CloudMe Java API

Revision 1.0.1, 2013-09-20

1. Tutorials

2. References

3. Get started

Tutorials

Create a new folder and upload a file

To begin with we have the following code to login

```
import Cloudme.CloudmeException;
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.killUser();

   }

}
```

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

```
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

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

Upload it via CloudmeFileManager

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

Full example

```
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");
     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.

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

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

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

Lets save the data to CloudMe's server

```
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

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

Full example

```
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();
   }

}
```

File and Folder browser

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.

```
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.

```
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

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

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

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

Full example

```
import java.util.ArrayList;

import Cloudme.CloudmeException;
import Cloudme.CloudmeFile;
import Cloudme.CloudmeFolder;
import Cloudme.CloudmeFolderNode;
import Cloudme.CloudmeUser;


public class cloudmetest {

   /**

    * @param args

    * @throws CloudmeException

    */


   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();
  }

}
```

File search

Lets say we want to find all jpg files on the hard drive, then we can use CloudmeFileManagers 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.

```
ArrayList<CloudmeFile> files =  user.getFileManager().FileSearch("/", "jpg", true);
     for(CloudmeFile f :files){

        System.out.println(f.getMetadata().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.

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

Should print something like this

```
<atom:title>Notepad.xml</atom:title><atom:published>2012-11-
14T10:47:08Z</atom:published><atom:updated>2012-11-14T10:47:08Z</atom:updated><atom:link re
l='alternate' type='text/xml' href='http://os.cloudme.com/v1/documents/562958547511009/4400003849
/1' length='93'/><atom:id>mid:10642bb09@xios.xcerion.com</atom:id><dc:folder>562958547511009</dc:
folder><dc:document>4400003849</dc:document><dc:root xmlns:dc="http://xcerion.com/directory.xsd">
settings</dc:root>
```

Full example

```
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");
    ArrayList<CloudmeFile> files =  user.getFileManager().FileSearch("/", "jpg", true);
    for(CloudmeFile f :files){

      System.out.println(f.getMetadata().getName());

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

}
```

Filelistener

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

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

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

```
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 a example of a class implementing the CloudmeFileListener interface that could be added as a file listener.

```
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

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

Getting all keys for some domain and its subdomains

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

Returns

```
<custom><game1 var='5'/></custom>
```

Getting a specific key

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

Returns

```
5
```

Full example

```
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();

   }

}
```

References

Classdiagram

https://www.cloudme.com/api/CloudMeClassUML.gif

Classes

CloudmeFile

This class represent a file on CloudMe

Methods

---

void **deleteFile**()

void **readFile**()

void **readStream**(String number)

void **downloadStream**(String number,String destpath)

ArrayList<String> **getAvailableStreams**()

void **downloadFile**(String destpath)

CloudmeFile **copyFile**(String destpath)

CloudmeFile **copyFile**(CloudmeFolder folder, String name)

void **moveFile**(String destpath,String name)

void **moveFile**(CloudmeFolder folder,String name)

void **renameFile**(String newname)

CloudmeMetadata **getMetadata**()

void **setMetadata**(CloudmeMetadata metadata)

void **saveMetadata**()

void **saveFile**()

void **copyStream**(String sourcestreamnumber,String destpath,String deststreamnumber)

void **deleteStream**(String sourcenumber)

org.w3c.dom.Document **getXmlDocument**()

byte[] **getData**()

void **setData**(byte[] data)

void **setData**(org.w3c.dom.Document document)

void **addListener**(CloudmeFileListener listener)

ArrayList<CloudmeStreamData> **getLoadedStreams**()

Method detail

---

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 files data into this CloudmeFile object

Throws:

CloudmeException

void readStream(String number)

Reads a streams data into a CloudmeStream object that is a member in this CloudmeFile object

Parameters:

number: The streams number

void downloadStream(String number,String destpath)

Download the data from a given stream that have been read into local memory to a destination on the local hard drive

Parameters:

number: The streams number
destpath: Path on the local hard drive where the stream should be downloaded.

Returns:

Throws:

CloudmeException

ArrayList<String> getAvailableStreams()

Returns the available streams for this file

Returns:

Returns an ArrayList of String that contain all available stream numbers

Throws:

CloudmeException

void downloadFile(String destpath)

Downloads this files data to the local hard drive

Parameters:

destpath: Path on the local hard drive where the file should downloaded

Throws:

CloudmeException

CloudmeFile 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

CloudmeFile 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

CloudmeFile moveFile(String destpath)

Moves this file to another location

Parameters:

destpath: Path to the new location

Throws:

CloudmeException

CloudmeFile 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 the 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 whese you wish to copy the stream
deststreamnumber: Copy the stream to this number

Throws:

CloudmeException

void deleteStream(String sourcenumber)

Deletes a stream on this file

Parameters:

sourcenumber: The streams number

Throws:

CloudmeException

org.w3c.dom.Document 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

byte[] getData()

Return the local data of this file

Returns:

Returns a byte array of this files local data.

void setData(byte[] data)

Sets the local data to a new value

Parameters:

data: The new value

Throws:

CloudmeException

void 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

void addListener(CloudmeFileListener listener)

Adds a file listener to this file

Parameters:

listener: A class that implements the CloudmeFileListener interface

Throws:

CloudmeException

ArrayList<CloudmeStreamData> 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

byte[] **readFile**(String path)

void **downloadFile**(String sourcepath,String destpath)

byte[] **readStream**(String path,String number)

void **downloadStream**(String path,String number,String destpath)

ArrayList<String> **getAvailableStreams**(String path)

void **copyStream**(String sourcepath,String sourcenumber,String destpath,String destnumber)

void **deleteStream**(String path,String sourcenumber)

CloudmeFile **createDocument**(String path)

void **deleteFile**(String path)

CloudmeFile **uploadFile**(CloudmeUser user,String sourcepath,String destpath)

CloudmeFile **simpleUniqueFileSearch**(String path,String query,boolean recursive)

ArrayList<CloudmeFile> **FileSearch**(String path,String query,boolean recursive)

ArrayList<CloudmeFile> **getFiles**(String path)

CloudmeFile **getFile**(String path)

CloudmeFile **copyFile**(String srcpath,String destpath)

CloudmeFile **copyFile**(String srcpath,CloudmeFolder folder,String name)

CloudmeFile **moveFile**(String srcpath,String destpath)

CloudmeFile **moveFile**(String srcpath,CloudmeFolder folder,String name)

void **renameFile**(String path,String newname)

Method detail

---

byte[] 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

void 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

byte[] 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

void 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

ArrayList<String> 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 files stream

Parameters:

sourcepath: Path to the file where the stream you wish to copy is located
sourcenumber: The streams 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 streams 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 represent 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 after 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 = dont search subfolders;

Returns:

Returns a CloudmeFile object that represents the found file.

Throws:

CloudmeException

ArrayList<CloudmeFile> FileSearch(String path,String query,boolean recursive)

Searches after 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 = dont 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:

CloudmeException

CloudmeFile getFile(String path)

Returns a file from a specified location

Parameters:

path: The path to the file

Returns:

Returns a CloudmeFile object that represent the file fetched

Throws:

CloudmeException

CloudmeFile copyFile(String srcpath,String destpath)

Copies a file to another location

Parameters:

srcpath: Path to the file you wish to copy
destpath: Path where you wish to store the copy

Returns:

Returns a CloudmeFile object that represents the copy of the file

Throws:

CloudmeException

CloudmeFile copyFile(String srcpath,CloudmeFolder folder,String name)

Copies a file to another location

Parameters:

srcpath: Path to the file you wish to copy
folder: CloudmeFolder where you want to store the copy
name: The name you give the copy of the file

Returns:

Returns a CloudmeFile object that represents the copy of the file

Throws:

CloudmeException

CloudmeFile moveFile(String srcpath,String destpath)

Moves a file to another location

Parameters:

srcpath: Path to the file you wish to move
destpath: Path where you wish to move the file

Returns:

Returns a CloudmeFile object that represents the moved file

Throws:

CloudmeException

CloudmeFile moveFile(String srcpath,CloudmeFolder folder,String name)

Moves a file to another location

Parameters:

srcpath: Path to the file you wish to move
folder: CloudmeFolder where you want to move the file.
name: The name you give the moved file

Returns:

Returns a CloudmeFile object that represents the moved file

Throws:

CloudmeException

void renameFile(String path,String newname)

Renames a file

Parameters:

path: Path to the file you wish to rename

Throws:

CloudmeException

CloudmeFolder

This class represent a folder on CloudMe

Methods

String **getName**()

String **getId**()

String **getParentid**()

CloudmeFolder **getParentfolder**()

void **changeFolderName**(String newname)

ArrayList<CloudmeFolder> **getChildFolders**()

CloudmeFolder **newFolder**(String foldername)

void **deleteFolder**()

void **moveFolder**(String destpath)

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

CloudmeFile **simpleUniqueFileSearch**(String query, boolean recursive)

ArrayList<CloudmeFile> **FileSearch**(String query, boolean recursive)

ArrayList<CloudmeFile> **getFiles**()

CloudmeFile **uploadFile**(String sourcepath)

CloudmeFile **getFile**(String name)

String **getPath**()

void **addListener**(CloudmeFolderListener listener)

Method detail

String getName()

Returns the name of the folder

Returns:

Returns a string with the folder name

String getId()

Returns the id of the folder.

Returns:

Returns a string with the folder id.

String getParentid()

Returns the id of the folders parent

Returns:

Returns a string with the folders parent id

CloudmeFolder getParentfolder()

Returns the parent of this folder

Returns:

Returns a CloudmeFolder object that represent the parent

Throws:

CloudmeException

void changeFolderName(String newname)

Changes the name of this folder

Parameters:

newname: The new name of the folder

Throws:

CloudmeException

ArrayList<CloudmeFolder> getChildFolders()

Returns all first level subfolders of this folder

Returns:

Returns an ArrayList of CloudeFolder objects that represent the subfolders

Throws:

CloudmeException

CloudmeFolder newFolder(String foldername)

Creates a new folder in this folder

Parameters:

foldername: Name of the new folder

Returns:

Returns a CloudmeFolder object that represent the new folder

Throws:

CloudmeException

void deleteFolder()

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.

Throws:

CloudmeException

void moveFolder(String destpath)

Moves this folder, it is not safe to use this folder object after,requires handling by a concrete folder listener

Parameters:

destpath: Path to the place where you wish to move the folder

Throws:

CloudmeException

void moveFolder(CloudmeFolder destfolder,String name)

Moves this folder, it is not safe to use this folder object after,requires handling by a concrete folder listener

Parameters:

destfolder: The CloudeMeFolder where you want to move this folder
name: The name the folder gets after being moved.

Throws:

CloudmeException

CloudmeFile simpleUniqueFileSearch(String query, boolean recursive)

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 "myfile.jpg"
recursive: Boolean, true = search subfolders;false = dont search subfolders;

Returns:

Returns a CloudmeFile object that represents the found file.

Throws:

CloudmeException

ArrayList<CloudmeFile> FileSearch(String query, boolean recursive)

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 "myfile.jpg"
recursive: Boolean, true = search subfolders;false = dont search subfolders;

Returns:

Returns an ArrayList of CloudmeFile that represent all found files

Throws:

CloudmeException

ArrayList<CloudmeFile> getFiles()

Returns all files in this folder

Returns:

Returns an ArrayList of CloudmeFile that represent all found files

Throws:

CloudmeException

CloudmeFile uploadFile(String sourcepath)

Uploads a file to this folder

Parameters:

sourcepath: Path to the file on the local hard drive.

Returns:

Returns a CloudmeFile that represent the uploaded file

Throws:

CloudmeException

CloudmeFile getFile(String name)

Returns the file with name **name**

Parameters:

name: Name of the file

Returns:

Returns a CloudmeFile object that represent the file

Throws:

CloudmeException

String getPath()

Returns the path to this folder

Returns:

Returns a string with the path to this folder.

void addListener(CloudmeFolderListener listener)

Adds a new listener to this folder

Parameters:

listener: 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

CloudmeFolder **newFolder**(String path)

void **changeFolderName**(String srcpath, String newname)

void **deleteFolder**(String path)

CloudmeFolder **getFolder**(String path)

ArrayList<CloudmeFolder> **getFolders**(String path)

void **moveFolder**(String srcpath,String destpath)

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

CloudmeFolderNode **getFolderTree**()

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:

srcpath: 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:

srcpath: Path to the folders old location
destpath: 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:

srcpath: Path to the folders old location
destfolder: CloudmeFolder 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 cant 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

CloudmeFolder **getFolder**()

CloudmeFolder **getFolderWithoutListening**()

ArrayList<CloudmeFolderNode> **getChildren**()

CloudmeFolderNode **getParent**()

void **printTree**()

Method detail

CloudmeFolder getFolder()

Returns the folder of this node.

Returns:

Returns a CloudmeFolder object that represent the folder in this node

CloudmeFolder getFolderWithoutListening()

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

Returns:

Returns a CloudmeFolder object that represent 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()

Return this nodes parent

Returns:

Returns this nodes parent as a CloudmeFolderNode object

void printTree()

Prints the folderstructre with this node as the root folder.

CloudmeMetadata

A class responsible for handling a files metadata. Each CloudmeFile object has a CloudmeMetadata object as a member.

Methods

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:

Returns a found attribute.

Throws:

CloudmeException

String getFolderId()

Returns the folder id of this files parent folder

Returns:

Returns a string with the folder id of this files parent folder

String getMime()

Returns the file type of this file in the MIME standard

Returns:

Returns a string that describes the file type in MIME.

String getName()

Returns the name of this file

Returns:

Returns a string with the file name

String getString()

Returns the whole metadata structure as sent from CloudMe

Returns:

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

byte[] **getData**()

String **getStreamnumber**()

void **setData**(byte[] data)

void **setStreamnumber**(String streamnumber)

Method detail

---

byte[] getData()

Returns the data stored in this stream

Returns:

Returns a byte array containing the streams data.

String getStreamnumber()

Returns the identifying number for this string

Returns:

Returns a string containing this stream identifying number.

void setData(byte[] data)

Sets this streams data to something new

Parameters:

data: A byte array of some new data you want to set this stream to.

void setStreamnumber(String streamnumber)

Changes the identifying number of this stream.

Parameters:

streamnumber: The new identifying number you wish to set this stream to.

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

CloudmeUser(String username, String password)

Methods

String **getAllKeyValues**(String path)

CloudmeFileManager **getFileManager**()

CloudmeFolderManager **getFolderManager**()

CloudmeFolder **getHomeFolder**()

String **getKeyValue**(String path, String key)

void **killUser**()

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

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()

Return 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

This interface should be implemented for a file listener.

Methods

---

void fileChanged(CloudmeFile fileevent, CloudmeFile changes)

void removeThisFile(CloudmeFile fileevent, boolean moved)

CloudmeFolderListener

This interface should be implemented for a folderlistener.

Methods

---

void addFolder(CloudmeFolder folderevent, CloudmeFolder newfolder, boolean moved)

void fileAdded(CloudmeFolder folderevent, CloudmeFile newfile, boolean moved)

void fileChanged(CloudmeFolder folderevent, CloudmeFile changedfile)

void fileRemoved(CloudmeFolder folderevent, CloudmeFile removedfile, boolean moved)

void folderChanged(CloudmeFolder folderevent, CloudmeFolder changes)

void removeChildFolder(CloudmeFolder folderevent, CloudmeFolder removedfolder, boolean moved)

void removeThisFolder(CloudmeFolder folderevent, boolean moved)

Query syntax

Terms

A term is a string you want to search for, or sometimes exclude from the search result. For the sake of simplicity you should always use ' or " around the terms.
You can put a **+** before a term force it to be included in the search query.
You can put a **-** before a term to force it to excluded in the search query.

Examples

```
"cat" "dog" "horse"
```

This would search for files that include **(cat OR dog OR horse)**

```
+"cat" +"dog" "horse"
```

This would search for files that include **(cat AND dog) AND MAYBE (horse)**

```
-"cat" +"dog" "horse"
```

This would search for files that include **(dog) AND NOT (cat) AND MAYBE (horse)**

Prefixed Terms

It is possible to add prefixes on terms to search certain metadata.It should be noted that prefixed terms have an AND relation to terms with another prefix and OR relation to terms with the same prefix.
The syntax for Prefixed terms looks like this: **prefix**:**term**.

Examples

```
-title:"animated" +title:"dog" type:"image/gif"
```

This would search for all gif files that contain dog but doesn't contain animated in the title, **(title:dog AND NOT title:animated) AND (type:image/gif)**

```
title:"animated" title:"dog" -type:"image/gif"
```

This would search for all files that contain animated or dog in the title, except for the gif files, **(title:animated OR title:dog) AND NOT (type:image/gif)**

List of prefixes

Query prefix	Atom entry field	Description
`author`	`atom:entry`	The author's name, email or URI.
`country`	`dc:country`	A two-letter ISO country code (same as used in domain names).
(no prefix)	`atom:content`	Full-text index

document	dc:document	The document ID as a decimal number.
	dc:enddate	This element is not indexed as-is. See below.
flag	dc:flag	A general-purpose flag that can be used by applications.
folder	dc:folder	The ID of the folder the document is located in (a decimal number).
	dc:atom:id	This element is not indexed.
keyword	dc:keyword	Keywords that describe the document.
lang	dc:lang	The language of the document as a two-character ISO code.
month		This prefix is calculated from the `startdate` and `enddate` elements.
namespace	dc:namespace	The document's XML namespace.
	atom:published	This element is not indexed as-is. See below.
pubmonth		This prefix is calculated from the `atom:published` element.
pubweek		This prefix is calculated from the `atom:published` element.
pubyear		This prefix is calculated from the `atom:published` element.
recipient	dc:recipient	The recipient's name, email or URI.
rights	atom:rights	A copyright string.
s1	dc:s1, ni:s1	A general purpose prefix that is also sortable
s1	dc:s2, ni:s2	A general purpose prefix that is also sortable
s2	dc:s3, ni:s3	A general purpose prefix that is also sortable
s3	dc:s4, ni:s4	A general purpose prefix that is also sortable
schema	atom:category type="schema" scheme="..."	The document's W3C schema.
	dc:startdate	This element is not indexed as-is. See below.
summary	atom:summary	A summary of the document's content.
tag	dc:tag	A user-specified tag (like "family", "London", "My wedding" etc).
title	atom:title	A documents title or rather simply the file name.
type	atom:link rel="alternate"	A documents MIME type.

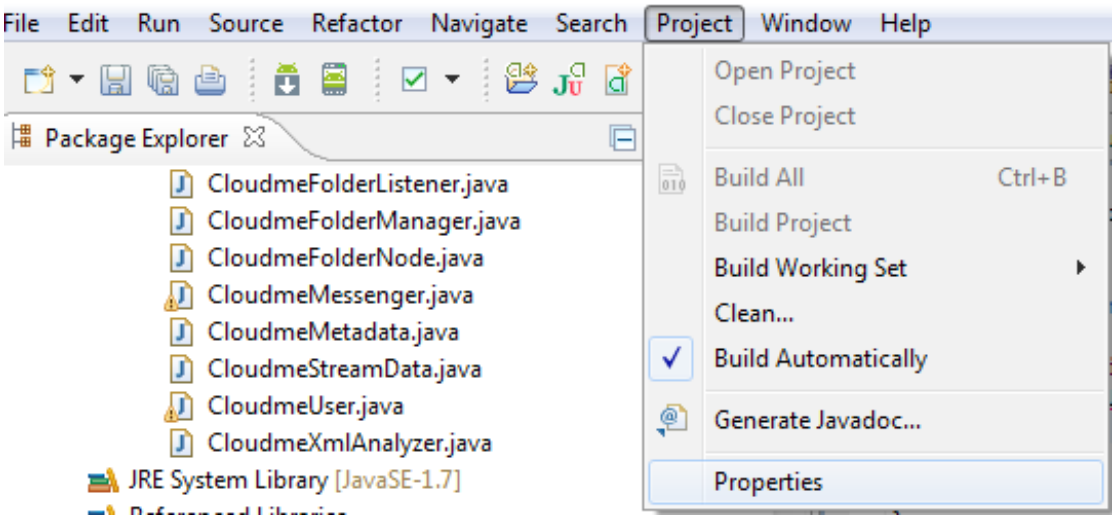
	type="..."	
week		This prefix is calculated from the `startdate` and `enddate` elements.
year		This prefix is calculated from the `startdate` and `enddate` elements.

Get started

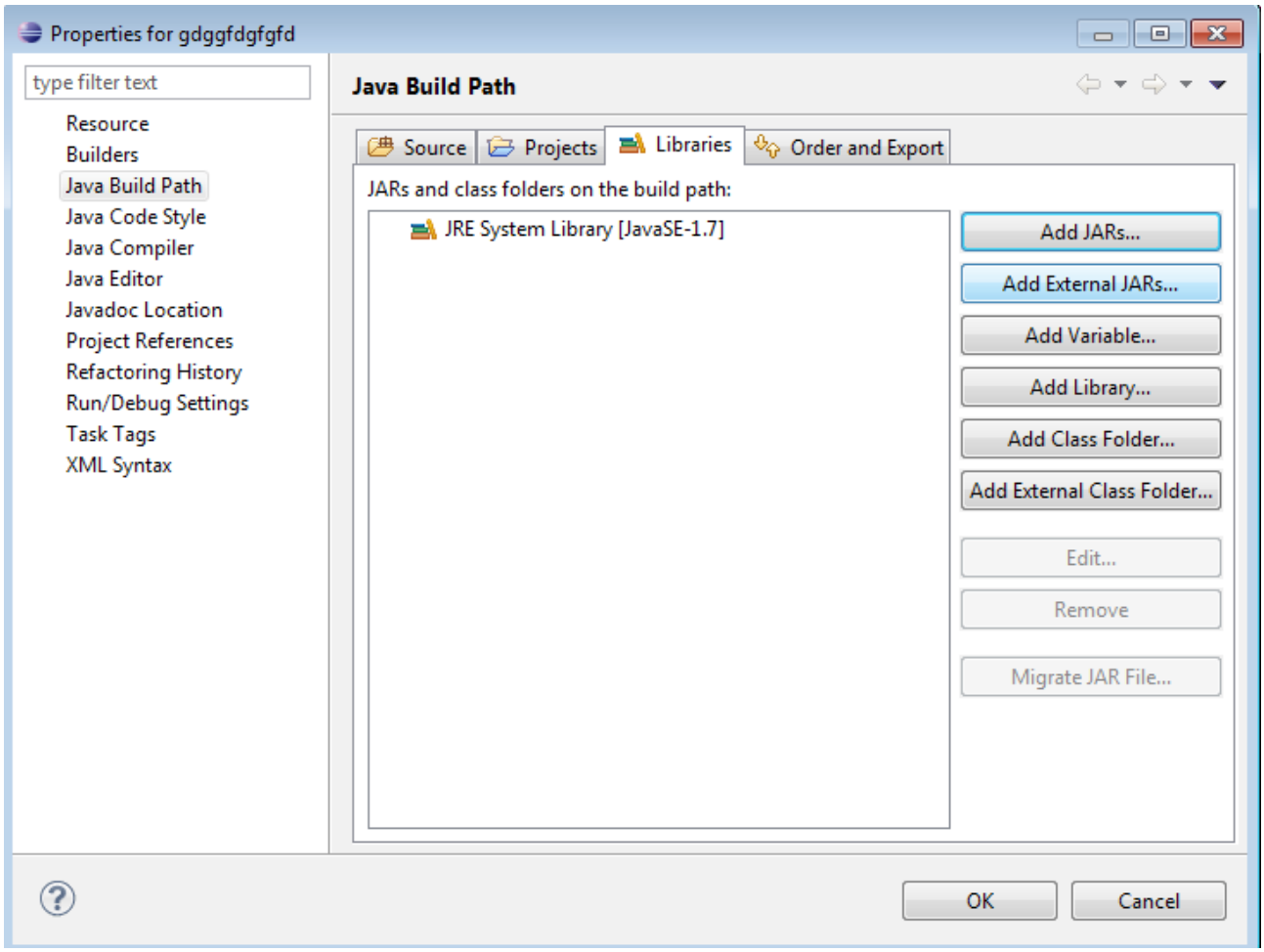
Eclipse

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


File
Edit
Run
Source
Refactor
Navigate
Search
Project
Window
Help
Package Explorer
CloudmeFolderListener.java
CloudmeFolderManager.java
CloudmeFolderNode.java
CloudmeMessenger.java
CloudmeMetadata.java
CloudmeStreamData.java
CloudmeUser.java
CloudmeXmlAnalyzer.java
JRE System Library [JavaSE-1.7]
Open Project
Close Project
Build All
Ctrl+B
Build Project
Build Working Set
Clean...
Build Automatically
Generate Javadoc...
Properties

Java Build Path->Add external JARs



The following should compile if done correctly

```
import Cloudme.CloudmeException;
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.killUser();

   }
```