C-SPACE
An affordable tool to turn the space surrounding us into a creative experience

Deliverable D.3.2

Video Data Extraction Software Module

WP 3 – c-Space mobile client

Task 3.1 – Hybrid user localization and acquisition, tagging and encoding of video streams

Revision: [Final]

Authors:
Angelos Anagnostopoulos (EPSILON)
Anestis Trypitsidis (EPSILON)
Marc Bonazountas (EPSILON)

Dissemination level	PU (public)
Contributor(s)	N.A.
Reviewer(s)	Marc Bonazountas (EPSILON)
Editor(s)	Angelos Anagnostopoulos (EPSILON), Anestis Trypitsidis (EPSILON),Marc Bonazountas (EPSILON)
Partner(s) in charge	EPSILON International
Due date	31/12/2014
Submission Date	14-01-2015

REVISION HISTORY AND STATEMENT OF ORIGINALITY

Revision	Date	Author	Organisation	Description
v.1.0	10/12/2014	Anestis Trypitsidis	EPSILON International	Structure of the document
v.2.0	23/12/2014	Anestis Trypitsidis	EPSILON International	Content inserted
v.2.1	24/12/2014	Angelos Anagnostopoulos	EPSILON International	Content inserted
v.3.0	05/12/2014	Angelos Anagnostopoulos	EPSILON International	Final review
v.4.0	10/01/2015	Marc Bonazountas	EPSILON International	Final review
v.5.0	14/01/2015	Anestis Trypitsidis	EPSILON International	Review and formatting

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Moreover, this deliverable reflects only the author's views. The European Community is not liable for any use that might be made of the information contained herein.

Table of content

Table of figures

Acronyms

API	Application Programming Interface
CAI	Content Access Infrastructure
GIS	Geographic Information System
HTTP	Hypertext Transfer Protocol
J2EE	Java 2 Platform, Enterprise Edition
JSON	JavaScript Object Notation
PostGIS	Spatial extension to PostgreSQL database
PostgreSQL	Object-relational database management system
MVC	Model View Controller
SQL	Structured Query Language
WSDL	Web Service Definition Language

1. Introduction

This document describes the design and implementation process followed for the Video Data Extraction Software Module, used in the context of the c-Space project.

The main idea behind the module is to upload media content (i.e. a video stream or an image) generated by the c-Space end-user mobile application to the CAI (Content Access Infrastructure). Upload is performed using the HTTP protocol and using well known standards. Chunked (i.e. partial) upload is supported, providing 'resume upload' functionality to the mobile application (or any other upload client implemented).

Additionally, meta-data is uploaded along with the media content, in order for the CAI to be able to correctly classify and connect the uploaded material to existing entities in the CAI database.

In the following sections, both the server-side (CAI) and client-side (Android) mechanisms implemented are explained. The technical process followed, in each case, is described, along with the resulting API. Any considerations that must be taken into account are also reported. Finally, future actions are enumerated.

2. Server (CAI) Side

The CAI server has been implemented as a HTTP REST API, using the Spring MVC J2EE framework, which sits on top of a GIS enabled PostgreSQL database (PostGIS – PostgreSQL spatial extension). Hard disk storage space has also been allocated to the CAI, for storing the uploaded media content.

The database tables/entities related to the uploaded media content are depicted in the following entity diagram:

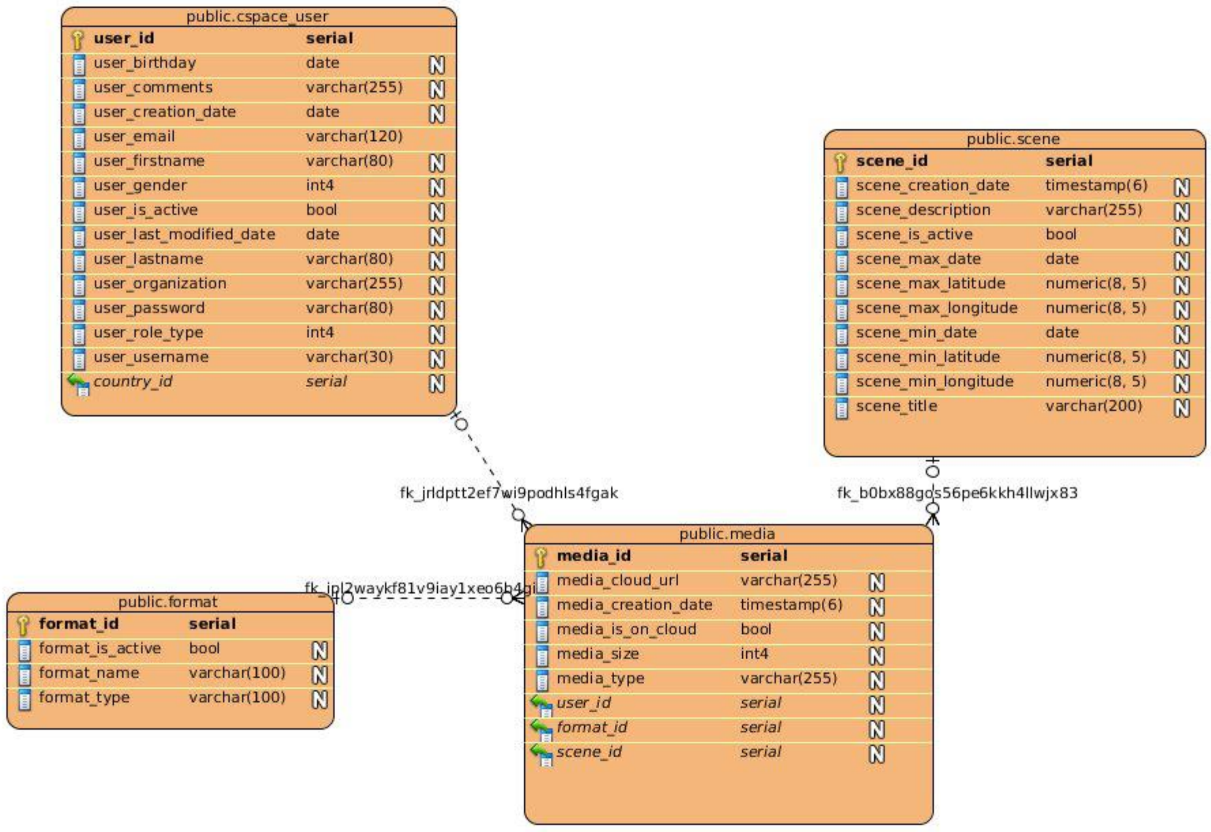

public.cspace_user
user_id serial
user_birthday date N
user_comments varchar(255) N
user_creation_date date N
user_email varchar(120)
user_firstname varchar(80) N
user_gender int4 N
user_is_active bool N
user_last_modified_date date N
user_lastname varchar(80) N
user_organization varchar(255) N
user_password varchar(80) N
user_role_type int4 N
user_username varchar(30) N
country_id serial N
public.scene
scene_id serial
scene_creation_date timestamp(6) N
scene_description varchar(255) N
scene_is_active bool N
scene_max_date date N
scene_max_latitude numeric(8, 5) N
scene_max_longitude numeric(8, 5) N
scene_min_date date N
scene_min_latitude numeric(8, 5) N
scene_min_longitude numeric(8, 5) N
scene_title varchar(200) N
fk_jrldptt2ef7wi9podhls4fgak
fk_b0bx88gqs56pe6kkh4llwjx83
fk_ipl2waykf81v9iay1xeo6h4gi
public.format
format_id serial
format_is_active bool N
format_name varchar(100) N
format_type varchar(100) N
public.media
media_id serial
media_cloud_url varchar(255) N
media_creation_date timestamp(6) N
media_is_on_cloud bool N
media_size int4 N
media_type varchar(255) N
user_id serial N
format_id serial N
scene_id serial N

Figure 1: CAI Upload Entity Diagram

a. The Media Entity

The main entity related to any uploaded content is the Media one. In essence, it represents an uploaded file.

It's properties are:

- *media_id*: Unique Integer ID. Automatically set by the CAI for new records.
- *media_cloud_url*: The file name. Automatically set by the CAI for new records.
- *media_creation_date*: Creation time-stamp. Automatically set by the CAI for new records.

- *media_is_on_cloud*: Boolean flag, indicating whether the file exists or not in the CAI. Automatically set by the CAI for new records.
- *media_size*: The uploaded file size, in bytes. Automatically set by the CAI for new records.
- *media_type*: Represents the uploaded media's type, as perceived in the c-Space context. Automatically set by the CAI for new records. Values currently include: “RAW” for images/video uploaded into the reconstruction pipeline, “RECONSTRUCTION” for 4D reconstructions and “AUXILIARY” for supplementary material, such as historical photos.

b. The Scene Entity

A Scene is a way of logically grouping files together. It represents an “event”, both in time and space. An example would be a rock concert, covering a specific area (defined by a bounding box) and timespan. At the time of writing, scenes are managed by the CAI, to facilitate testing between the different c-Space modules. As the project evolves, though, scene management (creation/selection) functionality will be exposed to clients via the CAI REST API.

It's properties are:

- *scene_id: Unique Integer ID. Automatically set by the CAI for new records.*
- *scene_creation_date: Creation time-stamp. Automatically set by the CAI for new records.*
- *scene_title: A textual title. Will be available for setting via the REST API.*
- *scene_description: A textual description. Will be available for setting via the REST API.*
- *scene_is_active: A Boolean flag indicating whether the scene will be included in searches or not. Managed by the CAI.*
- *scene_min_date: A time-stamp indicating the earliest point in time pertaining to this scene. Will be available for setting via the REST API.*
- *scene_max_date: A time-stamp indicating the latest point in time pertaining to this scene. Will be available for setting via the REST API.*
- *scene_min_latitude: The lowest latitude value (in WGS84) that the scene's bounding box covers. Will be available for setting via the REST API.*
- *scene_max_latitude: The highest latitude value (in WGS84) that the scene's bounding box covers. Will be available for setting via the REST API.*
- *scene_min_longitude: The lowest longitude value (in WGS84) that the scene's bounding box covers. Will be available for setting via the REST API.*
- *scene_max_longitude: The highest longitude value (in WGS84) that the scene's bounding box covers. Will be available for setting via the REST API.*

c. The CspaceUser Entity

The **CspaceUser** entity depicts one of the following:

- *a person using the mobile application*

- *a CAI platform administrator*
- *a third party system accessing the CAI (such as the 4d reconstruction platform)*

At the time of writing, users are managed by the CAI, to facilitate testing between the different c-Space modules. As the project evolves, though, user registration functionality will be exposed to clients via the CAI REST API.

It's properties are:

- user_id*: Unique Integer ID. Automatically set by the CAI for new records.*
- user_birthday*: A time-stamp. Will be available for setting via the REST API.*
- user_comments*: Textual comments. Will be available for setting via the REST API.*
- user_creation_date*: Creation time-stamp. Automatically set by the CAI for new records.*
- user_email*: The user's email. Will be available for setting via the REST API.*
- user_firstname*: Will be available for setting via the REST API.*
- user_gender*: Integer property, possible values are: 1-male, 2-female, 3-unspecified. Will be available for setting via the REST API.*
- user_is_active*: Boolean flag, indicating whether the user can log in to the c-Space platform. Managed by the CAI.*
- user_last_modified_date*: Modification time-stamp. Automatically set by the CAI.*
- user_lastname*: Will be available for setting via the REST API.*
- user_organization*: Will be available for setting via the REST API.*
- user_password*: The user's password. Will be available for setting via the REST API.*
- user_role_type*: Integer property, possible values are: 1-user, 2-system, 3-admin. Automatically set by the CAI.*
- user_username*: An (optional) user name. Managed by the CAI.*
- country_id*: An integer, identifying the user's country. A list of countries, for selection, will be exposed via the CAI REST API.*

d. The Format Entity

Represents various media content formats. A format is automatically assigned by the CAI to any newly uploaded content, based on it's MIME type. It contains only formats relevant to the c-Space platform.

It's properties are:

- *format_id: Unique Integer ID. Automatically set by the CAI for new records.*
- *format_is_active: Boolean flag, indicating whether the CAI accepts files of that format or not. Automatically managed by the CAI.*

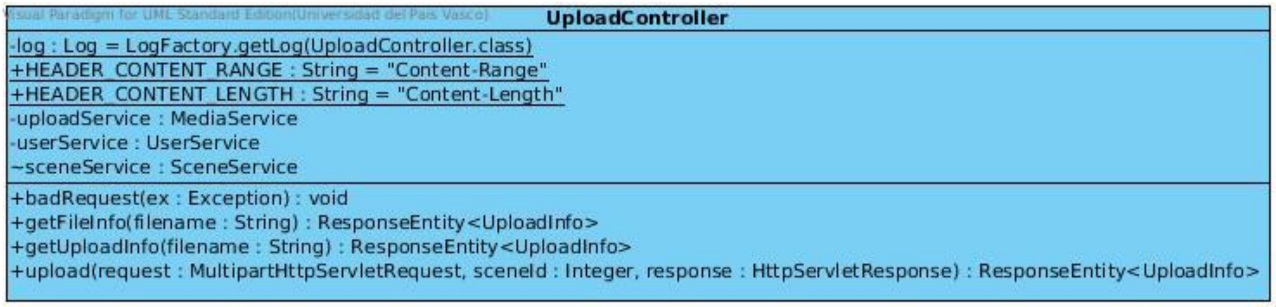
- *format_name: A textual representation of the format file extension, as defined by IANA Media types[1], e.g. 'AVI', 'JPEG'.*
- *format_type: A textual representation of the format type, as defined by IANA Media types, e.g. 'VIDEO', 'APPLICATION'.*

From a high level work-flow perspective, each time a c-Space user (be that a mobile application user or a system, such as the 4D reconstruction pipeline) successfully uploads a new file, a new **Media** entry is created in the database. At the same time, a new folder, unique for the uploaded file and the user uploading it, is created in the file system, and the file is stored in that folder. That way, filename conflicts are avoided, ensuring that even the same user is able to upload a file with the same name multiple times.

If the uploaded file needs to be classified under a specific, existing, c-Space Scene, then a matching parameter is included in the upload request. The CAI then performs the necessary linking between the newly created **Media** and Scene entities.

e. CAI Upload API

The Spring MVC class handling uploads is depicted in the following class diagram:


Visual Paradigm for UML Standard Edition(Universidad del Pais Vasco)

```
UploadController
-log : Log = LogFactory.getLog(UploadController.class)
+HEADER_CONTENT_RANGE : String = "Content-Range"
+HEADER_CONTENT_LENGTH : String = "Content-Length"
-uploadService : MediaService
-userService : UserService
~sceneService : SceneService
+badRequest(ex : Exception) : void
+getFileInfo(filename : String) : ResponseEntity<UploadInfo>
+getUploadInfo(filename : String) : ResponseEntity<UploadInfo>
+upload(request : MultipartHttpServletRequest, sceneId : Integer, response : HttpServletResponse) : ResponseEntity<UploadInfo>
```

Figure 2: CAI Upload Controller

The HTTP REST API method related to the upload is depicted in the following WASDL extract:

```
<resource path="/upload">

        <method id="upload" name="POST">

              <doc title="UploadController.upload" xml:lang="en">Uploads and
                    stores  a media file to the CAI. Supports chunked upload. The
                    client must send a HTTP multipart request. Once the file is
                    uploaded, it is automatically categorized as
                    raw/reconstruction. If a 'sceneId' parameter is included in
                    the request, the file is attached to that scene (if it exists
                    on the CAI)</doc>

              <request>
```

[1] IANA Media types: http://www.iana.org/assignments/media-types/media-types.xhtml

```
                              <param xmlns:xs="http://www.w3.org/2001/XMLSchema"
                                    name="sceneId"

                              style="query" type="xs:int" required="false" />

                      </request>

                      <response status="200">

                                <representation mediaType="application/json" />

                     </response>

                 </method>

        </resource>
```

f. The /upload method

Uploads and stores a media file to the CAI. Supports chunked upload. The client must send a HTTP multipart request, as defined in the Multipart Content-Type W3C[2] specification document. Once the file is uploaded, it is automatically categorized as raw/reconstruction. If a 'sceneId' parameter is included in the request, the file is attached to that scene (if it exists on the CAI)

Parameters:

- *file: an HTTP Multipart request parameter, ponting to the actual file being uploaded.*
- *sceneId: An (optional) existing Scene identifier, under which to attach the uploaded content.*

Headers:

- *Content-Range HTTP Header: contains info about the number of bytes being uploaded, as well as the total size of the uploaded file, as described in Section 14.16 - Content-Range of the* HTTP/1.1: Header Field Definitions*.[3]*
- *Basic Authentication Header: contains the c-Space user email and password, under which the upload is being performed, as defined* in Section 11.1 - Basic Authentication Scheme *of the Hypertext Transfer Protocol specification.[4]*

Return type: a JSON 'UploadInfo' object containing the following data:

- *name (String): the file name*
- *size (Long) : the uploaded file's current size*

[2] Multipart Content-Type: http://www.w3.org/Protocols/rfc1341/7_2_Multipart.html

[3] HTTP/1.1:Header Field Definitions: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

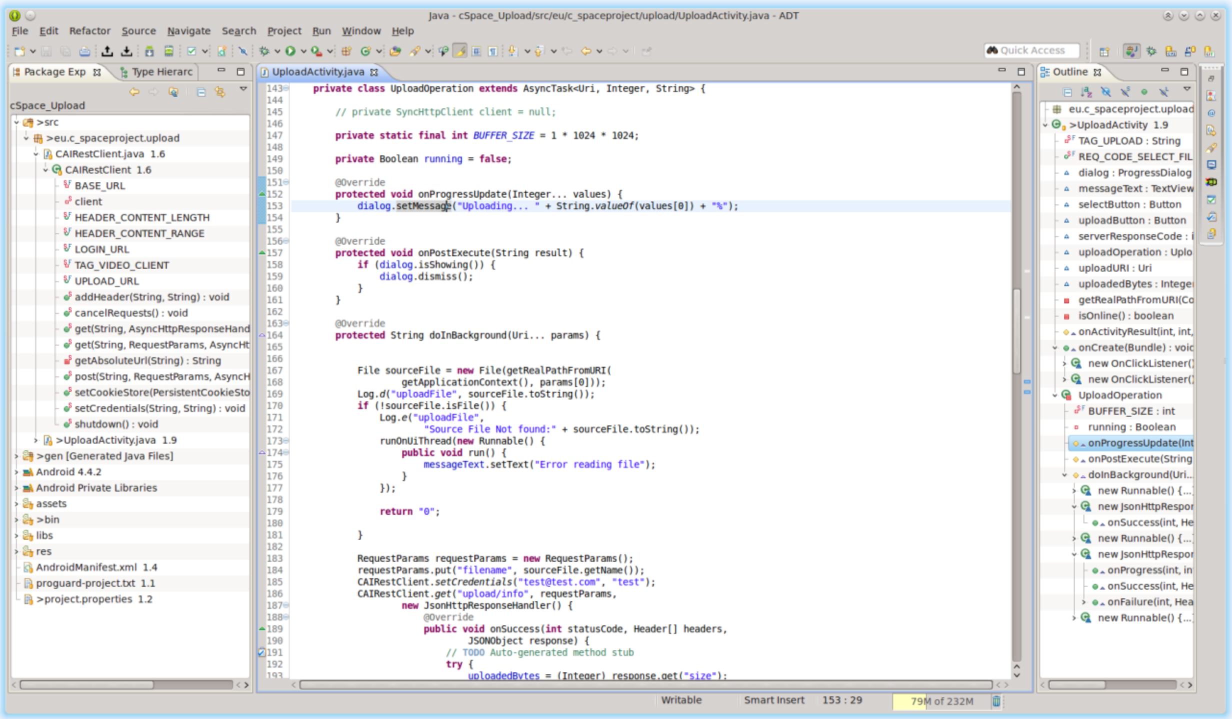
[4] Section 11.1 – Basic Authentication Scheme: http://www.w3.org/Protocols/HTTP/1.0/spec.html#BasicAA

- *isComplete (Boolean): a flag indicating whether the upload is complete or not.*

In the case of chunked upload, the 'size' & 'isComplete' fields become crucial, since the client can monitor the upload progress and continue sending file chunks or not.

3. Client (Android) side

The core of the Android code related to the uploading process is based on Android Asynchronous Http Client[5], an Apache Licensed library, implementing an asynchronous callback-based Android HTTP client. A c-Space specific wrapper class, CAIRestClient, was created around this library, along with a sample Android application, showcasing the entire upload work-flow, including chunked upload support.



```
    private class UploadOperation extends AsyncTask<Uri, Integer, String> {

        // private SyncHttpClient client = null;

        private static final int BUFFER_SIZE = 1 * 1024 * 1024;

        private Boolean running = false;

        @Override
        protected void onProgressUpdate(Integer... values) {
            dialog.setMessage("Uploading... " + String.valueOf(values[0]) + "%");
        }

        @Override
        protected void onPostExecute(String result) {
            if (dialog.isShowing()) {
                dialog.dismiss();
            }
        }

        @Override
        protected String doInBackground(Uri... params) {


            File sourceFile = new File(getRealPathFromURI(
                    getApplicationContext(), params[0]));
            Log.d("uploadFile", sourceFile.toString());
            if (!sourceFile.isFile()) {
                Log.e("uploadFile",
                        "Source File Not found:" + sourceFile.toString());
                runOnUiThread(new Runnable() {
                    public void run() {
                        messageText.setText("Error reading file");
                    }
                });

                return "0";

            }

            RequestParams requestParams = new RequestParams();
            requestParams.put("filename", sourceFile.getName());
            CAIRestClient.setCredentials("test@test.com", "test");
            CAIRestClient.get("upload/info", requestParams,
                    new JsonHttpResponseHandler() {
                        @Override
                        public void onSuccess(int statusCode, Header[] headers,
                                JSONObject response) {
                            // TODO Auto-generated method stub
                            try {
                                uploadedBytes = (Integer) response.get("size");
```

Figure 3: c-Space Upload Development Environment

The main classes involved in the Android code are depicted in the following class diagram:

[5] Android Asynchronous Http Client: http://loopj.com/android-async-http/

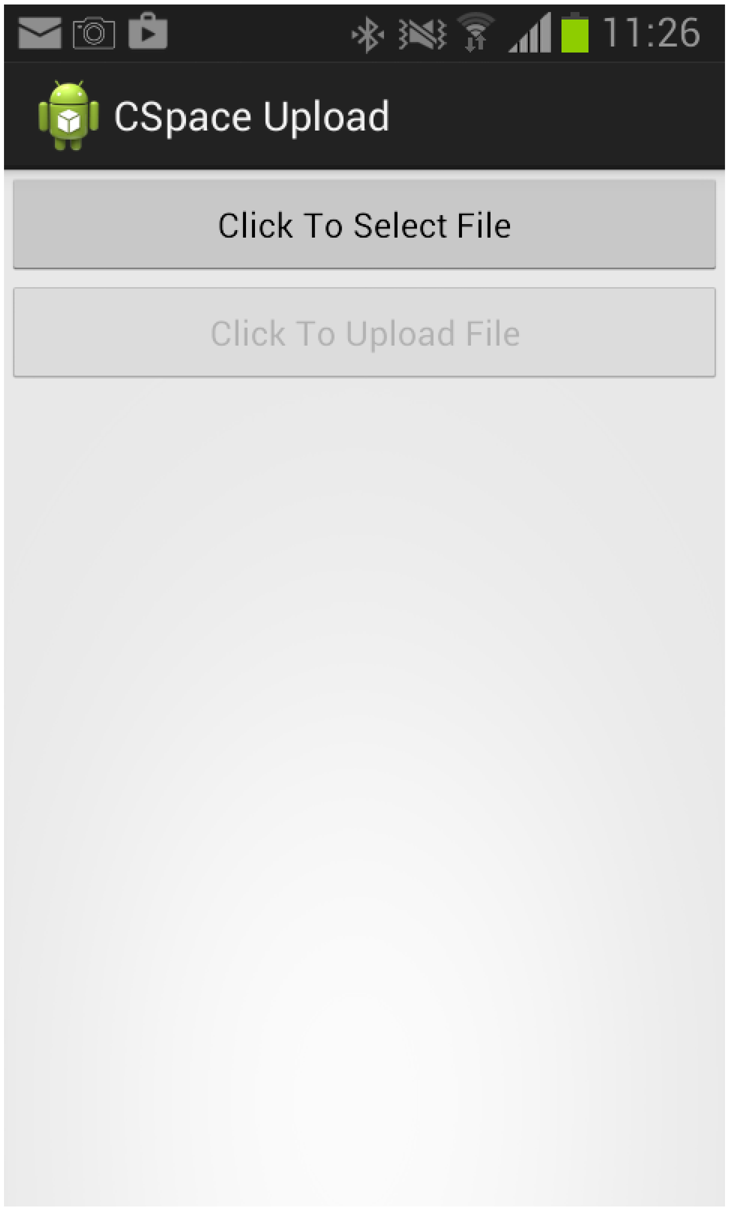


Figure 4: c-Space sample Upload Application

a. The UploadActivity class

'UploadActivity' is a standard Android Activity, handling the sample application's GUI. The user browses and selects a file from the mobile device's storage system, using the 'Click to Select File' button, and then presses the 'Click to Upload File' to initiate the actual upload to the CAI, via the 'UploadOperation' class.

b. The UploadOperation class

The 'UploadOperation' is a an inner class of UploadActivity. At the same time it is a sub-class of Android's AsyncTask (which means it gets executed in a separate thread), since it actually sends the data to the CAI, which is time consuming and should not block the application interface.

It's 'doInBackground' method recursively calls the CAIRestClient's 'upload' method, checking the returned result each time, in order to determine whether the upload is complete or not.

Once complete (or failed) it notifies the main application UI thread, via the 'onSuccess' and 'onFailure' methods accordingly.

c. The CAIRestClient

Contains the “low-level” HTTP calls to the CAI.

The methods pertaining to the upload are, at the moment of writing, the following:

- *setCredentials*: Called once, before the first HTTP call to the CAI. Adds the Basic Authentication Header, which should contain the c-Space user email and password, as defined in Section 11.1 - Basic Authentication Scheme of the Hypertext Transfer Protocol specification.
  - Parameters:
    - username (String): a valid c-Space user's email.
    - Password (String): the user's password.
  - Return type: void.
- *addHeader:* Adds an HTTP header to the next HTTP call. .
  - Parameters:
    - header (String): the HTTP header name (e.g.”Content-Range”).
    - value (String): the header's value.
  - Return type: void.
- *post: performs an HTTP post to the CAI.*
  - Parameters
    - url (String): The CAI HTTP URL to make the call to (e.g. “upload”)
    - requestParams: A loopj RequestParams instance, containing the HTTP request parameters to include in the request.
    - responseHandler: a sub-class of loopj's AsyncHttpResponseHandler , handling the HTTP response.
- cancelRequests: used to cancel all ongoing request.

Sending a file/chunk upload request to the CAI requires the following calls, described in order of execution:

1. CAIRestClient.setCredentials: In order to set a valid c-Space user's credentials (email/password).
2. CAIRestClient.addHeader: In order to add the Content-Range HTTP Header, which contains info about the number of bytes being uploaded, as well as the total size of the uploaded file, as described in Section 14.16 - Content-Range of the HTTP/1.1: Header Field Definitions.

3. CAIRestClient.post: Performs the actual upload. It's parameters should have the following values:
    a. url: “upload”
    b. requestParams: a RequstParams instance, containing the following values:
        i. *file* (ByteArrayInputStream): a Java InputStream, containing the bytes to upload.
        ii. *sceneId* (Integer, optional) an existing c-Space Scene ID, under which to classify the uploaded file.
        iii. responseHandler: an instance of loopj's JsonHttpResponseHandler. It handles both request success and failures, in the following methods:
            - onSuccess: accepts a CAI JSON UploadInfo object, containing the following properties:
                - size (Integer): the upload file's size.
            - onFailure: accepts an HTTP statusCode, indicating the error thrown by the CAI.

The 'UploadInfo' JSON response (as described in the 'CAI upload method' section of this document) received after each successful request, is examined to determine whether an upload has completed or not.

4. The Upload process

The sequence diagrams depicting the upload operation between the mobile application, as well as the upload operation from the 4D reconstruction pipeline, and the CAI server are shown below.

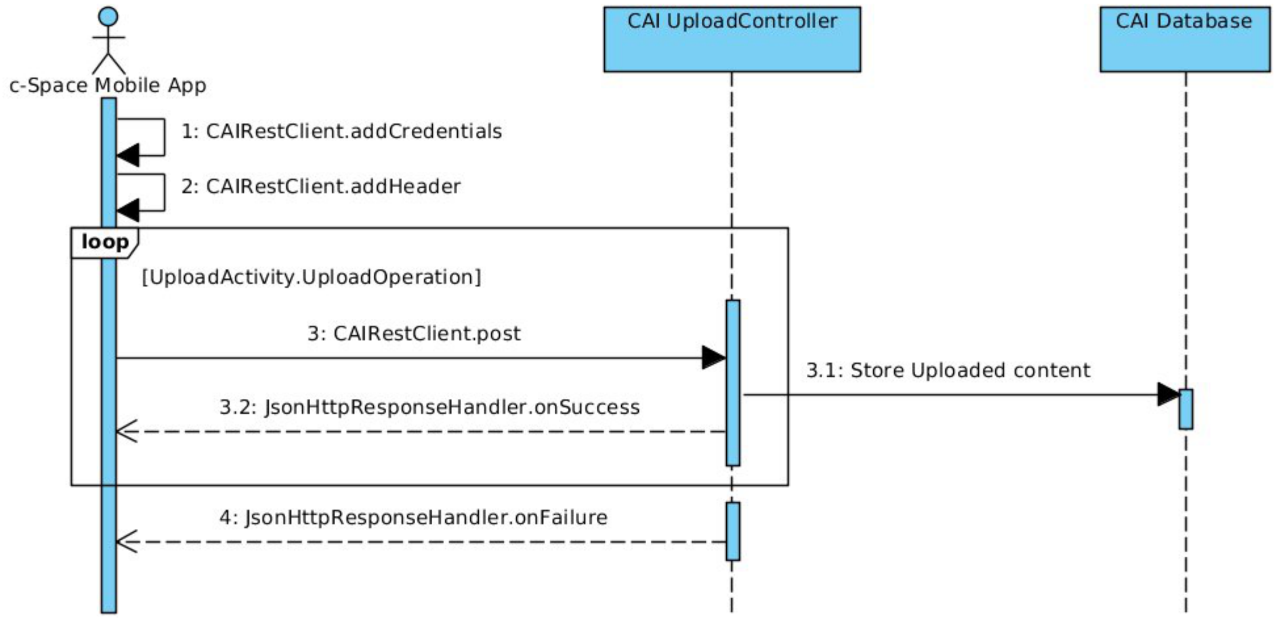

c-Space Mobile App
CAI UploadController
CAI Database
1: CAIRestClient.addCredentials
2: CAIRestClient.addHeader
loop
[UploadActivity.UploadOperation]
3: CAIRestClient.post
3.1: Store Uploaded content
3.2: JsonHttpResponseHandler.onSuccess
4: JsonHttpResponseHandler.onFailure

Figure 5: c-Space Mobile Application Upload Sequence Diagram

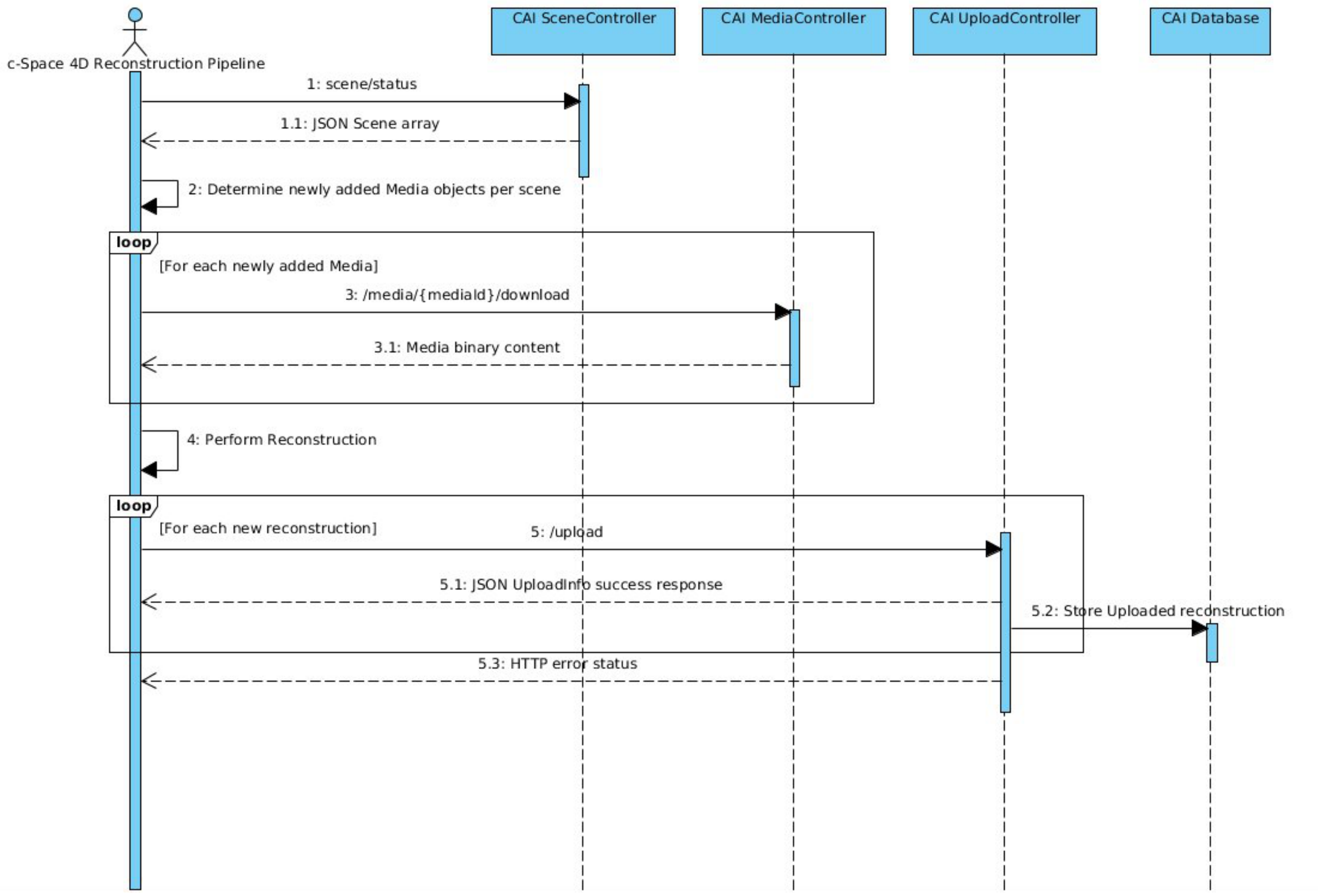

c-Space 4D Reconstruction Pipeline
CAI SceneController
CAI MediaController
CAI UploadController
CAI Database
1: scene/status
1.1: JSON Scene array
2: Determine newly added Media objects per scene
loop
[For each newly added Media]
3: /media/{mediaId}/download
3.1: Media binary content
4: Perform Reconstruction
loop
[For each new reconstruction]
5: /upload
5.1: JSON UploadInfo success response
5.2: Store Uploaded reconstruction
5.3: HTTP error status

Figure 6: c-Space Reconstruction Upload

5. Notes & Future Actions

Aside from the mobile application being used as the main client for the upload, other entities (such as project partner FRAUNHOFER) have already implemented their own custom clients, in order to be able to upload additional content (such as 4D reconstructions produced by the reconstruction pipeline). Since HTTP well known standards have been used throughout the implementation, the basic principles between different clients remain the same.

Next steps will involve adding additional parameters, as required by the c-Space partner responsible for the mobile application developer. It may also include, if the need arises, implementing other upload clients, such as a web based upload, for c-Space users. A refactoring of the Android client code will also be performed, in order to simplify usage by partner GRAPHITECH.

6. References

1 IANA Media types. Available online from: http://www.iana.org/assignments/media-types/media-types.xhtml

2 Multipart Content-Type. Available online from: http://www.w3.org/Protocols/rfc1341/7_2_Multipart.html

3 Section 14.16 - Content-Range of the HTTP/1.1: Header Field Definitions: Available online from: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

4 Section 11.1 - Basic Authentication Scheme – Benefits Available online from: http://www.w3.org/Protocols/HTTP/1.0/spec.html#BasicAA

5 Android Asynchronous Http Client. Available online from: http://loopj.com/android-async-http/