KML-O lite

Ian James Daniel Gonzales

Project in IMT4951 - Applied Digital Workflow
Master in Media Technology
Avdeling for informatikk og medieteknikk
Høgskolen i Gjøvik, 2009

Avdeling for
informatikk og medieteknikk
Høgskolen i Gjøvik
Postboks 191
2802 Gjøvik

Department of Computer Science
and Media Technology
Gjøvik University College
Box 191
N-2802 Gjøvik
Norway

KML-O lite

Ian James Daniel Gonzales

27.10.2009

Abstract

The background and motivation for this project can be stated by this quote of Kjell Are Refsvik, our class advisor, in one of his e-mails to our class.

> "'The amount of work needed to write KML-documents by hand or explicitly using a GUI software and/or proprietary solutions could be described as a problem that takes time and money and locks data and users to proprietary solutions.'"

As such a tool which can handle the problem mentioned above would be ideal.

The ultimate goal of the project is to make a KML[1] application/program that can read meaningful files with geo-data and produce a KML file as an output. The KML file should give a reference to the original input files. Furthermore an additional feature could possibly be implemented. The said feature is to make the solution dynamic, so that newly added files to a particular folder on a server would be added instantly to the KML file constantly.

With that said, I proceeded to make a prototype of said application in Python. In general it is possible to use any other scripting languages (such as Perl, Ruby, etc.) or general programming languages (such as Java, C++, etc.) to implement this, but my choice of Python will be discussed in a later chapter.

I had to gather images in different formats (but mainly in JPG format) and with geo-datas. It was also eminent to use applications that can read a KML-file. Google Maps was the obvious and perhaps the easiest choice, but I also used Google Earth to test the results locally in my computer.

My program and findings will show you that it is possible to make a program that can automatize the mentioned problem above. But throughout my work, I have encountered many problems and difficulties with making this application. I will go deeper into explaining these things later.

[1] `http://code.google.com/intl/no/apis/kml/documentation/whatiskml.html`

Preface

I would like to thank Juan Antonio Sances for providing me with most of the images I used, and also Reda for letting me test my program in your Mac. Cheers to all the international students in Sørbyen - it is nice to have a cheerful bunch of people. Also I wish us Good luck for our finals!

I would also like to thank my previous mentors in University of Oslo as well as my present mentors in Høgskolen i Gjøvik.

Lastly I want to give a big thank to the open source community as well as programmers/developers who helps newbies like me with your vast knowledge!

Ian James Daniel Gonzales, 27.10.2009

Contents

1 Introduction

1.1 The Project in General

This is a project in IMT4951, Applied Digital Workflow - a course I have this semester. For our final task in the course, we are asked to do 1 of 5 projects described in the course's *website* [1]. Common for all of them is to have an understanding and knowledge on how the flow of digital medium is being used with real world problems - and thereby applications.

Some of these projects focus on web applications or a good presentation in the web. This is not the case with my project. I have chosen to make an application which produces KML-files. Although my application is not Internet-based, what its functionality and product can be further used in other *mashup*[2] projects.

Another important aspect common for all of these projects is documentation with LaTeX[3] and good documentation in general. So aside from my project proposal and my *readme*[4]-document for my API, all of my documents concerning this project are written in LaTeX. This is also a good opportunity for us to be used to LaTeX in preparation for future master and perhaps PhD thesis. I also use a revised version of a TeX-template from HiG[5].

The last aspect in this project I had to consider is the idea of intellectual property rights and open source sharing (*see Good Copy, Bad Copy*)[6]. While I only used code snippets from official application tutorials, I have all throughout my work been reading and researching other peoples materials and solutions in the Internet. These are of course materials which they allowed to be shared to other people and perhaps especially to other programmers. This is what allowed me to finish my work. I have also re-used most of my codes from when I was still doing my bachelor. It is my wish to also share my work to other people under creative common license[7][8]. In developing KML-O, it was in the back of my mind to use free and open sourced applications and environment for my API. This is to ensure that I could share my API fully free - in other words aside from buying a computer, you would not need to buy anything to make my API work.

[1] http://www.ansatt.hig.no/kjellr/imt4951/index.html
[2] http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29
[3] http://en.wikipedia.org/wiki/LaTeX
[4] http://www.stud.hig.no/~090285/readme.html
[5] http://www.hig.no/imt/thesis/latex
[6] http://www.goodcopybadcopy.net/
[7] http://en.wikipedia.org/wiki/Creative_commons
[8] http://creativecommons.org/

1.2 The Application Itself

As mentioned in the abstract, the need for a tool for making a KML-file based on geo-tagged images is the driving motivation for making this project. I will show you that it is possible to make such an automation using Python and make it shareable and readable to other people, so that it is possible for other people to build upon it in the future.

> **3. Tool for creating KML-files**
> A general tool that could be very useful, is one that could take a group of well described files (that includes geo-data) and create a reference to them using the Keyhole Markup Language (KML) based on the metadata inside the files. An additional challenge could be to make the solution dynamic, so that newly added files to a particular folder on a server would be added instantly to the kml file constantly.

This is a relevant application since the need for KML-files in different mashup projects is an actual demand in the real world. See for example the first 2 project proposals in `http://www.ansatt.hig.no/kjellr/imt4951/index.html`.

1.2.1 KML-O lite

KML-O stands for Keyhole Markup Language - Output. This is because the application produces a KML file as the output. I also use 'lite' since this is a relatively small program. My approach in doing the program was early prototyping with agile development in mind. [9].

KML-Output is a Python based script developed in and mainly for Linux-based operative systems.[10] This is because I have tested it mainly in Ubuntu and found it working there perfectly. In Windows XP it also works fine with some adjustments with the `PATH` environment variable or alternatively a small 'hack' - which I also included inside the program. I tried running it in Mac, but since I don't have good enough experience with Mac OS, I could not install the additional Python modules. Although theoretically it should work.

You need Python to run the program. This is already pre-installed in most `UNIX` based OS such as Linux, Mac OS and Cygwin. For Windows and other OS it is needed to be installed manually or alternatively install Cygwin.

1.2.2 Version Notes

There are 4 main versions of the application;

Version 1 KMLo.py

Version 2 KMLo_v2.py

Version 3 KMLoGUI.py

[9] `http://en.wikipedia.org/wiki/Agile_software`
[10] See the conclusion to why mainly Linux-4

Version 4 KMLo_GUI.py

Version 1 Note

KMLo.py[11] is the very first prototype I have made. It has all the minimum functionalities described in the project description but also the additional dynamic solution for producing KML-file.

It makes use of two external modules, Image and Pyinotify. These are needed to be downloaded and installed for the script to work. Image is responsible for image manipulation and metadata extraction while Pynotify takes care of event-handling. The glob module is also imported, but is not necessary to download and install since it is already pre-installed in Python. This is responsible for taking specific image format(s) from the working directory.

The only allowed image format in this version is `'.JPG'`, and the only allowed working directory is the directory you run the script in. The script is also purely command line driven.

The events being watched is file-creation event and file-deletion event - though the file-deletion event doesn't really do anything.

You run the script by calling:
`>./KMLo.py` or `>python KMLo.py`

You stop the script by using `CTRL+C`.

Version 2 Note

Pyinotify functionality for event-handling is omitted. This is also more minimalistic than the first version. It came about as a response for troubles in installing python modules in Mac. [12]

From Pyinotify website;

> "Pyinotify relies on inotify, a Linux Kernel functionnality (starting with kernel version 2.6.13). inotify is an event-driven notification mechanism, its notifications are exported to user space through three system calls. Pyinotify binds these system calls and provides an implementation on top of them."

Mac does not use Linux kernel..
You run the script by calling:
`>./KMLo.py` or `>python KMLo.py`

Version 3 Note

KMLoGUI.py is my first attempt in to making a GUI for the application. There are now 3 events Pyinotify tries to watch here:file-creation, file-deletion, file-copy. It is worth noting that Pynotify

[11] `http://www.stud.hig.no/~090285/KMLo.py`
[12] `http://www.stud.hig.no/~090285/KMLo_.py`

can handle all sort of events other than the 3 mentioned here.

It also imports Tkinter and Pmw modules for generation of a GUI. So these modules are needed to be installed beforehand.

I stopped distributing this version since it has many bugs.

Version 4 Note

The latest version of the script. This version is well commented and more structurized.

Module Pyinotify and it's event-handling functions is replaced by time module and a self-written function for event-watcher called `def myupdate()`. This is the easier approach to how to make an event-watcher. The downside of this is that it cannot watch over advanced events other than file-creation and file-deletion, but the good thing is that it is not necessary for advanced events for this project.

The module glob is no longer in use. Instead a self-written function called `find()` finds the needed images. Image formats are all expanded to include JPG, jpg, JPEG, jpeg, PNG, png, GIF, gif, TIFF, tiff, TIF and tif.

The `def upload_file()` and `def show_link()` are added to upload the images and KML-file to a remote server, and to show the link of the uploaded file in Google Maps respectively.

2 Implementation

2.1 Data Collection and Description

KML-O uses images as input data. In its current version, it can take images in JPEG, GIF, TIFF and PNG format. While it is possible to add more picture formats, I have decided to limit it to these 4. It also needs to be said that the usual and recommendable format would be JPEG. It is usually not too large and is widely and popularly used format especially in web-based applications.

While this project is not primarily focused with data-collection, there are some things that needs to be mentioned here. The thought behind the API is that it would use images with geographical data like longitude, latitude and altitude of where an image was taken. For this reason, the use of cameras with built-in GPS functions is ideal. There is also the option of using an external GPS device which you can plug in to a digital camera. And alas a user can manually write in the geo-datas in to the image by using programs such as `EXIFTOOL`[1] or `EXIF.PY`[2]. In my case, I did not take any pictures myself. I have asked some friends and colleagues to donate some images with geo-data and also I have downloaded some from the Internet and manually put the geo-data using EXIFTOOL.

Although the essence of the API is to use geo-tagged images, I have made it so that it can read images without geo-data as well. When it finds images without geo-data it will use the default values for longitude, latitude and altitude which is (0,0,0). It will then set the `userComment` field to 'No geo-datas found...'. The following function is responsible for extracting the necessary data from an image;

```
#Responsible for getting the exif-datas from a image
    def getMetaData(self, file):
f = Image.open(file)

"""
The test here tests if the opened file has exifdatas which the Image module can read.
"""
if not hasattr(f, '_getexif'):
return ("No exif data", "No exif data", "No exif data", "No exif data", 0, 0, 0, "No exif
data")
try:
exift = f._getexif()
```

[1] `http://www.sno.phy.queensu.ca/~phil/exiftool/`
[2] `http://sourceforge.net/projects/exif-py/`

```
except:
a = 'nothing to do'

try:
createDate = exift[0x9003] # creation date
except:
createDate = "Unknown"
try:
imageDesc = exift[0x010e] # "Image Description"
except:
imageDesc = "No Image Description"
try:
copyright = exift[0x8298] # copyright
except:
copyright = "No Copyright"
try:
author = exift[0x013b] # Author/Photographer
except:
author = "Photographer Unknown"
try:
userComment =  str(exift[0x9286]) # "User Comment"
"""
The if-test tests if the first character is null(\x00), which is the default for all pictures
  if the author haven't changed the UserComment-tag. This means that this tag is undefined so
  I just set 'No User Comment'.

Because of some bug, even if a user puts in his own comment in the UserComment-tag, the first
part of the of the string will be 'ASCII\x00\x00...'. This messes up the kml-file being
generated, since it can't read the hex code.

So in the else-part, I take out this 'bugged' part.
"""
if userComment[0]=="\x00": # tests if the first character is null
userComment = "No User Comment"
else:
userComment = " ".join(userComment.split()[1:])
except:
userComment = "No User Comment"

"""
Taking the geo-datas. If none is found it is set to (0,0,0).
"""
```

```
try:
refLat =  exift[0x8825][1] # latitude reference
degLat=  exift[0x8825][2][0][0] # degree
miNLat =  exift[0x8825][2][1][0] # minute
secNLat = exift[0x8825][2][2][0] # second/numerator
secDLat = exift[0x8825][2][2][1] # second/denomenator
lat = self.getCoords(degLat, miNLat, (float(secNLat)/float(secDLat)), refLat)

refLong =  exift[0x8825][3] # longitude reference
degLong =  exift[0x8825][4][0][0] # degree
miNLong =  exift[0x8825][4][1][0] # minute
secNLong = exift[0x8825][4][2][0] # second/numerator
secDLong = exift[0x8825][4][2][1] # second/denomenator
lonG = self.getCoords(degLong, miNLong, (float(secNLong)/float(secDLong)), refLong)

altitude =  float(exift[0x8825][6][0])/float(exift[0x8825][6][1]) # altitude
except:
lat = 0
lonG = 0
altitude = 0

userComment = "No geo-datas found. Coordinates set to (0,0,0) by default."

return (imageDesc, copyright, author, userComment, lonG, lat, altitude, createDate)
```

The function first tests if an image has `exif`[3] meta data. If it doesn't find any it will use the default values `("No exif data", "No exif data", "No exif data", "No exif data", 0, 0, 0, "No exif data")`. The function is extracting the following `exif`[4] metadata;

- Image Description
- Copyright
- Author
- User Comment
- Longitude
- Latitude
- Altitude
- Creation Date

There are other metadata that can be extracted such as what type of camera was used in capturing the image, but I think the meta data I use now are the most relevant. There is always

[3] `http://en.wikipedia.org/wiki/Exif`
[4] `http://www.sno.phy.queensu.ca/~phil/exiftool/TagNames/EXIF.htmldata`

a default value set to these fields if the function cannot find it (f. ex. if the image is computer generated). Values found here are used in the creation of the KML-file.

The function also relies in the python module `Image`[5] for reading and getting the `EXIF` data. This is instead of calling and using an external program like `EXIFTOOL` (f. ex. `os.system('EXIFTOOL img.jpg')`).

2.2 Data Transformation

The program is again not primarily focused in transforming and transcoding data - at least not in the sense of transcoding an image format to another or resizing it (F. ex. by using *ImageMagick:* [6] `mogrify jpg -thumbnail 60x60 *.JPG`). But with that said, It is a good idea for future implementations to have a function, for example `def resizeImage(image)`, that resizes an image to a smaller thumbnail so that it is easier and faster to upload the images in a remote server and also so that they are faster to upload and show in Google Maps.

The Python module `Image` together with Python itself provides the necessary library and utilities to be able to do image transformation and transcoding so that you don't need to call an external program to do this like ImageMagick.

2.3 Data Movement

This is also not included in the project's specification. But I still wanted to add it as an extra functionality for my final version of the application. The following function is responsible for data movement;

```
"""
Uploads the images and kml-file to the chosen server by calling scp. At the moment this
function is only for UNIX based OS. Final command would be something like;
'scp img1 img2 img3 kmlo.KML USERNAME@loke.hig.no:public_html' where;
self.e.get() = gets USERNAME
self.f.get() = gets HOSTNAME
self.g.get() = gets DESTINATION_FOLDER
"""
    def upload_file(self):
try:
tmp = 'scp '
for a in self.dic['image_files3']: #Getting the images
tmp = tmp + str(a) + ' '
tmp = tmp + 'kmlo.KML ' + self.e.get() + '@' + self.f.get() + ':' + self.g.get()
```

[5] `http://www.pythonware.com/products/pil/'`
[6] `http://www.imagemagick.org/script/command-line-tools.php`

```
run = os.system(tmp)
if run:
message = self.errorMSG('Cannot run: %s'% tmp)
except:
message = self.errorMSG('Error in uploading the kml file and images')
```

This function relies on the UNIX' protocol, scp[7]. It is also possible to use sftp[8] instead, but since I would like to run an one-line command call for uploading images I've used scp.

The function looks at the array of the images to be uploaded and adds them one by one to the command call and finally also the KML-file. Lastly it will use the input for USERNAME, HOSTNAME and DESTINATION_FOLDER. It will then call scp. The user will then be asked to write their PASSWORD.

The call would not be successful if the USERNAME, HOSTNAME, DESTINATION_FOLDER or PASSWORD are wrong or not filled.(Please read the *readme.txt* for how to run the API and the description of the input fields, etc..)

Although Python has its own module for doing data movement, I have decided to use this solution instead because of some problems I have encountered (I will discuss this in the conclusion). Because of this, the function won't work in windows (without cygwin[9]) or platforms that doesn't have the scp protocol.

2.4 Mashup using Google Maps

A successful run of the program will produce a KML-file. This can be run in Google Maps, but it will depend on the KML-file being accessible in the Internet. So a user needs to upload the KML-file either manually or by using the built-in function to upload files (see chapter 2.3).

I've also used Google Earth to run the KML-file locally in my PC. I have seen that there are other applications that may be able to run the file and produce mashup (like Google Earth as a Web browser add-on), but since Google Maps is the most famous one I decided to choose it. But for future implementation, it is possible to make a choice of whether to run the KML-file locally by using f. ex. Google Earth or in the Internet by Google Maps or other similar applications.

This function is responsible for showing the uploaded KML-file in Google Maps;

```
 #Shows the uploaded kml-file in googlemaps
    def show_link(self):
try:
```

[7] http://en.wikipedia.org/wiki/Secure_copy
[8] http://en.wikipedia.org/wiki/SSH_file_transfer_protocol
[9] http://en.wikipedia.org/wiki/Cygwin

```
    webbrowser.open_new('http://maps.google.com/maps?q='+self.p1.get()+'kmlo.KML')
except:
message = self.errorMSG('Error in opening your link in the googlemaps:
%s'% str(self.p1.get()))
```

It calls for the user's default Web browser (Mozilla, Internet Explorer, Opera, etc..) and go to Google Maps' (`http://maps.google.com/maps?q=`) default page + the Web address of the KML-file.
`self.p1.get()` takes the user-written address from the `IMAGE_URL` field. (See *readme.txt [1] for how to use the API and the explanations of the different input fields*)

2.5 Data Automation

This will be the biggest part and main essence of this application and project - Work Automation. The central core for the automation is writing the KML-file. The following function has the main and final responsibility for making the KML;

```
#Writes the kml-file and calls it kmlo.KML
    def makeKML(self):

if self.makeKMLButton:
self.listbox2.setlist(' ') #Resets the 2nd listbox
self.dic['added_files'] = [] #Resets array for added_files

imageDic = {}
outfile = open('kmlo.KML', 'w')
outfile.write('<?xml version="1.0" encoding="UTF-8"?>\n')
outfile.write('<kml xmlns="http://www.opengis.net/kml/2.2">\n')
outfile.write('<Document>\n')
outfile.write('<name>Automatic Generation of KML file based on images</name>\n')

for image in self.dic['image_files']:
imageDic[image] = self.getMetaData(image[0])
outfile.write('<Placemark>\n')
outfile.write('<name>')
outfile.write(image[0].split(os.sep)[-1])
outfile.write('</name>\n')
outfile.write('<description>\n <![CDATA[\n')
outfile.write('<table border="0" padding="5">\n')
outfile.write('<tr>\n')
outfile.write('<td width="600px" valign="top" float="top">\n')
outfile.write('<p>Taken '+(imageDic[image])[7]+'</p>\n')
```

```
outfile.write('<img src='+ self.p1.get() +str(image[0].split(os.sep)[-1])+' width="300px" border="1p
outfile.write('<p><small>\n')
outfile.write('Photography by: '+imageDic[image][2]+'</br>\n')
outfile.write('Copyright: '+imageDic[image][1]+'</br>\n')
outfile.write('Additional Description: '+imageDic[image][0]+'</br>\n')
outfile.write('User Comment: '+imageDic[image][3]+'</br>\n')
outfile.write('</small></p>\n')
outfile.write('</td>\n')
outfile.write('</tr>\n')
outfile.write('</table>\n')
outfile.write(']]>\n')
outfile.write('</description>\n')
outfile.write('<Point><coordinates>')
outfile.write(str(imageDic[image][4])+','+str(imageDic[image][5])+','+str(imageDic[image][6]))
outfile.write('</coordinates></Point>\n')
outfile.write('</Placemark>\n')


outfile.write('</Document>\n')
outfile.write('</kml>')
outfile.close()
return 0
```

The function makes use of the general tree structure common for all XML documents[10]. Also to be able to understand the arrays and *dictionaries*[11](also called hashmaps in other programming languages) being used in the script, we need to look at how they are declared at the start of the script;

```
self.dic = {} # dictionary to hold all valid values (targetname, max_size, max_age, etc.)

#initialize dic
self.dic['targetnames'] = [] #All available file formats
self.dic['image_files'] = [] #Contains images to be used in KML together with their file size
(and possibly other things like file age)
self.dic['image_files2'] = [] #Contains all images read in the working directory. This works as
the 'watcher', since it will know if new files are added.
self.dic['image_files3'] = [] #Basically the same as 'self.dic['image_files']', but without the
additional infos such as file size.
self.dic['added_files'] = [] #All new files added to the working directory is added here.
```

[10] http://en.wikipedia.org/wiki/Xml
[11] http://www.tutorialspoint.com/python/python_dictionary.htm

```
self.makeKMLButton = False #Turns to true if the "make_kml-button" is used.
```

There is 1 main dictionary which has arrays(also called lists) as elements. The most important among these lists are;

- `dic['targetnames']`: This contains what file formats are allowed as input
- `dic['image_files2']`: Acts as a watcher in that it will contain all allowed images in the working directory at a period. This is periodically updated to see if there are new images added in the directory by calling `def myupdate()`.
- `dic['image_files']`: This contains the actual images that is going to be used in the creation of the KML-file. Every time the `MAKE_KML` botton is used, this is updated and eventually used.

One function we also need to take note of is `def getCoords(degrees, minutes, second, direction_reference)`. It is responsible for converting from *sexagesimal*[12] format to decimal format.[13]

We need to the conversion because by default the geo-data in images are in Sexagesimal format, while KML uses decimal format.

```
# Function to convert from sexagecimal to decimal
    def getCoords(self, d, m, s, ind):
"""
    You have degrees, minutes, and seconds (-73deg 59' 14.64") instead of decimal degrees
    (-73.9874 deg)
    The whole units of degrees will remain the same
    Divide the seconds by 60 (14.64 / 60 = 0.244)
    Sum the resulting to the minute and divide by 60 (59.244 / 60 = 0.9874)
    The resulting is the decimal value of degrees (0.9874)
"""
sec = float(s/60)
miN = float((sec+m)/60)
deg = float(d+miN)
# If it is a West or S longitude coordinate, negate the result.
     if ind == 'W':
     deg = deg * -1

     if ind == 'S':
     deg = deg * -1

return deg
```

[12] http://en.wikipedia.org/wiki/Sexagesimal
[13] http://www.sunearthtools.com/dp/tools/conversion.php

The `if-tests` at the end of the function negates the result if the orientation for the geo-data is either west or south and not north or east.

To sum the main idea of the script see figure below;

```
Import Python-modules
Start main class
Declare Necessary Variables (Dictionary, List, etc.) and Initiate them
Make graphical user interface
Loops and wait for user interaction
While (1):
   if QUIT: Terminate and exits program
   else if MAKE_KML:
        Checks all files in the working directory
        Checks accepted image formats
        Store all valid image-files in a list
        Starts to write KML-file
            for each image in list do:
                Extract needed metadata
                    if no metadata: use default values
                    if metadata==geodata:
                        Convert from sexagesimal to decimal
                Write image_name, image_description, copyright, author, user_comment and
                gps-coordinates in KML-file
        Finish writing the KML-file
        Set the script in 'watcher' mode
            for every 2 sec: check for new files
                if new files found: MAKE_KML
  else if UPLOAD_FILE:
        Upload images and KML-file to IMAGE_URL
  else if SHOW_KML:
        Show uploaded KML-file in Google Maps
  else (All other invalid inputs)
        Show error message
```

Note: The above pseudo-coding is not based on any coding conventions of any sort. This is only my poor attempt to give a summarize version of what the script does. I am sorry if I offend any other programmers who might read this.

3 End Product

I wont write much in this part here since Python has a good add-on to generate documentation of the codes - *epydoc*[1]. The link to the technical documentation is here `http://www.stud.hig.no/~090285/pydoc/`.

It makes use of the comments written in the script. It can also show the documentations of the imported modules so that a future developer would know all the technical aspects of the script.

3.1 Author's Note

I think I have broken some major convention for implementing a program in Python. The first one is when I try to import python modules. Not only did I put a 'hack' in the program itself so that it modifies the PATH variable if your running the API in Windows, but also because the program will try to install the python modules by itself. The 'hack' is so that it can try to find python modules in other common directories where python modules are usually installed. This will still not ensure the success of finding the modules, since a user can install a module in a totally different path than the default ones without changing the PATH or PYTHONPATH variable. Also the program will nonetheless fail in installing the modules if you are running the script in a non Linux based OS. So when the script fails in either hacking and forcibly installing the modules it will terminate and should show a window error message.

Anyway, this is not the normal way to do this. What I mean is that, you don't normally put the installation process (or some of it) inside the program itself. It is more normal to have an installer program or just make a documentation of how to install and use the program. That is why I have made the readme-file and added a document of how to install python modules in different OS.

My only reason for adding the hack and the force install was to automate everything as well as a try to make the program compatible in both Mac, Windows and Linux OS. The normal convention for this is to make different versions for different OS. This is why I have said that the latest version is made in and mainly for Linux based OS. Yet the program as of the latest version should be runnable and workable in all the 3 OS-es mentioned (I believe!).

The GUI frame of the program is a recycle from my old python programs. This is why there are a couple of variables that aren't used. I didn't have time cleaning them. There are also a few variables whose names are not intuitive or easy to identify with.

[1] `http://epydoc.sourceforge.net/index.html`

Some of the codes I have recycled were also based from an object-oriented program. This is why a programmer who reads my code might get confused on why I use a certain approach in doing 1 thing in some parts while I use a different approach in doing the same thing in another part. Like for example when I use the global dictionary/hashmap in most of the functions in the program, I simply access the dictionary by saying `self.dic['some key'] = 'some value'`. While in `def add_file(self, filepath, arg)` I use `arg` to access `self.dic`. This is not necessary since `self.dic` is not a protected or hidden variable.

Ideally it would be better to make a separate class for the GUI part and another one for the KML-generation part.

3.2 Additional Features

In addition to the GUI, I have implemented 3 other additional features not included in the main project specification in the program;

Watcher Mode : This sets the script in 'watcher mode'. It means it will watch the chosen directory for new images the user might put in it. It will automatically makes a new KML file when it detects new images.

Upload Files : This uploads your images and KML-file into a remote server. This relies that both USERNAME, HOSTNAME and DESTINATION_FOLDER have valid data. This function only works for UNIX-based systems (Linux, Mac OS, Cygwin) since it makes use of the API 'scp' for this version.

Show KML : If the images and KML-file are properly uploaded to a remote server, this function will call Google Maps and show you the resulted KML-file in it.

4 Conclusion

4.1 Process

I have managed to do what I set out to do. I started out with an activity plan on how to approach the project [2], and I proceeded on following it.

In the beginning I was on time and following my plan, but the programming and implementation took a lot more time than I expected. It wasn't because I had a particular difficulty and problem with programming, but rather because I had so much idea that I wanted to add and try in the program. I also wanted to make the program flawless - its functionality, coding, comments and other aspects involved that I found myself doing more and more things.

My educational background before coming to HiG was highly influenced by the different programming paradigms and programming in general. This is fine in itself, but the problem is that I have been 'out of practice' with programming for 2 years now. My effectiveness was really poor when I started programming. I wanted to use everything I learned before, but most of them I remember vaguely. I had to revisit my old notes and codes as well as online tutorials to relearn even the basic things.

I also wanted to try different modules, approach in solving problems, tips and tricks and other related issues that seems interesting to me. All these researching and relearning took more time than programming the script itself. I was done with my first prototype in just 1 week (maybe less), and it already had all the ingredients the project description was asking for including the additional program feature. Even before I started programming, I spent a good amount of time deciding which programming language I would write my application with.

In the end I had to be satisfied with making a program that will at least work and has all the functionalities described in the project description. So no the program is not flawless or perfect as I wanted it to be. But how 'perfect' can you make a program really? I found that very hard to answer. It was a bitter-sweet feeling for me. I knew most of the things that I needed to use for a good program but couldn't put it in, and at the same time I'm just happy with my working program.

I'm also satisfied with publishing my images and KML-O lite under CC license. I also stayed true in using only free and open source tools in developing my application. Finally I hope this can be a good contribution to the open source community.

4.1.1 Documentation

This project was a good chance for us to practice writing in LaTeX. At start it was ironic since the idea of LaTeX is to make (academic) documentation easy, but what I found out (and I think the others as well) was that it was very hard to get used to writing in it. It was easier to write in Microsoft Words for example.

I have downloaded and used HiG's template written by Ivar Farup. This is the same format they recommend for the master thesis. I had to do some tweaking with the templates to adjust them to my preference.

Although I still cannot say that I can use LaTeX more efficiently and effectively in writing documents than in Microsoft Words for example, I do appreciate the functionality and technique it provides. I also believe that in proper time and perhaps motivation I could use it more efficiently and effectively. If anything it is always fun for me to learn new things especially programming related. My biggest headache in using LaTeX is that i have to recompile every time I want to see the the pdf-file or just want to test if my LaTeX-codes are working. The compilation time is time consuming!

With that said, I only wrote about 80% of my documentations in LaTeX. My project proposal was written in LyX which is based in LaTeX , but it doesn't show you the codes but merely the result. My program instruction, *readme.txt/readme.html*, is written in plain html format and text format for simplicity. And for the full technical specification and codes of my program, I used *epydoc* to make a whole Web page for it instead of writing them in chapter 3.

As for my website, I regret to inform that it is very minimal and simple. Since my project was not focused on web application (i.e. mashup web application), but rather making a tool that can be used for mashup applications and services, I decided not to concentrate on my project's website[1]. Instead I wanted to use what time I had left to making this document and some minor tweaking with the program.

4.1.2 My Choices

Firstly I have chosen Ubuntu as my main platform in developing this API. It is free and is `Linux` based which is in general a good platform for programmers. It has most of the necessary tools and applications for programmers.

Secondly I chose `Python` as my scripting language since I had good experience with it before. It is elegant, has good amount of official pre-installed modules but also modules that the open source community has developed.

With `Python` you don't need to compile your codes like in Java and C. This way a program-

[1] `http://www.stud.hig.no/~090285/kmlo.html`

mer/developer can already see the codes from the program itself just by opening it with a text-editor such as *emacs*.

Because of the readily available Python modules and its continuing growth, users don't need to call external applications/programs for certain need(s). In my case this is event handling, image manipulation, calling webbrowser, etc..

In general the idea behind Python is "Less is More". Meaning that less code is more attractive and better. If I had written this application is Java, I think I would have 3x or more lines of codes than I have in Python.

I also chose to make the final version of KML-O with graphical user interface. This is to make the user interaction with the application as easy as possible. It is also easier to visualize future implementations of the program if you have an image of the application and what it does.

Whole throughout this document I have been using the terms application, API, program and script in defining my KML-O. I use application/API when I am talking about KML-O as a computing tool, - a program when I want to talk about it in a programming context, - and as a script when I want to specify its automation function.

4.2 Product

I am happy with KML-O and the KML-file it produces. I have achieved my goals and even added some new functions. Again, I would have loved to make my codes and the program in general more finesse had I gotten the time.

The KML file is showing and using KML's basic tags. For this project's purposes it is enough, but in the future I'd like to see this expanded.

Read also Chapter 3.1, about my additional notes about the program.

4.3 Future Work

Things to do in the future:

- Make different versions for different OS
- Make a separate installer package/program
- Use event-handler module such as PYINOTIFY
- Make a similar function as `def upload_file()` for file uploads for Windows
- Consider asking the user for image resizing before uploading the images
- Make a better code for choosing image formats. Consider using regular expression pattern

matching[2] [3]

- Consider expanding the KML file and using other functions of KML
- Make better website for the project
- See Chapter 3.1 for understanding other flaws in the program
- Add forgotten references in the bibliography for this document
- Finish and add the appendix for this document
- Add more images

4.4 Final Words

I have learned a lot in doing this project as well as having the course itself. Firstly I am more aware of how digital workflow is applied in real life. Secondly I learned a lot about different Python modules and I got to practice in being a programmer again. Thirdly I gained valuable knowledge in using LaTeX and got some insight on how I would go about in writing any future master thesis. Fourthly is that I learned the basics of KML and what kind of applications and perhaps mashup projects that may use it. I am also now more aware of the value of open source development and movement as well as intellectual property rights.

I can actually see this turning into a master thesis, and I it is fun to think about the future aspects of the application and project itself.

4.5 Bibliographic References

Some other references I have used which I haven't link are; The KML handbook [3], Installation Guide for python modules [4], XML tutorial [5], Digital workflow site [6], and my project proposal [7].

This is still incomplete because I have read and used quite many online articles and tutorial I wanted to put here, but some of them I have forgotten and also I don't have time for looking for them again now. I apologize terribly for this.

[2] My solution for this atm. is based mainly on the fact that I wanted to make the program usable in both Windows and Linux. Windows OS is a case-insensitive OS so choosing either '*.JPG' or '*.jpg' will give the same results. While in linux, which is a case-sensitive OS, it will differentiate between the two.

[3] I also think that the Image module has a check for this, but I haven't looked at it yet.

Bibliography

[1] Gonzales, I. J. D. *readme.txt*, 2009. (Program instruction for KML-O lite).

[2] Gonzales, I. J. D. 2009. Activity plan.xls. (Activity Plan written in Excel).

[3] Wernecke, J. 2008. *The KML Handbook: Geographic Visualization for the Web*. Addison-Wesley Professional.

[4] Ward, G. *inst.pdf*, 2000. (A manual for installing python modules).

[5] K, T. 1_xml_lesson_intro. Technical report, Høgskolen i Gjøvik.

[6] Refsvik, K. A. Imt4891 digital workflow fundamentals.

[7] Gonzales, I. J. D. 2009. ianjamesdanielgonzales_project_proposal_v1.3. (The pre-project report).