of the Sync API

Url:	<p>RESOURCE_URL + comments</p> <p>Example:</p> <p>http://demo.eyeos.com/comments</p>
Method:	POST
Body:	<pre>{ "id": "2401", "user": "tester1", "text": "Test comments", "cloud": "Stacksync" }</pre>
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "id": "2401", "user": "tester1", "text": "Test comments", "cloud": "Stacksync", "status": "NEW", "time_created": "201506101548" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>

deleteComment(oauth, id, user, cloud, time_created)

Delete a comment associated to a file shared on the cloud.

Url:	RESOURCE_URL of the configuration file.
Method:	DELETE
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id – Identifying number of the file • user – Name of user • cloud – Identifier of the cloud • time_created – Date and time the comment was created
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "id": "2401", "user": "tester1", "text": "Test comments", "cloud": "Stacksync", "status": "DELETED", "time_created": "201506101548" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>

Delete a comment in the cloud through the use of the Sync API

Url:	RESOURCE_URL + comment/{id}/{user}/{cloud}/{time_created} Example: http://demo.eyeos.com/comment/11/tester1/Stacksync/201506101548
Method:	DELETE
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "id": "2401", "user": "tester1", "text": "Test comments", "cloud": "Stacksync", "status": "DELETED", "time_created": "201506101548" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>

getComments(oauth, id, cloud)

Obtain the list of comments of a file shared on the cloud.

Url:	RESOURCE_URL of the configuration file.
Method:	GET
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id – Identifying number of the file. • cloud – Identifier of the cloud
Return:	<p><i>Metadata vector or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>[{"id": "2401", "user": "tester1", "text": "Test comments", "cloud": "Stacksync", "status": "NEW", "time_created": "201506101548"}] {"error": 404, "description": "Incorrect params"}</pre>

Obtain the list of comments of the file in the cloud through the use of the Sync API

Url:	RESOURCE_URL + comment/{id}/{cloud} Example: http://demo.eyeos.com/comment/2401/Stacksync
	GET
Method:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>[{"id": "2401", "user": "tester1", "text": "Test comments", "cloud": "Stacksync", "status": "NEW", "time_created": "201506101548"}] {"error": 404, "description": "Incorrect params"}</pre>

insertEvent(oauth, user, calendar, cloud, isallday, timestart, timeend, repetition, finaltype, finalvalue, subject, location, description, repeattype)

Create a new event in the calendar.

Url:	RESOURCE_URL of the configuration file.
Method:	POST
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • calendar – Calendar identifier. • cloud – Identifier of the cloud where the calendar is saved. • isallday – Specifies if the event takes up the whole day. • timestart – Start date and time of the event. • timeend – End date and time of the event • repetition – Specifies if the event is repeated on different days. • finaltype – Final type of event. • finalvalue – End date and time of the event if it goes on for several days. • subject - Identifier of the event. • location - Location of the event. • description - Complementary information of the event. • repeattype – How the event is repeated.

Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "NEW", "type": "event" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>
---------	---

Create new event in the cloud calendar through the use of the Sync API

Url:	<p>RESOURCE_URL + event</p> <p>Example:</p> <p>http://demo.eyeos.com/event</p>
Method:	POST
Body:	<pre>{ "user": "eyeos", "calendar": "personal", "cloud": "Stacksync", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n" }</pre>

Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "NEW", "type": "event" }</pre> <p><i>{ "error": 404, "description": "Incorrect params" }</i></p>
---------	--

deleteEvent(oauth, user, calendar, cloud, timestart, timeend, isallday)

Delete an event in the calendar.

Url:	RESOURCE_URL of the configuration file.
Method:	DELETE
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • calendar – Calendar identifier. • cloud – Identifier of the cloud where the calendar is saved. • timestart – Start date and time of the event. • timeend – End date and time of the event • isallday – Specifies if the event takes up the whole day.

Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "DELETED", "type": "event" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>
---------	---

Delete an event in the calendar in the cloud through the use of the Sync API

Url:	<p>RESOURCE_URL + event/{user}/{calendar}/{cloud}/ {timestart}/{timeend}/{isallday}</p> <p>Example: http://demo.eyeos.com/event/eyeos/personal/Stacksync/201419160000/201419170000/0</p>
Method:	DELETE
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "DELETED", "type": "event" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>

updateEvent(oauth, user, calendar, cloud, isallday, timestart, timeend, repetition, finaltype, finalvalue, subject, location, description, repeattype)

Update the event data in the calendar.

Url:	RESOURCE_URL of the configuration file.
Method:	PUT
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • calendar – Calendar identifier. • cloud – Identifier of the cloud where the calendar is saved. • isallday – Specifies if the event takes up the whole day. • timestart – Start date and time of the event. • timeend – End date and time of the event • repetition – Specifies if the event is repeated on different days. • finaltype – Final type of event. • finalvalue – End date and time of the event if it goes on for several days. • subject - Identifier of the event. • location - Location of the event. • description - Complementary information of the event. • repeattype – How the event is repeated.
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "CHANGED", "type": "event" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>

Update an event in the calendar in the cloud through the use of the Sync API

Url:	RESOURCE_URL + event Example: http://demo.eyeos.com/event
Method:	PUT
Body:	<pre>{ "user": "eyeos", "calendar": "personal", "cloud": "Stacksync", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n" }</pre>
Return:	<p>Metadata or in case of an error it returns an error structure:</p> <ul style="list-style-type: none"> - error: Error number - description: Error description. <p>Examples:</p> <pre>{ "user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "CHANGED", "type": "event" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>

getEvents(oauth, user, calendar, cloud)

Obtain a list with all the events in the calendar.

Url:	RESOURCE_URL of the configuration file.
Method:	GET

Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • calendar – Calendar identifier. • cloud – Identifier of the cloud where the calendar is saved.
Return:	<p><i>Metadata vector or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>[{"user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "NEW", "type": "event"}] {"error":404, "description": "Incorrect params"}</pre>

Obtain the list with the events in the calendar in the cloud through the use of the Sync API

Url:	<p>RESOURCE_URL + event/{user}/{calendar}/{cloud}</p> <p>Example:</p> <p>http://demo.eyeos.com/event/eyeos/personal/Stacksync</p>
Method:	GET

Return:	<p><i>Metadata vector or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>[{"user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "NEW", "type": "event"}] {"error": 404, "description": "Incorrect params"}</pre>
---------	---

insertCalendar(oauth, user, name, cloud, description, timezone)

Create a new calendar.

Url:	RESOURCE_URL of the configuration file
Method:	POST
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • name – Calendar identifier. • cloud – Identifier of the cloud where the calendar is saved. • description - Complementary information of the calendar. • timezone - Timezone of the calendar.
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{"user": "eyeos", "name": "personal", "description": "detail", "timezo ne": "0", "cloud": "Stacksync", "status": "NEW", "type": "calendar"} {"error": 404, "description": "Incorrect params"}</pre>

Create a new calendar in the cloud through the use of the Sync API

Url:	RESOURCE_URL + calendar Example: http://demo.eyeos.com/calendar
Method:	POST
Body:	{ "user": "eyeos", "name": "personal", "cloud": "Stacksync", "description": "detail", "timezone": "0" }
Return:	Metadata or in case of an error it returns an error structure: - error: Error number - description: Error description. Examples: {"user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "NEW", "type": "calendar"} {"error": 404, "description": "Incorrect params"}

deleteCalendar(oauth, user, name, cloud)

Delete a calendar.

Url:	RESOURCE_URL of the configuration file
Method:	DELETE
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • name – Calendar identifier. • cloud – Identifier of the cloud where the calendar is saved.

Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "DELETED", "type": "calendar" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>
---------	--

Delete a calendar in the cloud through the use of the Sync API

Url:	<p>RESOURCE_URL + calendar/{user}/{name}/{cloud}</p> <p>Example:</p> <p>http://demo.eyeos.com/calendar/eyeos/personal/Stacksync</p>
Method:	DELETE
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "DELETED", "type": "calendar" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>

updateCalendar(oauth, user, name, cloud, description, timezone)

Update calendar data.

Url:	RESOURCE_URL of the configuration file
Method:	PUT
Signature:	HMAC-SHA1

Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • name – Calendar identifier. • cloud – Identifier of the cloud where the calendar is saved. • description - Complementary information of the calendar. • timezone - Timezone of the calendar.
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{“user”:”eyeos”,”name”:”personal”,”description”:”detail”,”timezo ne”:”0”, “cloud”: ”Stacksync”, “status”: ”CHANGED”, ”type”: ”calendar”} {"error": 404, "description": "Incorrect params"}</pre>

Update a calendar in the cloud through the Sync API

Url:	<p>RESOURCE_URL + calendar</p> <p>Example:</p> <p>http://demo.eyeos.com/calendar</p>
Method:	PUT
Body:	<pre>{ "user": "eyeos", "name": "personal", "cloud": "Stacksync", "description": "detail", "timezone": "0" }</pre>
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{“user”:”eyeos”,”name”:”personal”,”description”:”detail”,”timezone” :”0”, “cloud”: ”Stacksync”, “status”: ”CHANGED”, ”type”: ”calendar”} {"error": 404, "description": "Incorrect params"}</pre>

getCalendars(oauth, user, cloud)

Obtain the list with all the user's calendars

Url:	RESOURCE_URL of the configuration file
Method:	GET
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • cloud – Identifier of the cloud where the calendar is saved.
Return:	<p><i>Metadata vector or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>[{"user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "CHANGED", "type": "calendar"}] {"error": 404, "description": "Incorrect params"}</pre>

Obtain the list with all the user's calendars

Url:	<p>RESOURCE_URL + calendar/{user}/{cloud}</p> <p>Example:</p> <p>http://demo.eyeos.com/calendar/eyeos/Stacksync</p>
Method:	GET
Return:	<p><i>Metadata vector or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>[{"user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "CHANGED", "type": "calendar"}] {"error": 404, "description": "Incorrect params"}</pre>

getCalendarsAndEvents(oauth, user, cloud)

Obtain a list with all the user's calendars and events.

Url:	RESOURCE_URL of the configuration file.
Method:	GET
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • cloud – Identifier of the cloud where the calendar is saved.
Return:	<p><i>Metadata vector or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>[{"user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "NEW", "type": "calendar"}, {"user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "NEW", "type": "event"}] {"error": 404, "description": "Incorrect params"}</pre>

Obtain a list of all the calendars and events in the cloud through the use of the Sync API

Url:	RESOURCE_URL + calEvents/{user}/{cloud} Example: http://demo.eyeos.com/calEvents/eyeos/Stacksync
Method:	GET

Return:	<p><i>Metadata vector or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>[{"user": "eyeos", "name": "personal", "description": "detail", "timezone": "0", "cloud": "Stacksync", "status": "NEW", "type": "calendar"}, {"user": "eyeos", "calendar": "personal", "isallday": 0, "timestart": "201419160000", "timeend": "201419170000", "repetition": "null", "finaltype": "1", "finalvalue": "0", "subject": "test", "location": "Barcelona", "description": "detail", "repeattype": "n", "cloud": "Stacksync", "status": "NEW", "type": "event"}] {"error": 404, "description": "Incorrect params"}</pre>
---------	---

deleteCalendarsUser(oauth, user, cloud)

Delete all the calendars and events of the user.

Url:	RESOURCE_URL of the configuration file
Method:	DELETE
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • user - eyeOS user. • cloud – Identifier of the cloud where the calendar is saved.
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{"delete": "true"} {"error": 404, "description": "Incorrect params"}</pre>

Delete all the calendars and events in the cloud with the Sync API

Url:	RESOURCE_URL + calUser/{user}/{cloud} Example: http://demo.eyeos.com/calUser/eyeos/Stacksync
Method:	DELETE
Return:	<i>Metadata or in case of an error it returns an error structure:</i> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <i>Examples:</i> {"delete": "true"} {"error": 404, "description": "Incorrect params"}

lockFile(oauth, id, cloud, user, ipserver, datetime, timelimit)

Unblock an eyeDocs file.

Url:	RESOURCE_URL of the configuration file
Method:	POST
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id - Identifying number of the element in the specific cloud. • cloud – Identifier of the cloud where the file is saved. • user - eyeOS user. • ipserver – IP address of the eyeOS server. • timelimit – Maximum time in minutes to block a file • datetime – Current date and time
Return:	<i>Metadata or in case of an error it returns an error structure:</i> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <i>Examples:</i> {"lockFile": "true"} {"error": 404, "description": "Incorrect params"}

Block the eyeDocs file in the cloud through the use of the Sync API

Url:	RESOURCE_URL + lockFile Example: http://demo.eyeos.com/lockFile
Method:	POST
Body:	{ "id": "2401", "cloud": "Stacksync", "user": "eyeos", "ipserver": "demo.eyeos.com", "datetime": "201419170000", "timelimit": 10 }
Return:	Metadata or in case of an error it returns an error structure: - error: Error number - description: Error description. Examples: { "lockFile": "true" } { "error": 404, "description": "Incorrect params" }

updateDateTime(oauth, id, cloud, user, ipserver, datetime)

Update the metadata with the date and time of the latest change.

Url:	RESOURCE_URL of the configuration file
Method:	PUT
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id - Identifying number of the element in the specific cloud. • cloud – Identifier of the cloud where the file is saved. • user - eyeOS user. • ipserver – IP address of the eyeOS server. • datetime – Current date and time

Return:	<i>Metadata or in case of an error it returns an error structure:</i> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <i>Examples:</i> <pre>{“updateFile”: “true”} {"error":404, "description": "Incorrect params"}</pre>
---------	---

Update date and time of last change in the eyeDocs file in the cloud with the Sync API

Url:	RESOURCE_URL + updateTime Example: http://demo.eyeos.com/updateTime
Method:	PUT
Body:	<pre>{ "id": "2401", "cloud": "Stacksync", "user": "eyeos", "ipserver": "demo.eyeos.com", "datetime": "201419170000" }</pre>
Return:	<i>Metadata or in case of an error it returns an error structure:</i> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <i>Examples:</i> <pre>{“updateFile”: “true”} {"error": 404, "description": "Incorrect params"}</pre>

unLockFile(oauth, id, cloud, user, ipserver, datetime)

Unblock an eyeDocs file.

Url:	RESOURCE_URL of the configuration file.
Method:	PUT
Signature:	HMAC-SHA1

Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id - Identifying number of the element in the specific cloud. • cloud – Identifier of the cloud where the file is saved. • user - eyeOS user. • ipserver – IP address of the eyeOS server. • datetime – Current date and time
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{“unlockFile”: true} {"error": 404, "description": "Incorrect params"}</pre>

Unblock the eyeDocs file in the cloud through the use of the Sync API

Url:	<p>RESOURCE_URL + unlockFile</p> <p>Example:</p> <p>http://demo.eyeos.com/unlockFile</p>
Method:	PUT
Body:	<pre>{ "id": "2401", "cloud": "Stacksync", "user": "eyeos", "ipserver": "demo.eyeos.com", "datetime": "201419170000" }</pre>
Retorno:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{“unlockFile”: true} {"error":404, "description": "Incorrect params"}</pre>

getMetadataFile(oauth, id, cloud)

Obtain the metadata of the file.

Url:	RESOURCE_URL of the configuration file.
Method:	GET
Signature:	HMAC-SHA1
Parameters:	<ul style="list-style-type: none"> • oauth – OAuthRequest object. Contains the values of the consumer key and secret of the configuration file. In addition to the access token. • id - Identifying number of the element in the specific cloud. • cloud – Identifier of the cloud where the file is saved.
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "id": "2150", "user": "eyeos", "cloud": "Stacksync", "ipserver": "192.168.56.101", "datetime": "2012-05-12 11:50:00", "status": "open" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>

Obtain the metadata of the eyeDocs file in the cloud with the Sync API

Url:	RESOURCE_URL + lockFile/{id}/{cloud} Example: http://demo.eyeos.com/lockFile/2401/Stacksync
Method:	GET
Return:	<p><i>Metadata or in case of an error it returns an error structure:</i></p> <ul style="list-style-type: none"> - <i>error: Error number</i> - <i>description: Error description.</i> <p><i>Examples:</i></p> <pre>{ "id": "2150", "user": "eyeos", "cloud": "Stacksync", "ipserver": "192.168.56.101", "datetime": "2012-05-12 11:50:00", "status": "open" }</pre> <pre>{ "error": 404, "description": "Incorrect params" }</pre>