Developing an API wrapper with usability in mind

by

Hampus Malmkvist

hamma881@student.liu.se

LIU-IDA/LITH-EX-A—13/015--SE
Developing an API wrapper with usability in mind

av

Hampus Malmkvist

hamma881@student.liu.se

LIU-IDA/LITH-EX-A—13/015--SE

Handledare: Anders Fröberg
Examinator: Erik Berglund
Abstract
When you are creating new software today, more often than not you will use API’s. Users choosing between different API’s would probably look at functionality but also accessibility and ease of use. It would therefore seem motivated for API providers to create usable and accessible API’s. This thesis is about the things that make an API more usable, and then those things are used in practice to increase the usability of CloudMe’s API. CloudMe’s core API is documented to increase usability and a wrapper for the API is developed to allow people more ways to use CloudMe’s service. The API wrapper is created in Java and was developed thoroughly with regards to usability.
# Table of Contents

Abstract .................................................................................................................. 3

1 Introduction ............................................................................................................. 6
   1.1 Background ........................................................................................................ 6
   1.2 Purpose ............................................................................................................... 6
   1.3 Limitations ........................................................................................................ 6
   1.4 Method ............................................................................................................. 6

2 Preliminary Investigation ....................................................................................... 7
   2.1 API and usability ............................................................................................. 7
   2.2 Concrete practices that could make an API better .............................................. 7
      2.2.1 General guidelines .................................................................................. 7
      2.2.2 Class design ............................................................................................. 8
      2.2.3 Method design Use the same parameter order in methods that use the same or similar parameters [2] ................................................................. 8
      2.2.4 Exception design ...................................................................................... 9
   2.3 Discussion of practices ...................................................................................... 9

3 The Actual Case .................................................................................................... 11
   3.1 About the API the wrapper is created for ......................................................... 11
   3.2 Why is a wrapper needed ................................................................................ 11
   3.3 Preparations before implementation .................................................................. 14
      3.3.1 SOAP [6] .................................................................................................. 14
      3.3.2 REST ....................................................................................................... 14
      3.3.3 CloudMe’s API ......................................................................................... 14

3.4 Implementation ................................................................................................... 15
   3.4.1 General ........................................................................................................ 15
   3.4.2 CloudmeMessenger, send SOAP and REST messages with JAVA .......... 15
   3.4.3 CloudmeXmlHandler .................................................................................. 16
   3.4.4 CloudmeUser .............................................................................................. 16
   3.4.5 CloudmeDriveListener ................................................................................ 16
   3.4.6 CloudmeException ...................................................................................... 16
   3.4.7 CloudmeFolderManager .............................................................................. 16
   3.4.8 CloudmeFolder ............................................................................................ 17
   3.4.9 CloudmeFolderListener .............................................................................. 17
   3.4.10 CloudmeFileManager ................................................................................. 17
   3.4.11 CloudmeFile .............................................................................................. 17

4
1 Introduction
This paper is about the development of an API wrapper and factors that could contribute to the wrappers usability.

1.1 Background
CloudMe is a cloud storage service available for many different clients; a virtual desktop, iPhone app, Android app, Easy Upload that installs on Windows, Mac and Linux. All of these clients use a public API that consists of over 100 state-less SOAP and REST messages. Making this API public would enable third party developers to use the service in applications. Furthermore a wrapper for the API would also enable less knowledgeable developers to use the API.

1.2 Purpose
The purpose of this thesis was to identify the most important methods in CloudMe’s API and document them in a wiki, and then creating a wrapper for the API in Java. The wrapper would allow people with less knowledge of SOAP and REST to use the functionality of CloudMe. The wrapper would also be documented. With good documentation, parts of the core API and the wrapper could then be made public for third party developers.

1.3 Limitations
CloudMe’s API is far too extensive for a full coverage, therefore only the methods I found the most important were documented. The same goes for the wrapper, which doesn’t cover all methods present in the core API.

1.4 Method
The development of the API wrapper has been an iterative process. New functionality and refactoring of existing code was a regular occurrence to meet the expectations of the client. When looking at factors that could impact the usability, several different sources was used to give a more honest view.
2 Preliminary Investigation

The reason for developing an API wrapper is often to increase usability and boost user’s efficiency. It therefore seems natural to find what defines an API’s usability and how you can measure it.

2.1 API and usability

Usability is defined by the following attributes according to Jakob Nielsen [1].

Learnability
How easy is it to use the API without any previous experience? Can you use the API without looking in the documentation, if not how often must you read the documentation?

Efficiency
How efficient can you finish considered tasks with the API.

Memorability
How easy is it to remember core functionality within the API when you haven’t used the API for some time. Do you have to check the documentation again?

Errors
How easy it is for users to make mistakes, how the API handles error messaging and how severe errors users can cause.

Satisfaction
Subjective level of pleasure felt when using the API.

This is however general usability attributes so it might be interesting to look for attributes adjusted for API’s. According to Joshua Bloch [2] a good API has the following attributes.

Easy to learn
Easy to use, even without documentation
Hard to misuse
Easy to read and maintain code that uses the API
Powerful enough to satisfy requirements
Easy to expand
Adjusted for its target audience

It’s hard to believe that a good API wouldn’t be usable. So the question is how one can achieve all these attributes when you develop an API.

2.2 Concrete practices that could make an API better

2.2.1 General guidelines

Do not show more than necessary
Make all classes and methods as private as possible. Decreases the risk for misuse and also makes the API more convenient [3] [2].

Do not do more than necessary
It is better to add more functionality later instead of removing. people using the API might have to recode if you remove some functionality at a later stage. If you are uncertain whether a function is needed or not, then leave it out, at least until you are more certain [3][2].
Be careful with using other API’s in your API
Using other API’s in your code makes it dependent on other API’s. Sometimes it may be motivated as it can save a lot of time, or if your API is a wrapper for another API [3].

Use Modular architecture with low coupling between modules
Makes testing easier and also simplifies a potential expansion of the API [3].

Use well thought out names for classes and methods. Follow a naming standard.
With good names the need to read the documentation could decrease. By following a naming standard the code will be more structured and easier to read [2]

Documentation is important
It will facilitate the use of the API if the users easily can read and understand how methods and classes are supposed to be used. With good documentation it is easier to reuse code for other projects.

2.2.2 Class design
Factories is preferred over pure constructors
Factories give more control than constructors, especially when it comes to error handling. Factories handles the allocation of dynamic memory, and in case of some error can reallocate it as well, which the user otherwise might had to do manually [3].

Methods is preferred over public variables
Public variables can only change a value while methods(getters, setters) are more powerful and can stop the variable to be set to certain un allowed values for example [3]. It’s also more flexible if the internal structure of the API changes, the user does not have to change anything in her code if the method has the same in and out parameters, even if variables and flow changes.

[JAVA][C++11] Make all classes final
By making all classes final (virtual methods for C++) you prevent users to create subclasses, and using the API in an unintended way [3]. For example it stops access to protected variables and methods.

Put public declarations above protected and private in the code
It makes it easier for the user to find what is relevant. You may also consider putting methods that you think will be used more often above the others [4].

Use Create-Set-Call instead of forcing arguments in the constructor [9]
According to the referred link, programmers generally prefer this style. Some of the reasons were more control and freedom (these are not opposites in programming).

2.2.3 Method design
Use the same parameter order in methods that use the same or similar parameters [2]
To use a consistent parameter order should facilitate for the user and lower the rate of errors.

Do not use too many in parameters to methods
Many input parameters increase the risk for the user to make mistakes when using the method. More than three parameters could be regarded as too many [2].

Avoid using the same type of parameters in a row
It is easier to mix up the parameter order when calling a method if they accept the same type of data [2].
**Do not use default values on in parameters**
Default values on in parameters could make the method dependent on some default value unknown for
the user. It is preferred to use several methods instead; the name of the methods can then better reflect
the functionality [4].

**Use custom data types or ENUMS instead of base types if all values of the base type are not allowed**
This prevents the user from giving incorrect values [4].

**Do not make the user do something that you could automate in your methods**
Gives the user a cleaner structure with less boilerplate code needed [2].

2.2.4 Exception design
**Notify the user about errors as soon as possible**
It is easier to debug a smaller area in the code [4].

[Java] **Checked or unchecked exceptions**
There seems to be a disagreement on what to prefer. Oracles official policy [5] states that checked
exceptions should be used if the user has a possibility to recover from it. Joshua Bloch et.al [2]
believes that unchecked is preferred as you do not need boilerplate code containing a lot of try-catch.
Both points seem valid.

2.3 Discussion of practices
All of the practices are recommended for API development by different sources. However, some of
them are discussable as they come with potential disadvantages as well.

**Do not show more than necessary**
This is supposed to prevent misuse, but sometimes unintended use of a product can be a good thing,
people could use it for things you did not even consider. A few examples of unintended use can be
seen in “Accidents and Unintended Uses – Top 10” by Jonny Williamson [13]. A more public API
could also make it easier for some people to understand the structure and flow of the API.

**Do not do more than necessary**
If your API is missing a potential feature that some user find important, said user might implement it
themselves. If you add the same or similar functionality to the API at a later stage it could lead to
complications for the user.

[JAVA][C++11] **Make all classes final**
To prevent unintended use, but sometimes unintended use can be a good thing [13]. By not making all
classes final it would also be possible for people to create subclasses to extend the functionality of the
API.

**Use Create-Set-Call instead of forcing arguments in the constructor**
Sometimes the constructor needs the arguments to be useful. Not forcing arguments in the constructor
can lead to extra code in the methods to check if internal parameters have been set or not.

**Do not use default values on in parameters**
No default parameters will probably increase the boilerplate code as you would need many similar
methods.
**Do not make the user do something that you could automate in your methods**
If you automate too much the API could become less powerful and less customizable.
3 The Actual Case

3.1 About the API the wrapper is created for

The API that is discussed in this report is CloudMe’s API (www.cloudme.com). CloudMe is a cloud storage service similar to Dropbox. The API consists of SOAP and REST messages, over one hundred different. There are all the functions one expects, such as download and upload files, change folder structure et cetera, but there is also less obvious functionality. An example is that the files stored on the server can have several so called streams; a stream is like a file within a file. An image have thumbnails in various resolutions, these are stored as streams.

3.2 Why is a wrapper needed

Without a wrapper there is an extensive amount of coding needed for even the simplest of operations, which I will demonstrate by showing some code that enables you to login through the API. The login method in CloudMe’s API looks like this.

```
login: [] (SOAP)
```

Used to login to the system and returns loads of information, most importantly the user id, homefolder id and session token.

Look simple enough. The following is java code to login, and then there is also a third party library used.

```java
final private static String host = "os.cloudme.com";
final private static String USER_AGENT_INFO = "Nothing special";
final private static String SOAP_HEADER = "<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/" SOAP-ENV:encodingStyle="""
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema"><SOAP-ENV:Body>";
final private static String SOAP_FOOTER = "</SOAP-ENV:Body></SOAP-ENV:Envelope>";

private static AuthScheme scheme = null;
private static Credentials creds = null;

private String username = "username";
private String password = "password";

public static void main(String args[]) {
    String action = "login";
    String body = "";
    String something = callServerStringResponse{action, body};
}

String callServerStringResponse(String soapAction, String body) {
    DefaultHttpClient httpClient = getConnection();
    byte[] b = null;
}
HttpPost httpPost = new HttpPost("http://" + host + "\/v1/");
httpPost.setHeader("soapaction", soapAction);
httpPost.setHeader("Content-Type", "text/xml; charset=utf-8");

final StringBuffer soap = new StringBuffer();
soap.append(SOAP_HEADER);
soap.append("<" + soapAction + ">");
soap.append(body);
soap.append("</" + soapAction + ">");
soap.append(SOAP_FOOTER);

System.out.println("\n\n\n" + soap.toString() + "\n\n"); // Prints the call so that the SOAP-call can be checked manually for errors

try {
    HttpEntity entity = new StringEntity(soap.toString());
    httpPost.setEntity(entity);
    HttpResponse response = httpClient.execute(httpPost);
    // Call to getResponse needs to be done before httpClient is shut down
    // otherwise the response disappears and null is returned
    String stringResponse = getResponse(response);
    httpClient.getConnectionManager().shutdown();
    return stringResponse;
} catch (Exception e) {
    System.out.println(e.getMessage());
}
httpClient.getConnectionManager().shutdown();
return null;

String getResponse(HttpResponse response) {
    try {
        DataInputStream stream = new DataInputStream(response.getEntity().getContent());
        StringBuffer buf = new StringBuffer();
        String tmp;
        while ((tmp = stream.readLine()) != null) {
            buf.append(tmp);
        }
        stream.close();
        return buf.toString();
    } catch (IOException e) {
        return "IOException: " + e.getMessage();
    }
}

DefaultHttpClient getConnection() {
    DefaultHttpClient httpClient = new DefaultHttpClient();
    HttpProtocolParams.setUseExpectContinue(httpClient.getParams(), false);
    HttpProtocolParams.setUserAgent(httpClient.getParams(), USER_AGENT_INFO);
    httpClient.getCredentialsProvider().setCredentials(new AuthScope(host, 80, "os@xcerion.com", "Digest"),
        new UsernamePasswordCredentials(username, password)); // Encrypts password & username

    HttpRequestInterceptor preemptiveAuth = new HttpRequestInterceptor() {
        public void process(final HttpRequest request, final HttpContext context) throws HttpException, IOException {
            AuthState authState = (AuthState) context.getAttribute(ClientContext.TARGET_AUTH_STATE);
            if (authState.getAuthScheme() == null) {
                if (creds != null && scheme != null) {
                    authState.setAuthScheme(scheme);
                    authState.setCredentials(creds);
                } else {
                    scheme = authState.getAuthScheme();
                }
            }
        }
    };

}
httpClient.addRequestInterceptor(preemptiveAuth, 0);

    return httpClient;
}

This is only to send, you also have to handle the return message you get back when you have sent the login request. That message can look like this.

```xml
  <SOAP-ENV:Body>
    <xcr:loginResponse xmlns:xcr="http://xcerion.com/xcerion.xsd">
      <userid>12885514210</userid>
      <username>hakelo</username>
      <firstname>john</firstname>
      <lastname>johson</lastname>
      <locale>en_US</locale>
      <lastlogin>2012-10-18T06:42Z</lastlogin>
      <logins>22</logins>
      <home>562958546675878</home>
      <library>562958546675879</library>
      <out>562958546675881</out>
      <contacts>171823088671</contacts>
      <settings/>
      <params/>
      <session>660d319261fe66912519b36794e7be</session>
      <mycloudme>http://my.cloudme.com/</mycloudme>
      <eula/>
      <drives>
        <drive>
          <driveId>30067187875</driveId>
          <ownerId>12885514210</ownerId>
          <folderId>562958546675878</folderId>
          <sysname>xios</sysname>
          <name></name>
          <system>home</system>
          <currentSize>2173132</currentSize>
          <quotaLimit>3221225472</quotaLimit>
          <maxFileSize>15726400</maxFileSize>
          <readLocked>false</readLocked>
          <writeLocked>false</writeLocked>
          <created>2012-09-18T15:52:11Z</created>
        </drive>
        <drives>
          <group id='17179869184' name='xios'/>
        </drives>
      </drive>
    </xcr:loginResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

If every user that is going to use CloudMe’s API need to write similar code to even begin using it then the efficiency and usability should increase if there is a API wrapper where the same thing is done by typing this.

```java
CloudmeUser user = new CloudmeUser("username", "password");
```
3.3 Preparations before implementation

Before starting on this project I had never used CloudMe’s API. So naturally it felt advantageous to get some idea of how the API works when I’m going to make a wrapper for it. As previously mentioned the API consists of SOAP and REST messages, both of which I never had used before, so the first thing I did was to take a closer look.

3.3.1 SOAP [6]

SOAP was originally an abbreviation of “Simple Object Access Protocol” but apparently that is no longer valid. SOAP is a XML based protocol that defines the structure of messages. SOAP can be used both one-way and like RPC (Remote Procedure Call) where you get an answer back. CloudMe uses RPC. Sending SOAP is not bound to any particular transfer protocol but the most common is HTTP which also CloudMe uses. SOAP can be used with all operating systems and programming languages as it’s basically clean xml which makes it flexible.

Example of SOAP in CloudMe:

```
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <renameDocument>
      <folder id='562958546675878'/>
      <document>Sunset.jpg</document> <document id='36546465654'/>
      <newName>RED.jpg</newName>
    </renameDocument>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3.3.2 REST

For an architecture to be considered RESTful there are several requirements, but that is outside the scope of this report. REST is an acronym for Representational State Transfer. REST is not a protocol in itself but can rather be seen as concept for accessing resources through URLs with the help of http (REST is not necessarily bound to http, but it is the most common and also what CloudMe uses). For each URL there might be several operations possible, the most common are POST, GET, PUT and DELETE.

An example of REST in CloudMe (uploads the file attached to the message to the folder with id 8464645646):

```
POST http://os.cloudme.com/v1/documents/8464645646 HTTP/1.1
```

3.3.3 CloudMe’s API

CloudMe’s API consists of over a hundred stateless SOAP / REST methods. To create a wrapper for an API it is necessary to have knowledge of the original API. To acquire this knowledge I documented the methods I found important in a wiki. All methods documented were tested by sending requests to the server and evaluating the answer and changes on the server. The thing I noticed when testing was that the login method is essential, as it gives the users root folder. Most operations require folder or/and file ids, and to retrieve these you must have one folder to start from, the root folder.

In total around thirty methods were documented, the documentation is provided in an appendix. All
thirty methods were not used in the wrapper, and a few methods not documented were used. An example of this is the method getFolderXml, it was documented but in the wrapper an undocumented REST method was used to retrieve folder information instead. The reason for this design choice can be found in the implementation chapter.

3.4 Implementation

3.4.1 General
The wrapper is written in JAVA with the help of the IDE Eclipse (Indigo). Under the development the things mentioned in section 2 have been considered and followed. All classes lie in a packet called CloudMe, this make it easy to limit access for users of the wrapper as you can use the default access modifier. A central part of the wrapper is the CloudmeUser Class (this is the only class with a public constructor) which is used to log in. Through this class you can access classes for handling folder and file operations. You can create several objects of this class at the same time, enabling you to use the wrapper for several users at the same time. Apart from the standard JAVA functionality the only third party library used is Apache HTTP client.

Fig1. Class diagram of the API wrapper

3.4.2 CloudmeMessenger, send SOAP and REST messages with JAVA
The first thing that’s needed is a way to communicate with the server API; otherwise it would be hard to test the functionality of the wrapper. There was already code available that could send SOAP and some REST messages. There were some limitations though, it was not possible to attach files to the
message and downloaded files were corrupt. Based on the code available I created a class, CloudmeMessenger. At the start it could just send simple messages so attaching files was not possible. It was impossible to send thread safe messages at this point. These things were implemented at a later stage.

3.4.3 CloudmeXmlHandler
When you send a message to the server API, you often get a reply. The reply is often in XML format. To retrieve relevant information from these XML strings I created the class CloudmeXmlHandler. It only contains static methods with default access, which only lets classes in the Cloudme package use them.

3.4.4 CloudmeUser
The CloudmeUser class handles the information you get when you log in through the SOAP login to the server API. This is the only class that can be created directly through its constructor. The reason for relinquishing the use of factory in this case is that there is only one type of user and therefore an extra factory class is somewhat excessive.

The CloudmeUser class is the backbone of the wrapper as it contains classes that handle folder and file operations. It is also responsible for keeping the local folder structure intact as the folder structure on the server could be changed by different agents. To enable listening on changes on the server there is a class called CloudmeDriveLister that listens for changes on the server and then delegates them forward to CloudmeUser for handling. Every CloudmeUser object is therefore threaded to be able to handle all these changes it gets notified of. A user can create file and folder listeners, CloudmeUser also notifies these potential listeners of changes on the server.

CloudmeUser also handles so called “key values”. It is similar to a simple database where you can get and set data associated to given keys.

3.4.5 CloudmeDriveListener
This class is responsible for listening on changes for the users account on the server. As it constantly should listen for changes objects of this class needs to be threaded. To enable listening on the server this class sends a SOAP message called “createPorts” to the users virtual main drive. In response you get an URL that you can listen on with REST (http get). As objects of this class are threaded and also use CloudmeMessenger to listen on the URL, I had to add thread safe methods to CloudmeMessenger otherwise the thread could stay alive for an unknown amount of time.

3.4.6 CloudmeException
If something should go wrong within the CloudMe package CloudmeException will be thrown. It is a subclass of exception and therefore it is a checked exception. There are different views whether checked or unchecked is to prefer, see section 2.2.4.

3.4.7 CloudmeFolderManager
This class is responsible for folder operations. The most common folder operations are available, such as creating, remove, move, retrieve and change name on folders. Access to this class occurs through CloudmeUser. The root node of the local folder structure is stored in this class. This tree grows dynamically as needed. A reason for having a local image of the folder structure is that you can iterate through it faster as you don’t have to ask the server about folders you searched previously.

CloudmeFolderManager saves all folders a user opens with “weak-references” to enable messaging to eventual folder listeners.
3.4.8 CloudmeFolderNode
To create the local folder tree this class is used. It is a normal tree structure where subfolders are the current nodes children. The tree grows dynamically as it otherwise would take a considerable amount of time to create the whole structure for larger folder structures. Initially I used a method called getFolderXml to retrieve information about folders. It led to some problems as it returns the whole folder structure at once. The folder structure can be close to infinite which would lead to a very slow wrapper. So instead of using getFolderXml a REST method that only gives all first level subfolders of a given folder was used. Using the REST method also carry another advantage, Cloudme uses xlinks that can contain folder information that you have to check if you use getFolderXml, something that’s not necessary when using the REST method as it always return all folder directly. CloudmeFolderNode has methods for retrieving folders from strings, for example “/myfolder/myfolder2/” would return the folder object for myfolder2.

3.4.8 CloudmeFolder
CloudmeFolder is a class that represents a folder. Objects of this class are returned from operations on CloudmeFolderManager, CloudmeFolderNode or CloudmeFolder. CloudmeFolder basically has the same methods as CloudmeFolderManager, but without the need to specify which folder in the arguments. There are also some exclusive methods, you can get the parent and children of this CloudmeFolder object. It is possible for users to add folder listeners to this class. Folder listeners are specified by a certain interface (CloudmeFolderListener).

3.4.9 CloudmeFolderListener
An interface the user can implement to create folder listeners.

3.4.10 CloudmeFileManager
CloudmeFileManager has methods for file operations. These methods include operations such as downloading, uploading, search, remove and copy files. Objects of this class have a list of “weak-references” for all the files allocated by the user, enabling notification to eventual file listeners.

3.4.11 CloudmeFile
Objects of this class represent files on the server. CloudmeFile has the same methods as CloudmeFileManager but you do not have to provide a path to the file. It is possible to store files in the local memory, perhaps modify something and then write back to the server. You can also download files directly to the hard drive. CloudmeFile also handles streams, which basically can be seen as other files stored in a file. A class handling a files metadata (describing information) is also stored in CloudmeFile, it’s called CloudmeMetadata.

3.4.12 CloudmeStreamData
CloudmeStreamData stores the data and identifying number of a stream. An example of a stream in CloudMe is that all the images you upload automatically gets thumbnails in various sizes, these thumbnails are stored as streams to the file. It is also possible to add arbitrary data as a stream to a file.

3.4.13 CloudmeMetadata
CloudmeMetadata stores a files metadata and provides methods for access and modification. Metadata has several important attributes, for example the name of the file, its id and the file type. It is possible to add custom metadata, as long as you follow the standard used by the default metadata.
3.4.14 Cloudme
Cloudme is a class that only contains a few static methods for the other classes. Not publicly available for users of the wrapper.

3.4.15 Problems encountered during implementation
The server sends most of the messages in text format, so Java's string type was used to receive them. This is a problem when you try to download files that often are in binary format. The files became corrupt as string has a 16 bit representation and therefore binary data represented by simple bytes are damaged. It took a while to discover this as normal text files was possible to download without corruption.

Initially a single thread was used to listen for changes and handle them. However, it turned out that if several changes were reported one after another, there was a chance for an event to be missed with the thread busy processing a previous event. This was solved by introducing an extra thread whose sole purpose was to listen and add all messages it receive in a queue. Another thread then iterates through the queue and handles all messages.

When you move a folder on the server through an API call, it is not necessarily ready to be used again right away. In order to manage this, a listener is needed for the local folder object. The listener has to apply the changes to the local object. The listener needs to be implemented by the user, but there is an interface to follow provided. However, the local folder structure is kept intact automatically.

For file and folder listeners to be notified of changes you have to keep track of the files and folders in some way. It is easy to just add a list of all files and folders that the user has opened, but problems can arise if the user deletes the file or folder reference. The list will still have a reference to the object so garbage collection will not be invoked. By making the lists keeping track of opened files and folders to weak references the problem is solved. Weak references are not strong enough to prevent garbage collection on their own, so when the user no longer references the object, it is removed.

3.5 Documentation
As noted before I documented the most important methods of CloudMe’s API, to get knowledge about the system. Another point of documenting them was that CloudMe wanted to make parts of their API public, and my documentation was supposed to serve as a base for that documentation. So I added a structure to the methods I documented and a tutorial for how you could get started with the API. The API wrapper would also be made public, and therefore needed good documentation as well. All public methods and classes were documented and a few tutorials on how to use the API were written. The documentation for both the core API and the Java wrapper can be found in the appendix.
4 Evaluation

There is information on what makes an API usable, but not so much on how you evaluate a finished API [8]. You could use some form of user testing, but it is quite expensive, especially since programming knowledge is needed to use an API. The wrapper will allow users who know Java to use some of the functionality offered by the core API. Some of these users would probably not have been able to use the core API in reasonable time, and therefore the wrapper effectively increases the user base. One could argue that the usability of a system increases if it is possible for more users to use it in a meaningful way, which could mean that the wrapper would increase the overall usability of the system.

4.1 Follow-up on concrete practices that could make an API better

4.1.1 General guidelines

**Do not show more than necessary**
All data members in the classes are private, the classes are final and the methods not necessary for users are private or default (only classes in the Cloudme package have access).

**Do not do more than necessary**
This is a hard question to answer as it is quite subjective. There is a method that takes a path to a local file and then uploads the file to Cloudme. The local file is automatically made to a file object in Java's standard library. It would be possible to get a more focused API by forcing the user to make this transformation themselves. But I think the benefits outweigh the negative aspects in this case, also you could always implement a method where the user needs to give a file object to the method in future.

**Be careful with using other API’s in your API**
Cloudmes server API is obviously used as it is the whole point of the wrapper. Apaches HttpClient is also used to be able to communicate with Cloudmes server. To implement the same functionality as Apaches HttpClient yourself only to be used by this API is not realistic.

**Use Modular architecture with low coupling between modules**
As all operations are based around the fact that you are logged in through CloudmeUser, there are a dependency between CloudmeUser and the other classes that supply public methods. You could also argue that this API is too small to consist of several modules; it could be seen as one.

**Use well thought out names for classes and methods. Follow a naming standard.**
I use the naming convention for Java by oracle, [10].

**Documentation is important**
All public methods are documented with Javadoc.

4.1.2 Class design

**Factories is preferred over pure constructors**
There is only one public constructor in the API. That constructor is in CloudmeUser which is used to log in. To add a factory for this particular constructor is not very motivated as there is only one kind of user.
Methods is preferred over public variables
There are no public variables in the API. All access and modification is done through getters and setters.

[JAVA][C++11] Make all classes final
All classed are final

Put public declarations above protected and private in the code
This was not strictly followed. I had public methods that used the same private methods. When implementing and troubleshooting it felt easier to have the private method closer to the public ones as it made discovering bugs easier. All private variables are at the bottom of the code. I was not able to rank methods on which is being used the most as I do not have enough data to make sound decisions.

Use Create-Set-Call instead of forcing arguments in the constructor
They are as said before only public constructor. The constructor has two arguments that is needed to enable the user to log in. These arguments are always needed as the class wants to listen for changes as soon as possible. To use Create-Set-Call in this case would be unnecessary as you always need both arguments.

4.1.3 Method design
Use the same parameter order in methods that use the same or similar parameters
I have consistently tried to use the same parameter order.

Do not use too many in parameters to methods
I have at most three in parameters on public methods.

Avoid using the same type of parameters in a row
I chose to use the String data type for quite a few arguments, making it impossible to avoid using the same parameters in a row for some cases without adding custom data types.

Don't use default values on in parameters
There are no default values on any parameters.

Use custom data types or ENUMS instead of base types if all values of the base type are not allowed
Found no suitable place to use this.

Do not make the user do something that you could automate in your methods
I would like to say that I follow this, but I analyze my own code so the correctness can be discussed.

4.1.4 Exception design
Notify the user about errors as soon as possible
The methods throw exceptions as soon as possible.

[Java] Checked or unchecked exceptions
I use checked exceptions as the user can recover from most errors in the API.
4.2 Complexity of the API wrapper

A complex API should be less usable than a less complex one [11]. To calculate the complexity of an API the normal metrics such as cyclomatic complexity are not very useful as the implementation part of the API is not visible for the user. Instead you can use the following metrics [11] [12] to calculate the complexity of API’s (or code designs in general). All of the metrics use a table to map data types to weighted values where a larger value indicates higher complexity.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bool</td>
<td>0</td>
</tr>
<tr>
<td>Char and Int</td>
<td>1</td>
</tr>
<tr>
<td>Float</td>
<td>2</td>
</tr>
<tr>
<td>Array</td>
<td>3</td>
</tr>
<tr>
<td>Pointer</td>
<td>5</td>
</tr>
<tr>
<td>Record, Struct or Object</td>
<td>6</td>
</tr>
<tr>
<td>File</td>
<td>10</td>
</tr>
</tbody>
</table>

*Fig.2 Type-Value Table given in [12]*

4.2.1 Metrics defined in [12]

**Operation argument complexity (OAC):**

\[ OAC = \sum P(i) \]

where P(i) is the value of each parameter and the return value, according to fig.2.

**Interface Size (IS):**

\[ IS = N + \sum P(i) \]

where N is the number of parameters and P(i) is the value of each parameter and the return value, according to fig.2. To get the IS for the whole class just take the sum of the IS for all its methods.

**Interaction Level (IL):**

\[ IL = I + \sum S(i) \]

where I is the number of interactions and S(i) is the value for each interaction. To calculate the value of an interaction you multiple the values for the data types involved. To get the IL for the whole class just take the sum of the IL for all its methods.

4.2.2 Example

```java
public void changeFolderName(String srcpath, String newname) throws CloudmeException {
    CloudmeFolder temp = this.foldertree.getChildFolderFromPath(srcpath);
    String action = "modifyFolder";
    String body = "<folder id=" + temp.getParentId() + ""/>
                   <childFolder id=""+temp.getId()+""
                   +""+"/>+"newname+"</name>";
    CloudmeMessenger messenger = new CloudmeMessenger();
    String returnmessage = messenger.callServerStringResponse(user.getCredentials(),
    action, body);
    Cloudme.checkForError(returnmessage);
    temp.setName(newname);
}
```

\[ OAC = 6(string)+6(string) = 12 \]
\[ IS = 2(2 parameters)+6(string)+6(string) = 14 \]
\[ IL = 3(3 interactions) + (6*6 + 6*6 + 6*6)(value of each interaction) = 111 \]
4.2.3 Possibilities
With these metrics you could compare this API with similar ones and get a guideline of its relative usability. Unfortunately there does not seem to be any publicly available tools that analyze code after these specific metrics at this point. In [11] they use a tool called Metrix, but it does not seem to have been released.
5 Conclusion and future work

5.1 Conclusion
An API wrapper for the CloudMe service has been developed in Java. The goal was to make it as usable as possible while still fulfilling functional requirements. In regards to the practices said to improve usability in the report, the wrapper should be useable as it follows most of them, with a few exceptions that I could motivate.

5.2 Future work
There are several possibilities. You could make a static analysis of the usability using the method described in section 4.2 and compare it to other API’s. You could add additional functionality to the API wrapper as it does not cover all the functionality of CloudMe’s core API. Examples of added functionality could be ACL (access control list) and group management. You could port the API wrapper to other programming languages to increase the potential user base.
References

1. Usability 101: Introduction to Usability, Jakob Nielsen’s Alertbox: January 1, 1970
   http://www.nngroup.com/articles/usability-101-introduction-to-usability/
2. How to Design a Good API and Why it Matters, Joshua Bloch,
   http://lcsd05.cs.tamu.edu/slides/keynote.pdf
4. API Usability: Guidelines to improve your code ease of use, Mathieu Jacques, 21 Nov, 2004
   http://www.codeproject.com/Articles/8707/API-Usability-Guidelines-to-improve-your-code-ease#2.3.1
5. Unchecked Exceptions – The Controversy,
   http://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html
7. XML Linking Language(XLink) Version 1.1, 06 May 2010,
   http://www.w3.org/TR/xlink11/
8. API Peer Reviews: A method for Evaluating Usability of Application programming Interfaces
    http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html#367
13. Accidents and Unintended Uses – Top 10, Jonny Williamson,
1. Tutorials
   1.1 Create a new folder and upload a file
   1.2 Creating, editing and downloading a file
   1.3 File and Folder browser
   1.4 File search
   1.5 Filelistener
   1.6 Using key values

2. References
   2.1 Class diagram
   2.2 Classes
   2.3 Interfaces

3. Get started
   3.1 Eclipse
Create a new folder and upload a file

To begin with we have the following code to login

```java
import Cloudme.CloudmeException;
import Cloudme.CloudmeUser;

public class cloudmetest {

    public static void main(String[] args) throws CloudmeException {
        CloudmeUser user = new CloudmeUser("username", "password");
        user.killUser();
    }
}
```

To create a new folder we can use CloudmeFolderManager's newFolder method, which returns a CloudmeFolder object.

```java
CloudmeFolder newfolder = user.getFolderManager().newFolder("/newfolder");
```

Now we want to upload a new file to our new folder. You can do it in two ways

**Upload a file via CloudmeFolder**

```java
newfolder.uploadFile("D:/Downloads/image1.jpg");
```

**Upload it via CloudmeFileManager**

```java
user.getFileManager().uploadFile("D:/Downloads/image2.jpg", "/newfolder/");
```

Full example

```java
import Cloudme.CloudmeException;
import Cloudme.CloudmeFolder;
import Cloudme.CloudmeUser;

public class cloudmetest {

    public static void main(String[] args) throws CloudmeException {
        CloudmeUser user = new CloudmeUser("username", "password");
        CloudmeFolder newfolder = user.getFolderManager().newFolder("/newfolder");
        newfolder.uploadFile("D:/Downloads/image1.jpg");
        user.getFileManager().uploadFile("D:/Downloads/image2.jpg", "/newfolder/");
        user.killUser();
    }
}
```
Creating, editing and downloading a file

First we create a new document on Cloudme through CloudmeFileManager.

```java
CloudmeFile file = user.getFileManager().createDocument("/myfile1.txt");
```

Now we add some data to that file in the local memory.

```java
file.setData("this is some nice text in the file2".getBytes());
```

Let’s save the data to CloudMe’s server.

```java
file.saveFile();
// save local file to server
file.readFile();
// to make sure the local file data is the same as the servers data
```

Now we download the file to the local hard drive.

```java
file.downloadFile("D:/localfile2.txt");
```

Full example

```java
import java.util.ArrayList;
import Cloudme.CloudmeException;
import Cloudme.CloudmeFile;
import Cloudme.CloudmeFolder;
import Cloudme.CloudmeUser;

public class cloudmetest {
    /**
     * @param args
     * @throws CloudmeException
     */
    public static void main(String[] args) throws CloudmeException {
        CloudmeUser user = new CloudmeUser("username","password");
        CloudmeFile file = user.getFileManager().createDocument("/myfile1.txt");
        file.setData("this is some nice text in the file2".getBytes());
        file.saveFile();
        file.readFile();
        file.downloadFile("D:/localfile2.txt");
        user.killUser();
    }
}
```
In this tutorial we will print all files on the drive, to demonstrate how you can iterate through the folder/file structure.

First we would like to find the root folder.

```java
CloudmeFolderNode node = user.getFolderManager().getFolderTree();
```

We want to print all the files contained in each folder, we could use a recursive function to achieve this.

```java
public static void printAllFiles(CloudmeFolderNode node) throws CloudmeException{
}
```

To get all files for the current node, we retrieve the folder object and call the function `getAllFiles`

```java
for(CloudmeFile f : node.getFolder().getFiles()){
    System.out.println(f.getMetadata().getName());
}
```

All files for the current folder have been printed now, so let's call this function again for its children

```java
for(CloudmeFolderNode n : node.getChildren()){
    printAllFiles(n);
}
```

Full example

```java
import java.util.ArrayList;
import Cloudme.CloudmeException;
import Cloudme.CloudmeFile;
import Cloudme.CloudmeFolder;
import Cloudme.CloudmeFolderNode;
import Cloudme.CloudmeUser;

public class cloudmetest {
    public static void printAllFiles(CloudmeFolderNode node) throws CloudmeException{
        for(CloudmeFile f : node.getFolder().getFiles()){
            System.out.println(f.getMetadata().getName());
        }
        for(CloudmeFolderNode n : node.getChildren()){
            printAllFiles(n);
        }
    }

    public static void main(String[] args) throws CloudmeException {
        CloudmeUser user = new CloudmeUser("username", "password");
        CloudmeFolderNode node = user.getFolderManager().getFolderTree();
        printAllFiles(node);
        user.killUser();
    }
}
```
Let's say we want to find all jpg files on the hard drive, then we can use CloudmeFileManager's fileSearch. Here we search the root folder and all the sub folders for files that contain jpg in their name. Recurisive must be set to true to search sub folders. We then print the name of all found files. Information about a file is stored in CloudmeMetadata.

```java
ArrayList<CloudmeFile> files = user.getFileManager().fileSearch("/", "jpg", true);
for(CloudmeFile f : files){
    System.out.println(f.getName());
}
```

If it's a new CloudMe account the files printed should be

- Halmstad.jpg
- Berg.jpg
- Country Side.jpg
- Leafs2.jpg

You can also conduct unique file search, that will return the first found file that fulfill the search term. In this example we search for the file "Notepad.xml" and then prints its metadata.

```java
CloudmeFile file = user.getFileManager().simpleUniqueFileSearch("/", "Notepad.xml", true);
file.getMetadata().print();
```

Should print something like this

```xml
<atom:title>Notepad.xml</atom:title>
<atom:link rel='alternate' type='text/xml' href='http://os.cloudme.com/v1/documents/562958547511009/4400003849/1'/>
```

Full example

```java
import java.util.ArrayList;
import Cloudme.CloudmeException;
import Cloudme.CloudmeFile;
import Cloudme.CloudmeFolder;
import Cloudme.CloudmeUser;

public class cloudmetest {
    public static void main(String[] args) throws CloudmeException {
        CloudmeUser user = new CloudmeUser("username", "password");
        ArrayList<CloudmeFile> files = user.getFileManager().fileSearch("/", "jpg", true);
        for(CloudmeFile f : files){
            System.out.println(f.getName());
        }

        CloudmeFile file = user.getFileManager().simpleUniqueFileSearch("/", "Notepad.xml", true);
        file.getMetadata().print();
        user.killUser();
    }
}
```
It is possible to add listeners to folders and files. In this example we will look at file listeners.

To add a file listener you call CloudmeFile's addListener method:

```java
public void addListener(CloudmeFileListener listener);
```

When you create a file listener it should implement the interface CloudmeFileListener:

```java
package Cloudme;

public interface CloudmeFileListener {
    public void removeThisFile(CloudmeFile fileevent, boolean moved);
    public void fileChanged(CloudmeFile fileevent, CloudmeFile changes);
}
```

So for example if a file is removed on CloudMe, removeThisFile will be called for all listeners added to that file. If a file is moved removeThisFile will be called as well, but the moved argument will be set to true.

Here is an example of a class implementing the CloudmeFileListener interface that could be added as a file listener:

```java
import Cloudme.CloudmeFile;
import Cloudme.CloudmeFileListener;

public class ConcreteFileListener implements CloudmeFileListener {
    @Override
    public void removeThisFile(CloudmeFile fileevent, boolean moved) {
        System.out.println(fileevent.getMetadata().getName() + " DELETED, FROM LISTENER");
    }
    @Override
    public void fileChanged(CloudmeFile fileevent, CloudmeFile changes) {
        System.out.println("CHANGED, FROM LISTENER");
    }
}
```
Using key values

It is possible to store keys (variables) on CloudMe. You could for example store some variable for a game and it would be reachable from all devices with an internet connection. The methods for key operations lie in the CloudmeUser class.

Setting a key

```java
user.setKeyValue("/game1", "var", "5"); // sets the variable var to 5, var lies in the game1 domain.
```

Getting all keys for some domain and its subdomains

```java
String allkeys = user.getAllKeyValues("/"); // "/" is the top domain so returns all user keys stored
```

Returns

```java
Returns<
<
<game1
var
= '5'
/>
</

Getting a specific key

```java
String var1 = user.getKeyValue("/game1", "var");
```

Returns

```java
5
```

Full example

```java
import Cloudme.CloudmeException;
import Cloudme.CloudmeFolder;
import Cloudme.CloudmeUser;

public class cloudmetest {

/**
 * @param args
 * @throws CloudmeException
 */
public static void main(String[] args) throws CloudmeException {
    CloudmeUser user = new CloudmeUser("username", "password");
    user.setKeyValue("/game1", "var", "5");
    String allkeys = user.getAllKeyValues("/");
    String var1 = user.getKeyValue("/game1", "var");
    user.killUser();
}
}
```

Full example

```java
import Cloudme.CloudmeException;
import Cloudme.CloudmeFolder;
import Cloudme.CloudmeUser;

public class cloudmetest {

/**
 * @param args
 * @throws CloudmeException
 */
public static void main(String[] args) throws CloudmeException {
    CloudmeUser user = new CloudmeUser("username", "password");
    user.setKeyValue("/game1", "var", "5");
    String allkeys = user.getAllKeyValues("/");
    String var1 = user.getKeyValue("/game1", "var");
    user.killUser();
}
```
<table>
<thead>
<tr>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudmeFile</td>
</tr>
<tr>
<td>CloudmeFileManager</td>
</tr>
<tr>
<td>CloudmeFolder</td>
</tr>
<tr>
<td>CloudmeFolderManager</td>
</tr>
<tr>
<td>CloudmeFolderNode</td>
</tr>
<tr>
<td>CloudmeMetadata</td>
</tr>
<tr>
<td>CloudmeStreamData</td>
</tr>
<tr>
<td>CloudmeUser</td>
</tr>
</tbody>
</table>
CloudmeFile

This class represents a file on CloudMe.

**Methods**

- `void deleteFile()`: Deletes this file. This object will still exist for as long as you keep the reference to it, but the file on the server will be deleted.

  *Throws*: `CloudmeException`

- `void readFile()`: Reads a file's data into this CloudmeFile object.

  *Throws*: `CloudmeException`

- `void readStream(String number)`: Reads a stream's data into a CloudmeStream object that is a member in this CloudmeFile object.

  **Parameters**:
  - `number`: The stream's number.

- `void downloadStream(String number, String destpath)`: Downloads the data from a given stream that have been read into local memory to a destination on the local hard drive.

  **Parameters**:
  - `number`: The stream's number.
  - `destpath`: Path on the local hard drive where the stream should be downloaded.

  *Returns*: `null`

  *Throws*: `CloudmeException` or `FileNotFoundException` if the file cannot be found on the local hard drive.

- `ArrayList<String> getAvailableStreams()`: Returns the available streams for this file.

  *Returns*: An `ArrayList` of `String` that contain all available stream numbers.

  *Throws*: `CloudmeException` if the file cannot be found or if there are issues with retrieving the available streams.

- `void downloadFile(String destpath)`: Downloads the file to the specified destination path.

  **Parameters**:
  - `destpath`: Path on the local hard drive where the file should be downloaded.

  *Returns*: `null`

  *Throws*: `CloudmeException` if the file cannot be downloaded or if there are issues with the download process.

- `CloudmeFile copyFile(String destpath)`: Copies this file to a new location.

  **Parameters**:
  - `destpath`: The destination path where the file should be copied.


  *Throws*: `CloudmeException` if the file cannot be copied or if there are issues with the copy process.

- `CloudmeFile copyFile(CloudmeFolder folder, String name)`: Copies this file to a new folder within the specified folder.

  **Parameters**:
  - `folder`: The CloudmeFolder to which the file should be copied.
  - `name`: The name for the new file.


  *Throws*: `CloudmeException` if the file cannot be copied or if there are issues with the copy process.

- `void moveFile(String destpath, String name)`: Moves this file to a new location.

  **Parameters**:
  - `destpath`: The destination path where the file should be moved.
  - `name`: The name for the new file.

  *Throws*: `CloudmeException` if the file cannot be moved or if there are issues with the move process.

- `void moveFile(CloudmeFolder folder, String name)`: Moves this file to a new folder within the specified folder.

  **Parameters**:
  - `folder`: The CloudmeFolder to which the file should be moved.
  - `name`: The name for the new file.

  *Throws*: `CloudmeException` if the file cannot be moved or if there are issues with the move process.

- `void renameFile(String newname)`: Renames this file.

  **Parameters**:
  - `newname`: The new name for the file.

  *Throws*: `CloudmeException` if the file cannot be renamed or if there are issues with the rename process.

- `CloudmeMetadata getMetadata()`: Returns the metadata for this file.

  *Returns*: `null`

  *Throws*: `CloudmeException` if the file's metadata cannot be retrieved.

- `void setMetadata(CloudmeMetadata metadata)`: Sets the metadata for this file.

  **Parameters**:
  - `metadata`: The new metadata for the file.

  *Throws*: `CloudmeException` if the file's metadata cannot be set.

- `void saveMetadata()`: Saves the metadata for this file.

  *Throws*: `CloudmeException` if the file's metadata cannot be saved.

- `void saveFile()`: Saves the file to the local hard drive.

  *Throws*: `CloudmeException` if the file cannot be saved or if there are issues with the save process.

- `void copyStream(String sourcestreamnumber, String destpath, String deststreamnumber)`: Copies a stream from one file to another.

  **Parameters**:
  - `sourcestreamnumber`: The number of the source stream.
  - `destpath`: The destination path on the local hard drive.
  - `deststreamnumber`: The number of the destination stream.

  *Throws*: `CloudmeException` if the stream cannot be copied or if there are issues with the copy process.

- `void deleteStream(String sourcenumber)`: Deletes a stream from this file.

  **Parameters**:
  - `sourcenumber`: The number of the stream to be deleted.

  *Throws*: `CloudmeException` if the stream cannot be deleted or if there are issues with the delete process.

- `org.w3c.dom.Document getXmlDocument()`: Returns an XML representation of this file.

  *Returns*: `null`

  *Throws*: `CloudmeException` if the file's XML representation cannot be retrieved.

- `byte[] getData()`: Returns the raw data for this file.

  *Returns*: `null`

  *Throws*: `CloudmeException` if the file's data cannot be retrieved.

- `void setData(byte[] data)`: Sets the raw data for this file.

  **Parameters**:
  - `data`: The new data for the file.

  *Throws*: `CloudmeException` if the file's data cannot be set.


  **Parameters**:
  - `document`: The XML document containing the file's data.

  *Throws*: `CloudmeException` if the file's data cannot be set.

- `void addChangeListener(CloudmeFileListener listener)`: Adds a listener to this file.

  **Parameters**:
  - `listener`: The listener to be added.

  *Throws*: `CloudmeException` if the file's listener cannot be added.

- `ArrayList<CloudmeStreamData> getLoadedStreams()`: Returns the streams that have been loaded for this file.

  *Returns*: An `ArrayList` of `CloudmeStreamData` objects.

  *Throws*: `CloudmeException` if the loaded streams cannot be retrieved.
Downloads this file's data to the local hard drive
Parameters:
destpath: Path on the local hard drive where the file should downloaded
Throws:
CloudmeException

Cloudme copyFile(String destpath)
Copies this file to another location
Parameters:
destpath: Path to the new location
Returns:
Returns a CloudmeFile object that represent the copy of the file
Throws:
CloudmeException

Cloudme copyFile(CloudmeFolder folder, String name)
Copies this file to another location
Parameters:
folder: The CloudmeFolder where the copy should be placed
name: The copies name
Returns:
Returns a CloudmeFile object that represent the copy of the file
Throws:
CloudmeException

Cloudme moveFile(String destpath)
Moves this file to another location
Parameters:
destpath: Path to the new location
Throws:
CloudmeException

Cloudme moveFile(CloudmeFolder folder, String name)
Moves this file to another location
Parameters:
folder: The CloudmeFolder where the moved file should be placed
name: The new name for the moved file
Throws:
CloudmeException

void renameFile(String newname)
Renames this file
Parameters:
newname: The new name for this file
Throws:
CloudmeException

CloudmeMetadata getMetadata()
Returns the metadata for this file
Returns:
Returns a CloudmeMetadata object that holds the metadata.
Throws:
CloudmeException

void setMetadata(CloudmeMetadata metadata)
Sets the metadata for this file
Parameters:
metadata: Sets the metadata for this file to this CloudmeMetadata object
Throws:
CloudmeException

void saveMetadata()
Save local metadata for this file to the CloudMe file
Throws:
CloudmeException

void saveFile()
Save local file data to the CloudMe file
Throws:
CloudmeException

void copyStream(String sourcestreamnumber,String destpath,String deststreamnumber)
Copies a stream from this file to another file
Parameters:
sourcestreamnumber: The stream number of the stream you wish to copy
destpath: Path to the file where you wish to copy the stream
deststreamnumber: Copy the stream to this number
Throws:
CloudmeException
### deleteStream(String sourcenumber)

Deletes a stream on this file

**Parameters:**
- sourcenumber: The streams number

**Throws:**
- CloudmeException

### getXmlDocument()

If the file is an xml file, this return a dom document of the xml in the files data

**Returns:**
- Returns a org.w3c.dom.Document of the files data.

**Throws:**
- CloudmeException

### getData()

Return the local data of this file

**Returns:**
- Returns a byte array of this files local data.

### setData(byte[] data)

Sets the local data to a new value

**Parameters:**
- data: The new value

**Throws:**
- CloudmeException

### setData(org.w3c.dom.Document document)

Sets the local data from a xml dom document

**Parameters:**
- document: The dom document that you wish to set the files data to.

**Throws:**
- CloudmeException

### addListener(CloudmeFileListener listener)

Adds a file listener to this file

**Parameters:**
- listener: A class that implements the CloudmeFileListener interface

**Throws:**
- CloudmeException

### getLoadedStreams()

Returns a list of all stream currently loaded in the local memory

**Returns:**
- Returns an ArrayList of CloudmeStreamData the represents all loaded streams

**Throws:**
- CloudmeException
# CloudmeFileManager

This class is a member of CloudmeUser. It is responsible for file operations.

## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[] readFile(String path)</td>
<td>Returns a byte array of a files data</td>
</tr>
<tr>
<td>void downloadFile(String sourcepath, String destpath)</td>
<td>Downloads a file to the local hard drive</td>
</tr>
<tr>
<td>byte[] readStream(String path, String number)</td>
<td>Returns a byte array of a streams data</td>
</tr>
<tr>
<td>void downloadStream(String path, String number, String destpath)</td>
<td>Downloads a stream to the local hard drive</td>
</tr>
<tr>
<td>ArrayList&lt;String&gt; getAvailableStreams(String path)</td>
<td>Return an ArrayList of all available streams on a file.</td>
</tr>
</tbody>
</table>

## Method detail

### readFile(String path)

Returns a byte array of a files data

**Parameters:**  
path: path to the file

**Returns:**  
Returns a byte array containing the files data.

**Throws:**  
CloudmeException

### downloadFile(String sourcepath, String destpath)

Downloads a file to the local hard drive

**Parameters:**  
sourcepath: Path to the file on CloudMe  
destpath: Path to the location where the file should be stored on the local hard drive

**Throws:**  
CloudmeException

### readStream(String path, String number)

Returns a byte array of a streams data

**Parameters:**  
path: Path to the file  
number: The streams number

**Returns:**  
Returns a byte array containing the streams data.

**Throws:**  
CloudmeException

### downloadStream(String path, String number, String destpath)

Downloads a stream to the local hard drive

**Parameters:**  
sourcepath: Path to the file on CloudMe  
number: The streams number  
destpath: Path to the location where the stream should be stored on the local hard drive

**Throws:**  
CloudmeException

### getAvailableStreams(String path)

Return an ArrayList of all available streams on a file.
### Parameters:
path: Path to the file

### Returns:
Returns an ArrayList of String that contain all available streams

### Throws:
CloudmeException

### void copyStream(String sourcepath, String sourcenumber, String destpath, String destnumber)
Copies a stream from a file to another file's stream

#### Parameters:
sourcepath: Path to the file where the stream you wish to copy is located
sourcenumber: The stream's number
destpath: Path to the file where you want to copy the stream
destnumber: The number where you want to place the copied stream

#### Throws:
CloudmeException

### void deleteStream(String path, String sourcenumber)
Deletes a stream from a file

#### Parameters:
path: Path to the file where the stream you wish to delete is located
sourcenumber: The stream's number

#### Throws:
CloudmeException

### CloudmeFile createDocument(String path)
Creates a new document at a specified location

#### Parameters:
path: Path where the document should be created

#### Returns:
Returns a CloudmeFile object that represents the new document

#### Throws:
CloudmeException

### void deleteFile(String path)
Deletes a file

#### Parameters:
path: Path to the file you wish to delete.

#### Throws:
CloudmeException

### CloudmeFile uploadFile(String sourcepath, String destpath)
Uploads a file from the local hard drive to CloudMe

#### Parameters:
sourcepath: Path to the file on the local hard drive
destpath: The path on CloudMe where the file should be uploaded

#### Returns:
Returns a CloudmeFile object that represents the uploaded file

#### Throws:
CloudmeException

### CloudmeFile simpleUniqueFileSearch(String path, String query, boolean recursive)
Searches for files that fulfill the query. Returns the first file found. It will search all subfolders if recursive is set to true, otherwise only the folder specified by path.

#### Parameters:
path: Path to the folder that you want to search
query: What you want to search for, for example "myfile.jpg"
recursive: Boolean, true = search subfolders; false = don't search subfolders;

#### Returns:
Returns a CloudmeFile object that represents the found file.

#### Throws:
CloudmeException

### ArrayList<CloudmeFile> fileSearch(String path, String query, boolean recursive)
Searches for files that fulfill the query. Returns all files found. It will search all subfolders if recursive is set to true, otherwise only the folder specified by path.

#### Parameters:
path: Path to the folder that you want to search
query: What you want to search for, for example "myfile.jpg"
recursive: Boolean, true = search subfolders; false = don't search subfolders;

#### Returns:
Returns an ArrayList of CloudmeFile that represent all found files

#### Throws:
CloudmeException

### ArrayList<CloudmeFile> getFiles(String path)
Returns all files in a folder

#### Parameters:
path: Path to the folder

#### Returns:
Returns an ArrayList of CloudmeFile that represent all found files

#### Throws:
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Parameters</th>
<th>Returns</th>
<th>Throws</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>getFile(String path)</strong></td>
<td>Returns a file from a specified location</td>
<td>path: The path to the file</td>
<td>Returns a CloudmeFile object that represent the file fetched</td>
<td>CloudmeException</td>
</tr>
<tr>
<td><strong>copyFile(String srcpath,String destpath)</strong></td>
<td>Copies a file to another location</td>
<td>srcpath: Path to the file you wish to copy</td>
<td>Returns a CloudmeFile object that represents the copy of the file</td>
<td>CloudmeException</td>
</tr>
<tr>
<td><strong>copyFile(String srcpath,CloudmeFolder folder,String name)</strong></td>
<td>Copies a file to another location</td>
<td>srcpath: Path to the file you wish to copy</td>
<td>Returns a CloudmeFile object that represents the copy of the file</td>
<td>CloudmeException</td>
</tr>
<tr>
<td><strong>moveFile(String srcpath,String destpath)</strong></td>
<td>Moves a file to another location</td>
<td>srcpath: Path to the file you wish to move</td>
<td>Returns a CloudmeFile object that represents the moved file</td>
<td>CloudmeException</td>
</tr>
<tr>
<td><strong>moveFile(String srcpath,CloudmeFolder folder,String name)</strong></td>
<td>Moves a file to another location</td>
<td>srcpath: Path to the file you wish to move</td>
<td>Returns a CloudmeFile object that represents the moved file</td>
<td>CloudmeException</td>
</tr>
<tr>
<td><strong>renameFile(String path,String newname)</strong></td>
<td>Renames a file</td>
<td>path: Path to the file you wish to rename</td>
<td></td>
<td>CloudmeException</td>
</tr>
</tbody>
</table>
CloudmeFolder

This class represents a folder on CloudMe.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getName()</td>
<td>Returns the name of the folder. Returns a string with the folder name.</td>
</tr>
<tr>
<td>String getId()</td>
<td>Returns the id of the folder. Returns a string with the folder id.</td>
</tr>
<tr>
<td>String getParentId()</td>
<td>Returns the id of the folder's parent. Returns a string with the parent id.</td>
</tr>
<tr>
<td>CloudmeFolder getParentFolder()</td>
<td>Returns the parent of this folder. Returns a CloudmeFolder object that represents the parent.</td>
</tr>
<tr>
<td>void changeFolderName(String newname)</td>
<td>Changes the name of this folder. Parameters: newname: The new name of the folder.</td>
</tr>
<tr>
<td>ArrayList&lt;CloudmeFolder&gt; getChildFolders()</td>
<td>Returns all first level subfolders of this folder. Returns an ArrayList of CloudmeFolder objects that represent the subfolders.</td>
</tr>
<tr>
<td>CloudmeFolder newFolder(String foldername)</td>
<td>Creates a new folder in this folder. Parameters: foldername: Name of the new folder.</td>
</tr>
</tbody>
</table>

Method detail

String getName()

Returns the name of the folder.

Returns:

String getId()

Returns the id of the folder.

Returns:

String getParentId()

Returns the id of the folders parent.

Returns:

CloudmeFolder getParentFolder()

Returns the parent of this folder.

Returns:

Thrown:

CloudmeException

void changeFolderName(String newname)

Changes the name of this folder.

Parameters:

newname: The new name of the folder.

Thrown:

CloudmeException

ArrayList<CloudmeFolder> getChildFolders()

Returns all first level subfolders of this folder.

Returns:

Thrown:

CloudmeException

CloudmeFolder newFolder(String foldername)

Creates a new folder in this folder.

Parameters:

foldername: Name of the new folder.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void deleteFolder()</code></td>
<td>Deletes this folder. This object will still exist as long as you have a reference to it, but the folder is deleted on the server.</td>
</tr>
<tr>
<td><code>void moveFolder(String destpath)</code></td>
<td>Moves this folder. It is not safe to use this folder object after, requires handling by a concrete folder listener.</td>
</tr>
<tr>
<td><code>void moveFolder(CloudmeFolder destfolder, String name)</code></td>
<td>Moves this folder. It is not safe to use this folder object after, requires handling by a concrete folder listener. Parameters: destfolder (The CloudmeFolder where you want to move this folder) name (The name the folder gets after being moved).</td>
</tr>
<tr>
<td><code>CloudmeFile simpleUniqueFileSearch(String query, boolean recursive)</code></td>
<td>Searches after files that fulfill the query. Returns the first file found. It will search all subfolders if recursive is set to true, otherwise only this folder. Parameters: query (What you want to search for, for example &quot;myfile.jpg&quot;) recursive (Boolean, true = search subfolders; false = dont search subfolders). Returns: A CloudmeFile object that represents the found file.</td>
</tr>
<tr>
<td><code>ArrayList&lt;CloudmeFile&gt; FileSearch(String query, boolean recursive)</code></td>
<td>Searches after files that fulfill the query. Returns all files found. It will search all subfolders if recursive is set to true, otherwise only this folder. Parameters: query (What you want to search for, for example &quot;myfile.jpg&quot;) recursive (Boolean, true = search subfolders; false = dont search subfolders). Returns: An ArrayList of CloudmeFile that represents all found files.</td>
</tr>
<tr>
<td><code>ArrayList&lt;CloudmeFile&gt; getFiles()</code></td>
<td>Returns all files in this folder. Returns: An ArrayList of CloudmeFile that represents all found files.</td>
</tr>
<tr>
<td><code>CloudmeFile uploadFile(String sourcepath)</code></td>
<td>Uploads a file to this folder. Parameters: sourcepath (Path to the file on the local hard drive). Returns: A CloudmeFile that represents the uploaded file.</td>
</tr>
<tr>
<td><code>CloudmeFile getFile(String name)</code></td>
<td>Returns the file with name name. Parameters: name (Name of the file). Returns: A CloudmeFile object that represents the file.</td>
</tr>
<tr>
<td><code>String getPath()</code></td>
<td>Returns the path to this folder. Returns: A string with the path to this folder.</td>
</tr>
<tr>
<td><code>void addListener(CloudmeFolderListener listener)</code></td>
<td>Adds a new listener to this folder. Parameters:</td>
</tr>
</tbody>
</table>
The listener implementing the CloudmeFolderListener interface you want to add.
CloudmeFolderManager

This class is a member of CloudmeUser. It is responsible for folder operations.

Methods

<table>
<thead>
<tr>
<th>Method Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudmeFolder newFolder(String path)</td>
<td>Creates a new folder. Location and name specified in the path.</td>
</tr>
<tr>
<td>void changeFolderName(String srcpath, String newname)</td>
<td>Changes the name of a folder.</td>
</tr>
<tr>
<td>void deleteFolder(String path)</td>
<td>Deletes a folder.</td>
</tr>
<tr>
<td>CloudmeFolder getFolder(String path)</td>
<td>Fetches a folder.</td>
</tr>
<tr>
<td>ArrayList&lt;CloudmeFolder&gt; getFolders(String path)</td>
<td>Returns all the first level subfolders of a folder.</td>
</tr>
<tr>
<td>void moveFolder(String srcpath, String destpath)</td>
<td>Moves a folder to a new destination. Not safe to use new folder location directly afterwards, server must send back information about new folder id etc.</td>
</tr>
<tr>
<td>void moveFolder(String srcpath, CloudmeFolder destfolder, String name)</td>
<td>Moves a folder to a new destination. Not safe to use new folder location directly afterwards, server must send back information about new folder id etc.</td>
</tr>
</tbody>
</table>

Method detail

CloudmeFolder newFolder(String path)

Creates a new folder. Location and name specified in the path.
Parameters:
path: The path to the new folder, for example "/oldfolder/newfolder"
Returns:
Returns a CloudmeFolder object that represent the new folder.
Throws:
CloudmeException

void changeFolderName(String srcpath, String newname)

Changes the name of a folder.
Parameters:
sropath: Path to the folder
newname: The new name
Throws:
CloudmeException

void deleteFolder(String path)

Deletes a folder.
Parameters:
path: Path to the folder that you wish to delete
Throws:
CloudmeException

CloudmeFolder getFolder(String path)

Fetches a folder.
Parameters:
path: Path to the folder
Returns:
Returns a CloudmeFolder object of the fetched folder.
Throws:
CloudmeException

ArrayList<CloudmeFolder> getFolders(String path)

Returns all the first level subfolders of a folder.
Parameters:
path: Path to folder
Returns:
Returns a ArrayList of CloudmeFolder objects, representing the subfolders.
Throws:
CloudmeException

void moveFolder(String srcpath, String destpath)

Moves a folder to a new destination. Not safe to use new folder location directly afterwards, server must send back information about new folder id etc.
Parameters:
sropath: Path to the folders old location
destopath: Path to the location you wish to move the folder
Throws:
CloudmeException

void moveFolder(String srcpath, CloudmeFolder destfolder, String name)

Moves a folder to a new destination. Not safe to use new folder location directly afterwards, server must send back information about new folder id etc.
Parameters:
sropath: Path to the folders old location
| destFolder | Cloudme-folder object that represent the folder where you wish to move the folder. |
| name | A new name for the moved folder |
| Throws | CloudmeException |

**CloudmeFolderNode getFolderTree()**

*Returns the root node of the local folder structure.*

**Returns:**

Returns a CloudmeFolderNode that represent the root folder.
CloudmeFolderNode

This class is responsible for keeping a local tree structure of the users folder structure on the server. The structure is updated automatically and can't be modified by the user directly. The root node is accessible from the CloudmeFolderManager class. It's possible to iterate through the folder tree with this class.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudmeFolder getFolder()</td>
<td>Returns the folder of this node.</td>
</tr>
<tr>
<td>CloudmeFolder getFolderWithoutListening()</td>
<td>Returns the folder of this node, but listeners are disabled.</td>
</tr>
<tr>
<td>ArrayList&lt;CloudmeFolderNode&gt; getChildren()</td>
<td>Returns the children of this node.</td>
</tr>
<tr>
<td>CloudmeFolderNode getParent()</td>
<td>Returns this node's parent.</td>
</tr>
<tr>
<td>void printTree()</td>
<td>Prints the folder structure with this node as the root folder.</td>
</tr>
</tbody>
</table>

Method detail

**CloudmeFolder getFolder()**

Returns the folder of this node.

**Returns:**

Returns a CloudmeFolder object that represents the folder in this node.

**CloudmeFolder getFolderWithoutListening()**

Returns the folder of this node, but listeners are disabled.

**Returns:**

Returns a CloudmeFolder object that represents the folder in this node. There is no point in adding listeners to that CloudmeFolder object as listeners will be disabled.

**ArrayList<CloudmeFolderNode> getChildren()**

Returns the children of this node.

**Returns:**

Returns the children of this node as a ArrayList of CloudmeFolderNode.

**CloudmeFolderNode getParent()**

Returns this node's parent.

**Returns:**

Returns this node's parent as a CloudmeFolderNode object.

**void printTree()**

Prints the folder structure with this node as the root folder.
CloudmeMetadata

A class responsible for handling a file's metadata. Each CloudmeFile object has a CloudmeMetadata object as a member.

Methods

```java
void addNewMetadata(String newmeta)
String getAttributeValue(String attribute)
String getFolderId()
String getId()
String getMime()
String getName()
String getString()
void print()
void setAttributeValue(String attribute, value)
```

Method detail

void addNewMetadata(String newmeta)

Adds custom metadata.

Parameters:

- `newmeta`: The new metadata you wish to add. The data should be in xml.

String getAttributeValue(String attribute)

Searches the metadata for an attribute and returns it if found.

Parameters:

- `attribute`: The attribute to search for.

Returns:

A found attribute.

Throws:

CloudmeException

String getFolderId()

Returns the folder id of this file's parent folder.

Returns:

A string with the folder id of this file's parent folder.

String getMime()

Returns the file type of this file in the MIME standard.

Returns:

A string that describes the file type in MIME.

String getName()

Returns the name of this file.

Returns:

A string with the file name.

String getString()

Returns the whole metadata structure as sent from CloudMe.

Returns:

A string with the whole metadata structure.

void print()

Prints the metadata structure to the console.

void setAttributeValue(String attribute, value)

Searches after a attribute and changes its value if found.

Parameters:

- `attribute`: The attribute to search for.
- `value`: The new value you wish to set.

Throws:

CloudmeException
CloudmeStreamData

CloudmeStreamData is a class used to store stream data in the local memory. At this point it is not possible to create new StreamData, only to modify/overwrite existing streams.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[] getData()</td>
<td>Returns the data stored in this stream</td>
</tr>
<tr>
<td>String getStreamnumber()</td>
<td>Returns the identifying number for this string</td>
</tr>
<tr>
<td>void setData(byte[] data)</td>
<td>Sets this streams data to something new</td>
</tr>
<tr>
<td>void setStreamnumber(String streamnumber)</td>
<td>Changes the identifying number of this stream.</td>
</tr>
</tbody>
</table>

Method detail

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte[] getData()</td>
<td>Returns a byte array containing the streams data.</td>
</tr>
<tr>
<td>String getStreamnumber()</td>
<td>Returns a string containing this stream identifying number.</td>
</tr>
<tr>
<td>void setData(byte[] data)</td>
<td>Sets a byte array of some new data you want to set this stream to.</td>
</tr>
<tr>
<td>void setStreamnumber(String streamnumber)</td>
<td>Changes the identifying number you wish to set this stream to.</td>
</tr>
</tbody>
</table>
CloudmeUser

CloudmeUser is the main class of the api. You login to CloudMe when you create an object of this class. From this class you can retrieve the file and folder manager to handle file and folder operations.

Objects of CloudmeUser runs in a thread to get receive events from cloudmes server, but the methods are not threaded. You must call killUser when you don't intend to use the CloudmeUser object anymore.

The class can also set and get keyvalues(similar to a database) that you store on CloudMe's server.

Constructors

<table>
<thead>
<tr>
<th>Constructor detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudmeUser(String username, String password)</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>String getAllKeyValues(String path)</td>
</tr>
<tr>
<td>CloudmeFileManager getFileManager()</td>
</tr>
<tr>
<td>CloudmeFolderManager getFolderManager()</td>
</tr>
<tr>
<td>CloudmeFolder getHomeFolder()</td>
</tr>
<tr>
<td>String getKeyValue(String path, String key)</td>
</tr>
<tr>
<td>void killUser()</td>
</tr>
<tr>
<td>void setKeyValue(String path, String key, String value)</td>
</tr>
</tbody>
</table>

Constructor detail

CloudmeUser(String username, String password)

Creates and logs in a user with a given username and password

Parameters:
username: The username of the user to login
password: The password of the user to login

Throws:
CloudmeException

Method detail

String getAllKeyValues(String path)

Returns all key values from a given path

Parameters:
path: The path where the key values are stored.

Returns:
Returns a string with all the keys and their values.

Throws:
CloudmeException

CloudmeFileManager getFileManager()

Returns the filemanager

Returns:
Returns the CloudmeFileManager object.

CloudmeFolderManager getFolderManager()

Returns the foldermanager

Returns:
Returns the CloudmeFolderManager object.

CloudmeFolder getHomeFolder()

Returns the homefolder of the user.

Returns:
Returns the homefolder as a CloudmeFolder object.

String getKeyValue(String path, String key)

Returns a key value by giving a path and key.

Parameters:
path: Path to where the key is stored
key: Specifies which key to get the value from

Returns:
Returns a string with the value of the key.

Throws:
CloudmeException

void killUser()

Kills the thread of the user. Use this method when you are no longer using the CloudmeUser object.

void setKeyValue(String path, String key, String value)

Sets a key to a value. If the key doesn't exist, it will be created. So you can also use the method to add new keys.
Parameters:
- path: Path to where the key is stored, or where you want the create a new key.
- key: Specifies which key to set the value on. If the key doesn't exist, it specifies the name of the new key that will be added.

Throws:
- CloudmeException
Interfaces

CloudmeFileListener
CloudmeFolderListener
CloudmeFileListener

This interface should be implemented for a file listener.

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>void fileChanged(CloudmeFile fileevent, CloudmeFile changes)</td>
</tr>
<tr>
<td>void removeThisFile(CloudmeFile fileevent, boolean moved)</td>
</tr>
</tbody>
</table>
CloudmeFolderListener

This interface should be implemented for a folder listener.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void addFolder(CloudmeFolder folderEvent, CloudmeFolder newFolder, boolean moved)</code></td>
<td></td>
</tr>
<tr>
<td><code>void fileAdded(CloudmeFolder folderEvent, CloudmeFile newFile, boolean moved)</code></td>
<td></td>
</tr>
<tr>
<td><code>void fileChanged(CloudmeFolder folderEvent, CloudmeFile changedFile)</code></td>
<td></td>
</tr>
<tr>
<td><code>void fileRemoved(CloudmeFolder folderEvent, CloudmeFile removedFile, boolean moved)</code></td>
<td></td>
</tr>
<tr>
<td><code>void folderChanged(CloudmeFolder folderEvent, CloudmeFolder changes)</code></td>
<td></td>
</tr>
<tr>
<td><code>void removeChildFolder(CloudmeFolder folderEvent, CloudmeFolder removedFolder, boolean moved)</code></td>
<td></td>
</tr>
<tr>
<td><code>void removeThisFolder(CloudmeFolder folderEvent, boolean moved)</code></td>
<td></td>
</tr>
</tbody>
</table>
The first thing you need to do is to add all the JAR files contained in the CloudmeAPI folder to your project. In eclipse you can do it as follows.

### Project>Properties

#### Java Build Path->Add external JARs

The following should compile if done correctly

```java
import Cloudme.CloudmeException;
import Cloudme.CloudmeUser;

public class cloudmetest {
    
    public static void main(String[] args) throws CloudmeException {
        CloudmeUser user = new CloudmeUser("username", "password");

        user.killUser();
    }
}
```
Appendix 2: Core API documentation
SOA-REST API

1. Get started

2. API

2.1 CloudMe account
   2.1.1 Login (SOAP)
   2.1.2 Account info (REST)

2.2 Drives (SOAP)

2.3 Folders
   2.3.1 Get folders (REST)
   2.3.2 Delete folder (SOAP)
   2.3.3 Get folder ACL (SOAP)
   2.3.4 Get folder XML (SOAP)
   2.3.5 Modify folder (SOAP)
   2.3.6 Move folder (SOAP)
   2.3.7 New folder (SOAP)
   2.3.8 Set folder ACL (SOAP)

2.4 Documents
   2.4.1 Download file (REST)
   2.4.2 Upload file (REST)
   2.4.3 Copy document (SOAP)
   2.4.4 Create document (SOAP)
   2.4.5 Delete document (SOAP)
   2.4.6 Get document ACL (SOAP)
   2.4.7 Load metadata (SOAP)
   2.4.8 Move document (SOAP)
   2.4.9 Rename document (SOAP)
   2.4.10 Set document ACL (SOAP)

2.5 Webshares
   2.5.1 Create webshare (REST)
   2.5.2 Change webshare (REST)
   2.5.3 Delete webshare (REST)

2.6 Locale
   2.6.1 Get locale (REST)
   2.6.2 Set locale (REST)

2.7 Portlisteners (SOAP)

2.8 Messaging (SOAP)
To use CloudMe’s server api you must be able to send SOAP and REST messages. If you have no previous knowledge of using them you might consider using the Java API. If you still wish to use the raw api, a few helper methods are provided for java. They require the Apache http components library.

```java
import org.apache.http.impl.client.RequestInterceptor;
import org.apache.http.HttpRequest;
import org.apache.http.HttpResponseException;
try {
    StringEntity entity = new StringEntity(xmlString, "UTF-8" );
    entity.setContentType("application/xml");
    httpPut.setEntity(entity);
    System.out.println("Request Line:");
    System.out.println(httpPut.getRequestLine().toString() + "\n\n");
    HttpResponse response = httpClient.execute(httpPut);
    // Call to getResponse needs to be done before httpClient is shut down
    // otherwise the response disappeared and null is returned
    String stringResponse = getResponse(response); // not a thread safe
    return stringResponse;
} catch (Exception e) {
    System.out.println("exception");
    System.out.println(e.getMessage());
}
return null;

// Used for DELETE REST call
private synchronized String deleteServerREST(String deleteCall) {
    DefaultHttpClient httpClient = getConnection();
    HttpDelete httpDelete = new HttpDelete(deleteCall);
    try {
        System.out.println("Delete Request Line: ");
        System.out.println(httpDelete.getRequestLine().toString() + "\n\n");
        HttpResponse response = httpClient.execute(httpDelete);
        // Call to getResponse needs to be done before httpClient is shut down
        // otherwise the response disappeared and null is returned
        String stringResponse = getResponse(response); // not a thread safe
        return stringResponse;
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    httpClient.getConnectionManager().shutdown();
    return null;
}

// Used for POST REST calls
// Not complete since e.g. uploading a file would need the body to contain the file, not an xml string
private synchronized String postServerREST(String postCall, String xmlString) {
    DefaultHttpClient httpClient = getConnection();
    HttpPost httpPost = new HttpPost(postCall);
    try {
        StringEntity entity = new StringEntity(xmlString, "UTF-8" );
        entity.setContentType("text/plain");
        FileBody file = new FileBody(new File("bird.jpg"));
        StringBody comment = new StringBody("A nice bird");
        MultipartEntity reqEntity = new MultipartEntity();
        reqEntity.addPart("bin", file);
        reqEntity.addPart("comment", comment);
        httpPost.setEntity(entity);
        System.out.println("Request Line:");
        System.out.println(httpPost.getRequestLine().toString() + "\n\n");
        HttpResponse response = httpClient.execute(httpPost);
        // Call to getResponse needs to be done before httpClient is shut down
        // otherwise the response disappeared and null is returned
        String stringResponse = getResponse(response); // not a thread safe
        return stringResponse;
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    httpClient.getConnectionManager().shutdown();
    return null;
}

private synchronized String postServerREST2(String postCall, String xmlString) {
    DefaultHttpClient httpClient = getConnection();
    HttpPost httpPost = new HttpPost(postCall);
    try {
        StringEntity entity = new StringEntity(xmlString, "UTF-8" );
        entity.setContentType("text/plain");
        FileBody file = new FileBody(new File("bird.jpg"));
        StringBody comment = new StringBody("A nice bird");
        MultipartEntity reqEntity = new MultipartEntity();
        reqEntity.addPart("bin", file);
        reqEntity.addPart("comment", comment);
        httpPost.setEntity(entity);
        System.out.println("Request Line:");
        System.out.println(httpPost.getRequestLine().toString() + "\n\n");
        HttpResponse response = httpClient.execute(httpPost);
        // Call to getResponse needs to be done before httpClient is shut down
        // otherwise the response disappeared and null is returned
        String stringResponse = getResponse(response); // not a thread safe
        return stringResponse;
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    httpClient.getConnectionManager().shutdown();
    return null;
}

// Used for GET REST calls
private synchronized String getServerREST(String getCall) {
    DefaultHttpClient httpClient = getConnection();
   HttpGet httpGet = new HttpGet(getCall);
    try {
        System.out.println("Request Line: ");
        System.out.println(httpGet.getRequestLine().toString() + "\n\n");
        HttpResponse response = httpClient.execute(httpGet);
        // Call to getResponse needs to be done before httpClient is shut down
        // otherwise the response disappeared and null is returned
        String stringResponse = getResponse(response); // not a thread safe
        return stringResponse;
    } catch (Exception e) {
        System.out.println(e.getMessage());
    }
    httpClient.getConnectionManager().shutdown();
    return null;
String stringResponse = getResponse(response);
httpClient.getConnectionManager().shutdown();
return stringResponse;

} catch (IOException e) {
    System.out.println(e.getMessage());
    httpClient.getConnectionManager().shutdown();
    return null;
} catch (Exception e) {
    System.out.println(e.getMessage());
    httpClient.getConnectionManager().shutdown();
    return null;
}
In this example the user with the username hakelo is logged in. In the return message we can retrieve the users user id, homefolder id and drive id. You can get other information like the firstname and lastname of the home folder, you need a folder to start from to get information about other folders.

Example

In this example the user with the username hakelo is logged in. In the return message we can retrieve the users user id, homefolder id and drive id. You can get other information like the firstname and lastname of the user, this information is however not that important for other api calls.

Outgoing message

Example

In this example the user with the username hakelo is logged in. In the return message we can retrieve the users user id, homefolder id and drive id. You can get other information like the firstname and lastname of the user, this information is however not that important for other api calls.

Incoming answer
**Summary**

Get Account Information: **GET** [http://os.cloudme.com/v1/users/"userid"/account](http://os.cloudme.com/v1/users/"userid"/account)
Retrieves account information about a user with id "userid".
Account information consist of:
- **balance**: The user's balance.
- **type**: Free or paid account?
- **state**: State of the account
- **freeSize**: Free space available

**Example**

In this example we fetch the account information for the user with id 32348765

**Outgoing message**

```plaintext
GET https://os.cloudme.com/v1/users/"32348765"/account HTTP/1.1
```

**Incoming answer**

```
<account>
  <balance>0</balance>
  <type>FREE</type>
  <state>NORM</state>
  <freeSize>3221225472</freeSize>
</account>
```

---

Add Labels
Add Comment

Powered by [Atlassian Confluence](https://confluence.atlassian.com), the Enterprise Wiki.

Bug/feature request – Atlassian news – Contact administrators
Summary
getDriveInfo: [driveId]
Retrieves information about a drive with the id driveId

Example
In this example we want information about the drive with id 30067187875

Outgoing message
```xml
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <getDriveInfo>
            <drive id='30067187875'/>
        </getDriveInfo>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer
```xml
    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <xcr:getDriveInfoResponse xmlns:xcr="http://xcerion.com/xcRepository.xsd">
            <drive>
                <driveId>30067187875</driveId>
                <ownerId>12885514210</ownerId>
                <folderId>562958546675878</folderId>
                <sysname/>
                <name/>
                <system>home</system>
                <currentSize>2439768</currentSize>
                <quotaLimit>3221225472</quotaLimit>
                <maxFileSize>157286400</maxFileSize>
                <readLocked>false</readLocked>
                <writeLocked>false</writeLocked>
                <created>2012-09-18T15:52:11Z</created>
            </drive>
        </xcr:getDriveInfoResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
## Folders (REST)

### Summary

Folders: GET [http://os.cloudme.com/v1/folders/"folderid"]

Returns the subfolders of the folder with id folderid

### Example

Returns the subfolders for the folder with id 456445

### Outgoing message

```
GET http://os.cloudme.com/v1/folders/456445 HTTP/1.1
```

### Incoming answer

```
  <fs:folder id='562958547009387' name='Music'/>
  <fs:folder id='562958547009388' name='CloudMe'/>
  <fs:folder id='562958547009389' name='Pictures' listview='apps/system/listviews/documents_thumbnails.xsl'/>
</fs:folder>
```
Summary

deleteFolder: ["folderId", "childFolderId"]
Deletes a folder with id folderId, or its child if you supply childFolderId.

Example

In this example we want to delete the folder with id 562958546913094 which lies in the folder with id 562958546675878. The answer will tell you if the deletion was OK or not.

Outgoing message

  <SOAP-ENV:Body>
    <deleteFolder>
      <folder id="562958546675878"/>
      <childFolder id="562958546913094"/> //optional, if not given, the whole parent folder will be deleted
    </deleteFolder>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Incoming answer

  <SOAP-ENV:Body>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
getFolderACL

Summary

getFolderACL: ["folderId", "childFolderId"]
Retrieve the acl(access control list) on a folder with id folderId or its child if you supply childFolderId

Example

In this example we want to get the acl of the folder with id 562958546886038 that lies in the folder with id 562958546675878
In the answer we get the acl back, apparently the user with id 12885514210 is allowed to read, delete and change acl but not allowed to write.

Outgoing message

```xml
                   xmlns:SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <getFolderACL>
      <folder id='562958546675878'/> //id of folder
      <childFolder id='562958546886038'/> //optional id of child folder
    </getFolderACL>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
                   xmlns:SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
                   xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <xcr:getFolderACLResponse xmlns:xcr="http://xcerion.com/xcRepository.xsd">
      <acl tag='Ov9axeQk3q/6DX4tgfN2Q7OHZ58'> //acl tag
        <user id='12885514210' read='1' write='0' delete='1' changeACL='1' /> // acl properties
      </acl>
    </xcr:getFolderACLResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
In this example we want to retrieve the sub folder structure of the folder with id 562958546675878 which is the home folder of this user.

```
<?xml version="1.0" encoding="UTF-8"?>
  <SOAP-ENV:Body>
    <xcr:getFolderXMLResponse xmlns:xcr="http://xcerion.com/xcRepository.xsd"> // id of parent folder
      <xcr:getFolderXML id="562958546675878" name="Profile" type="folder"/>
      <xcr:getFolderXML id="562958546675882" name="CloudMe" type="folder"/>
      <xcr:getFolderXML id="562958546675883" name="Desktop" type="folder"/>
      <xcr:getFolderXML id="562958546675884" name="Documents" type="folder"/>
      <xcr:getFolderXML id="562958546675886" name="Music" type="folder"/>
      <xcr:getFolderXML id="562958546675887" name="Pictures" type="folder"/>
      <xcr:getFolderXML id="562958546675888" name="Trashcan" type="folder"/>
      <xcr:getFolderXML id="562958546885065" name="test" type="folder"/>
      <xcr:getFolderXML id="562958546885898" name="Applications" type="folder"/>
      <xcr:getFolderXML id="562958546885899" name="Data" type="folder"/>
      <xcr:getFolderXML id="562958546886038" name="Files" type="folder"/>
      <xcr:getFolderXML id="562958546886217" name="Guestbook" type="folder"/>
      <xcr:getFolderXML id="562958546886218" name="temp" type="folder"/>
      <xcr:getFolderXML id="562958546886220" name="Data" type="folder"/>
      <xcr:getFolderXML id="562958546886222" name="Settings" type="folder"/>
      <xcr:getFolderXML id="562958546886223" name="test" type="folder"/>
    </xcr:getFolderXMLResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
modifyFolder: ["folderId", "childFolderId", "name"]

Sets the name of a folder with id childFolderId of parent folder with id folderId to a new name.

Example

In this example we set the name of folder with id 562958546886038 inside the folder with id 562958546675878 to hololo. The answer will tell us if the operation was OK or not.

Outgoing message

```xml
  <SOAP-ENV:Body>
    <modifyFolder>
      <folder id='562958546675878'/>  //Parent folder id
      <childFolder id='562958546886038'/> //Child folder id
      <name>hololo</name>  // the new name
    </modifyFolder>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
  <SOAP-ENV:Body>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Summary

moveFolder: ["srcFolderId", "srcChildFolderId", "dstFolderId", "name"]

 Moves a folder with id srcChildFolderId with parent folder with id srcFolderId to a new folder with id dstFolderId and gives it the new name. If you don't supply a name, it will be given a random new one.

Example

In this example we move the folder with id 56295854686038 inside the folder 562958546675878 to the folder with id 562958546885898 and gives it the new name hololo. In the answer we can see whether the operation was OK or not.

Outgoing message

 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <moveFolder>
      <srcFolderId>562958546675878</srcFolderId> // parent folder id
      <srcChildFolderId>562958546886038</srcChildFolderId> // id of folder to be moved
      <dstFolderId>562958546885898</dstFolderId> //destination folder
      <name>hololo</name> //optional, if not used a random name will be given
    </moveFolder>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

Incoming answer

 xmlns:xcr="http://xcerion.com/xcRepository.xsd">
  <SOAP-ENV:Body>
    <xcr:moveFolderResponse>OK</xcr:moveFolderResponse> // successfull or not
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
Summary

newFolder:["folderId", "childFolder", "acl"]

Creates a new folder with the name childFolder inside the folder with id folderId. You can optionally supply a non default acl(access control list) as well.

Example

In this example we create a new folder with the name 123 inside the folder with id 562958546675878. We also supply a custom acl.

Outgoing message

```xml
  <SOAP-ENV:Body>
    <newFolder>
      <folder id='562958546675878'/> // Parent folder
      <childFolder id='123'/> // New folder name
      <acl>
        <user id='12885514210' read='1' write='0' delete='1' changeACL='1'/> //optional acl
      </acl>
    </newFolder>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
  <SOAP-ENV:Body>
    <xcr:newFolderResponse xmlns:xcr="http://xcerion.com/xcRepository.xsd">
      <newFolderId id='562958546913094'/> // the new folder id
    </xcr:newFolderResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
setFolderACL ("folderId", "childFolderId", "acl")

Sets the acl to a new acl in a folder with id folderId or its child with id childFolderId
Old acl entries are destroyed when this method is called so use with care.

Example

In this example we set a new acl to the folder with id 562958546886038 inside its parent folder with id 562958546675878
In the answer back we get the tag of the new acl.

Outgoing message

```xml
  <SOAP-ENV:Body>
    <setFolderACL>
      <folder id='562958546675878'/> //parent folder
      <childFolder id='562958546886038'/> //optional child folder
      <acl tag='kySmyfKl0RItPS2ktHzOtpNzlGc'>
        <user id='12885514210' read='1' write='1' delete='1' changeACL='1'/>
      </acl>
    </setFolderACL>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
  <SOAP-ENV:Body>
      <acl tag='ZLmQicYMXCHPwujVnhk4U1bCwaQ'/> //If succesfull returns the new acl tag
    </xcr:setFolderACLResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Summary

It's also possible to use the optional parameter dl=filename to set a name for the downloaded file e.g. https://www.cloudme.com/v1/documents/529854645000?440034058/Halmstad.jpg?dl=Halmstad.jpg

Example

Downloads a text file with id 786576, that lies in the folder with id 456546

Outgoing message

GET https://os.cloudme.com/v1/documents/529854645000/786576 HTTP/1.1

Incoming answer

This is a textfile,edited

Outgoing message

GET https://os.cloudme.com/v1/documents/456546/786576 HTTP/1.1

Incoming answer

This is a textfile,edited

Powered by Atlassian Confluence 2.9.1, the Enterprise Wiki.
Bug/feature request – Atlassian news – Contact administrators
**Upload File:** POST [http://os.cloudme.com/v1/documents/"folderId"]

You need to attach the file to the message body. For example in java with the apache http client you can attach a as follow:

```java
FileBody file = new FileBody(new File("bird.jpg"));
StringBody comment = new StringBody("A nice bird");
MultipartEntity reqEntity = new MultipartEntity();
reqEntity.addPart("bin", file);
reqEntity.addPart("comment", comment);
httpPost.setEntity(reqEntity);
```

**Example**

In this example we upload a file to the folder with id 45458454.

In the reply from the server you get useful information about the file you uploaded, like its document id.

**Outgoing message**

POST [http://os.cloudme.com/v1/documents/4555466 45557/1.1]

**Incoming answer**

```xml
  <atom:title>bird.jpg</atom:title>
  <atom:link rel="alternate" type="image/jpeg" href="http://os.cloudme.com/v1/documents/562958546675878/4398763170/1" length="63057"/>
  <atom:id>mid:1062fcca2@xios.xcerion.com</atom:id>
  <dc:folder>562958546675878</dc:folder>
  <dc:document>4398763170</dc:document>
</atom:entry>
```
Summary

copyDocument: ["srcFolderId", "srcDocumentId", "srcDocument", "dstFolderId", "dstDocument"]

Example

In this example we copy a file with the name Sunset.jpg in a folder with id 562958546675878 to the same folder and give the copy the name sunset_copy.jpg. The answer contains various metadata, the most important are probably <dc:document> as its the document id of the new copy of the file.

Outgoing message

```xml
 xmlns:xsd="http://www.w3.org/1999/XMLSchema"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xcr="http://xcerion.com/xcRepository.xsd">
  <SOAP-ENV:Body>
    <xcr:copyDocument>
      <srcFolder id='562958546675878'/> // id of the folder the file exist in
      <srcDocument>Sunset.jpg</srcDocument> // you can use document id overall i.e <srcDocument id='36546465654'/>
      <dstFolder id='562958546675878'/> // destination folder
      <dstDocument>sunset_copy.jpg</dstDocument> // name of the copy
    </xcr:copyDocument>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
 xmlns:xsd="http://www.w3.org/1999/XMLSchema"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xcr="http://xcerion.com/xcRepository.xsd">
  <SOAP-ENV:Body>
    <xcr:copyDocumentResponse>
      <atom:entry xmlns:atom="http://www.w3.org/2005/Atom"
                  xmlns:os="http://a9.com/-/spec/opensearch/1.1/
                  xmlns:dc="http://xcerion.com/directory.xsd"
                  xmlns:ni="http://xcerion.com/noindex.xsd">
        <atom:title>sunset_copy.jpg</atom:title> // name of the file
        <atom:link rel='alternate' type='image/jpeg' href='http://os.cloudme.com/v1/documents/562958546675878/4398684297/1' length='318203'/> // direct link to file
        <atom:id>mid:1062e9889@xios.xcerion.com</atom:id>
        <dc:folder>562958546675878</dc:folder> // folder the file is located in
        <dc:document>4398684297</dc:document> //document id
        <dc:width>800</dc:width> // width of jpg picture
        <dc:height>600</dc:height> //height of jpg picture
        <dc:thumbnail>32769</dc:thumbnail> //dpi of picture
        <dc:dpi>96x96</dc:dpi>
      </atom:entry>
    </xcr:copyDocumentResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Summary
createDocument(["folderId", "acl", "overwrite", "entry"])

Creates a document in a folder with the id folderId. The entry specifies the properties of document. The most basic entry would only specify the name, like in the example below.

overwrite determines if you should overwrite if a document with the same already exists. The default value for overwrite is false. acl allows you to give the document a specified access control list, as shown in the example below is optional.

Example
In this example we create a new document in the folder with id 562958546675878. We give it a custom acl and set overwrite to true. We also provide a simple entry with only the title of the new document. In the answer we get an entry which contains information about the create document, the most important being dc:document which is the document id.

Outgoing message

Incoming answer
Summary

deleteDocument:
["folderId", "documentId", "document"]

Deletes a document with id documentId or name document in the folder with id folderId. The answer from the server will only respond with OK if the document were deleted as intended.

Example

In the example we delete the file hello.txt in the folder 562958546675878. In the answer from the server we can see that the operation was OK.

Outgoing output

```xml
  <SOAP-ENV:Body>
    <deleteDocument>
      <folder id="562958546675878"/>
      <document>hello.txt</document>
    </deleteDocument>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
  <SOAP-ENV:Body>
    <xcr:deleteDocumentResponse>
      OK
    </xcr:deleteDocumentResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
**Summary**

`getDocumentACL(folderId, documentId)`

Gets the access control list of a document with id documentId or name document in the folder with id folderId.

In the answer from the server you get the access control list with its defining tag.

**Example**

In this example we want to get the acl of the file Sunset.jpg in the folder with id 562958146575878.

From the server we get the answer which contains an acl with the tag BfiU/Vkz980Y8xsetQYAYFMvR5s.

Apparently the user with id 12885514210 have full control over the file.

**Outgoing message**

```xml
  <SOAP-ENV:Body>
    <getDocumentACL>
      <folder id='562958146575878'/> //id of the folder where the file exist
      <document>Sunset.jpg</document> // you can use document id aswell i.e <document id='36546465654'/>
    </getDocumentACL>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

**Incoming answer**

```xml
  <SOAP-ENV:Body>
      <acl tag='BfiU/Vkz980Y8xsetQYAYFMvR5s'>
        <user id='12885514210' read='1' write='1' delete='1' changeACL='1'/> // acl of the file
      </acl>
    </xcr:getDocumentACLResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Summary

loadMetadata: ["folderId", "documentId", "document"]

Retrieves metadata (describing information) about a document with the id documentId or name document in the folder with id folderId.

Metadata between different files and file types will differ, a text file won't have the attribute height like a jpg would for example.

Example

In this example we want to load the metadata of the file bird.jpg in the folder with id 562958546675878.

In the answer we get an entry with the files metadata.

Outgoing message

```
  <SOAP-ENV:Body>
    <loadMetadata>
      <folder id='562958546675878' />
      <document>bird.jpg</document>
    </loadMetadata>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```
  <SOAP-ENV:Body>
    <xcr:loadMetadataResponse>
      <atom:entry>
        <atom:title>bird.jpg</atom:title>
        <atom:updated>2012-10-23T07:49:34Z</atom:updated>
        <atom:link rel='alternate' type='image/jpeg' href='http://os.cloudme.com/v1/documents/562958546675878/4398763170/1' length='262341'/>
        <atom:id>mid:1062fcca2@xios.xcerion.com</atom:id>
        <dc:folder>562958546675878</dc:folder>
        <dc:document>4398763170</dc:document>
        <dc:width>600</dc:width>
        <dc:height>500</dc:height>
        <dc:thumbnail>32769</dc:thumbnail>
        <atom:summary>Ducky<atom:summary>
        <dc:dpi>100x100</dc:dpi>
        <dc:businesscompanyname/>
        <atom:author/>
      </atom:entry>
    </xcr:loadMetadataResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
moveDocument(["srcFolderId", "srcDocumentId", "srcDocument", "dstFolderId", "dstDocument"])
Moves a document with id srcDocumentId or name srcDocument in the folder with id srcFolderId to another folder with id dstFolderId and gives it the new name dstDocument.

Example
In this example we move the document Red.jpg that resides in the folder with id 562958546675878 to the same folder and give it the new name moved_file.jpg

Outgoing message

```xml
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <moveDocument>
      <srcFolderId>'562958546675878'</srcFolderId>
      <srcDocumentId>'RED.jpg'</srcDocumentId>
      <dstFolderId>'562958546675878'</dstFolderId>
      <dstDocumentId>'moved_file.jpg'</dstDocumentId>
    </moveDocument>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
    xmlns:xcr="http://xcerion.com/xcRepository.xsd">
  <xcr:moveDocumentResponse>
    <atom:entry>
      <atom:title>'moved_file.jpg'</atom:title>
      <atom:updated>'2012-10-17T16:29:52Z'</atom:updated>
      <atom:link rel='alternate' type='image/jpeg' href='http://os.cloudme.com/v1/documents/562958546675878/4398620818/1'/>
      <atom:id>mid:1062da092@xios.xcerion.com</atom:id>
      <dc:folder>'562958546675878'</dc:folder>
      <dc:document>'4398620818'</dc:document>
      <dc:width>'800'</dc:width>
      <dc:height>'600'</dc:height>
      <dc:thumbnail>'32769'</dc:thumbnail>
      <dc:dpi>'96x96'</dc:dpi>
    </atom:entry>
  </xcr:moveDocumentResponse>
</SOAP-ENV:Envelope>
```
renameDocument: ["folderId", "documentId", "document", "newName"]

Renames a document with id `documentId` or name `document` in the folder with id `folderId` to the new name `newName`.

Example

In this example we rename to file `Sunset.jpg` in the folder with id `562958546675878` to the new name `RED.jpg`.

Outgoing message

```xml
  <SOAP-ENV:Body>
    <renameDocument>
      <folder id='562958546675878'/> // folder id where file is located
      <document>Sunset.jpg</document> // you can use document id manual i.e document id = 562958546675878
      <newName>RED.jpg</newName> // the new name of the file
    </renameDocument>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
  <SOAP-ENV:Body>
        <atom:title>RED.jpg</atom:title>
        <atom:link rel='alternate' type='image/jpeg' href='http://os.cloudme.com/v1/documents/562958546675878/4398620818/1' length='318203'/> // direct link to file
        <atom:id>mid:1062da092@xios.xcerion.com</atom:id>
        <dc:folder>562958546675878</dc:folder>
        <dc:document>4398620818</dc:document>
        <dc:width>800</dc:width>
        <dc:height>600</dc:height>
        <dc:thumbnail>32769</dc:thumbnail>
        <dc:dpi>96x96</dc:dpi>
      </atom:entry>
    </xcr:renameDocumentResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Summary

setDocumentACL: ["folderId", "documentId", "acl", "document"]

This call should be used with care, as old ACL entries will be removed when you set a new one.

Example

In this example we set the acl of the document "sunset.jpg" to a new one, which gives the user with id 1288514210 the right to write, delete and change acl of the file, but disallows reading of the file.

Outgoing message

```xml
  <SOAP-ENV:Body>
    <setDocumentACL>
      <folder id='562958546675878'/> // folder where file exist
      <acl tag='BfiU/Vkz980Y8xsetQYAYFMvR5s'>
        <user id='1288514210' read='0' write='1' delete='1' changeACL='1'/> // the new acl
      </acl>
      <document>"sunset.jpg"</document> // name of the file, you can use document id aswell i.e <document id='36546465654'/>
    </setDocumentACL>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
  <SOAP-ENV:Body>
    <xcr:setDocumentACLResponse>
      < acl tag='KDPzIsz0II/uaUn5eyTc4DonsaU'/> // tag of the new acl
    </xcr:setDocumentACLResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Summary

Create Webshare: POST http://os.cloudme.com/v1/users/"userId"/webshares
Creates a new webshare for the user with id userId

Example

We create a new webshare for the user with id 15151241

Outgoing message

POST http://os.cloudme.com/v1/users/15151241/webshares HTTP/1.1

And attach the webshare message in the body, an example is

```xml
<webshare name="webshareName" password="password" description="description">
  <folder id="562958546675878" name="hej"/>
</webshare>
```

Incoming answer

```xml
<webshare id="646307" userId="12885514210" name="webshareName" description="description" visibility="public" password="password" type="" created="2012-10-22T10:08:34Z" updated="2012-10-22T10:08:34Z">
  <folder name="hej" id="562958546675878"/>
</webshare>
```
Change Webshare

Summary

Change Webshare: POST http://os.cloudme.com/v1/users/"userId"/webshares/"webshareId"
Changes a webshare with the id webshareId owned by the user with id userId

Example

In this example we change the webshare with id 648548 owned by the user with id 12885514210

Outgoing message

POST http://os.cloudme.com/v1/users/12885514210/webshares/648548 HTTP/1.1
//where userid is the owner of the share and webshare id is the id of the webshare

You attach the changes you want to make to the message body, for example

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>webshare</td>
<td>name=’mina filer’ password=’rmf’ description=’my files’ visibility=’public’</td>
</tr>
<tr>
<td>folder</td>
<td>id=’562958546675878’ name=’hej’</td>
</tr>
</tbody>
</table>

Incoming answer

null
null
Delete webshare

Summary

Delete Webshare: DELETE http://os.cloudme.com/v1/users/"userId"/webshares/"webshareId"
Deletes a webshare with the id webshareId owned by user with the id userId

Example

In this example we deleted the webshare with id 45456456 owned by the user with id 45161616

Outgoing message

DELETE http://os.cloudme.com/v1/users/45161616/webshares/45456456 id" HTTP/1.1

Incoming answer

null
null
Summary

getLocale
Get the localization setting for the user

Example

In this example we want to get the localization of the user. In the answer we can see that the localization is en_US.

Outgoing message

```xml
  <SOAP-ENV:Body>
    <getLocale/>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
  <SOAP-ENV:Body>
    <xcr:getLocaleResponse xmlns:xcr="http://xcerion.com/xcRepository.xsd">
      <locale>en_US</locale>
    </xcr:getLocaleResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Summary

setLocale ["locale"]
Changes the localization to the new localization code locale

Example

In this example we set the locale to es-Mx which is Spanish-Mexican.

Outgoing message

```xml
 xmlns:SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <setLocale>
      <locale>es-MX</locale>
    </setLocale>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
 xmlns:SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xcr:="http://xcerion.com/xcRepository.xsd">
  <SOAP-ENV:Body>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
Summary
createPorts: ["folderListener", "documentListener", "userListener" "driveListener" "favoriteListener"]

Creates listeners on a folder and/or document and/or user and/or drive. You get a link back where information about an event will be sent if it occurs. You can listen to this link with REST GET. Can for example be used on a drive to listen to all folder and document changes. Some examples of events are, addFolder, removeFolder, addEntry, removeEntry, addEntry would be sent if you uploaded a new file.

Example
Creates a folderListener on the folder with id 562958546675878, a documentListener on bird.jpg document in folder with id 562958546675878 and an userListener of userid 12885514210. You get a reply back with an URL that you can listen on for changes with REST GET.

Outgoing message

Incoming answer
Summary

sendMessage: ["userId", "senderId", "message"]

Sends a message to the user with id userId from the user with id senderId. In the answer we can see whether the operation was "OK" or not.

Example

In this example the user with id 12885514210 sends "hello" to the user with id 12885514211

Outgoing message

```xml
  <SOAP-ENV:Body>
    <sendMessage>
      <user id="12885514211"/>
      <sender id="12885514210"/>
      <message>hello</message>
    </sendMessage>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Incoming answer

```xml
  <SOAP-ENV:Body>
    <xcr:sendMessageResponse>OK</xcr:sendMessageResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© Hampus Malmkvist