STUDY AND DEVELOPMENT OF AN AUTOPSY MODULE FOR AUTOMATED ANALYSIS OF IMAGE METADATA

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Jordi Doménech Fons

In partial fulfilment

of the requirements for the degree in

TELECOMMUNICATIONS TECHNOLOGIES AND SERVICES ENGINEERING

Advisor: Josep Pegueroles Vallés

Barcelona, June 2021

Abstract

The importance of cybersecurity is raising every day with the development of new technologies. Consequently, cybercrimes are notably increasing and, more than ever before, qualified people are needed to protect the rest of users from these type of infractions. Digital forensics is a very important part of the cybersecurity world. Furthermore, the ultimate goal of a digital forensics investigation is to preserve, identify, acquire and document digital evidence to be used in the court of law. As a result, this thesis dives deeply into the digital forensics field, especially in the analysis of image metadata exploring one of the most important software tools used by digital forensics investigators: *Autopsy*.

One of the gaps detected in the Autopsy framework is the lack of a very demanded feature consisting in the analysis, extraction and filtering of image metadata. For that reason, the main contribution done in this thesis consisted in designing and developing a software-based Autopsy plug-in module capable of analyzing image metadata and enable the user to perform the filtering of image data bases based on a set of configurable rules.

Thanks to the work done in this project, investigators will not be forced to run another external software to execute such specific processing function since, from now on, this feature is integrated into the Autopsy framework through this new available open source plug-in.

Resum

La importància de la ciberseguretat augmenta cada dia amb el desenvolupament de noves tecnologies. En conseqüència, els delictes cibernètics augmenten notablement i, més que mai, calen persones qualificades per protegir la resta d'usuaris d'aquest tipus d'infraccions. La forense digital és una part molt important del món de la ciberseguretat. A més, l'objectiu principal d'aquest camp consisteix en preservar, identificar, adquirir i documentar proves digitals per poder-les utilitzar després als tribunals de justícia. Com a resultat, aquesta tesi aprofundeix en el camp de la forense digital, especialment en l'anàlisi de metadades d'imatges fent us d'un dels programes més importants utilitzats pels investigadors: *Autopsy*.

Una de les carències detectades a Autopsy és la manca d'una característica molt demandada que consisteix en l'anàlisi, extracció i filtratge de metadades d'imatges. Per aquest motiu, la principal contribució feta en aquesta tesi consisteix en dissenyar i desenvolupar un mòdul capaç d'analitzar metadades d'imatges i permetre a l'usuari realitzar un filtratge de bases de dades d'imatges basant-se en un conjunt de regles configurables.

Gràcies a la feina feta en aquest projecte, els investigadors no es veuran obligats a utilitzar un altre programa extern per executar aquesta funció de processament tan específica ja que, a partir d'ara, aquesta característica s'integra a Autopsy mitjançant aquest nou mòdul disponible de codi obert.

Resumen

La importancia de la ciberseguridad aumenta día a día con el desarrollo de nuevas tecnologías. En consecuencia, los ciberdelitos están aumentando notablemente y, más que nunca, se necesitan personas calificadas para proteger al resto de usuarios de este tipo de infracciones. La ciencia forense digital es una parte muy importante del mundo de la ciberseguridad. Además, el objetivo final de una investigación forense digital es preservar, identificar, adquirir y documentar pruebas digitales para su posterior uso en los tribunales. Como resultado, esta tesis profundiza en el campo de la ciencia forense digital, especialmente en el análisis de metadatos de imágenes explorando una de las herramientas de software más importantes utilizadas por los investigadores: *Autopsy*.

Una de las carencias detectadas en Autopsy es la falta de una función muy demandada capaz de analizar, extraer y filtrar metadatos de imágenes. Por ello, el principal aporte realizado en esta tesis consiste en diseñar y desarrollar un módulo en Autopsy capaz de analizar metadatos de imágenes y permitir al usuario realizar un filtrado de bases de datos de imágenes en base a un conjunto de reglas configurables.

Gracias al trabajo realizado en este proyecto, los investigadores no se verán obligados a usar otro software externo para ejecutar una función de procesamiento tan específica ya que, a partir de ahora, esta característica se integra en Autopsy a través de este nuevo módulo disponible de código abierto.

"It ain't about how hard you hit, it's about how hard you can get hit and keep moving forward."

Rocky Balboa (S. Stallone)

Acknowledgements

First of all, I would like to thank my family; specially my mum, my dad, my brother and my uncles. They have always helped and listened me during the good and bad moments, giving me great ideas and support to end my bachelor degree with this fantastic thesis.

I would also like to give special thanks to my project supervisor, Josep Pegueroles Vallés. He has given me the opportunity to start learning about the cybersecurity field, specially digital forensics; a specialization I always found interesting as well as necessary to nowadays society. Apart from that, he always supported and advised me during the last years of the bachelor degree and, of course, during the final degree thesis. Without him, this project would not have been possible.

Last but not least, I am thankful to the Software Engineer Phil Harvey, author of ExifTool, that let me use his excellent software in my thesis with no inconvenience.

Revision history and approval record

Revision	Date	Purpose
0	28/02/2021	Document creation
1	06/03/2021	Document revision
2	30/03/2021	Document revision
3	19/06/2021	Document revision
4	21/06/2021	Document delivery

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Jordi Doménech Fons	jordi.domenech.fons@estudiantat.upc.edu
Josep Pegueroles Vallés	josep.pegueroles@upc.edu

Written by:		Reviewed and approved by:	
Date	20/06/2021	Date	20/06/2021
Name	Jordi Doménech Fons	Name	Josep Pegueroles Vallés
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

1. Introduction

As the world continues to digitize, the dependence on technology in the personal and professional live is increasing fast: almost everyone is involved in technology in one way or another. However, this rapid shift to the digital age has been accompanied by a growing increase of cybercrimes, that is crimes involving basically a computer and a network. Fighting this type of infraction requires tech-savvy sleuthing, and digital forensics does exactly that.

Digital Forensics, also known as computer forensics, is the process of investigating crimes committed using any type of computing device (such as computers, servers, laptops, cell phones, tablets, digital camera, networking devices, Internet of Things (IoT) device or any type of data storage device) and also extends data in transit which is transmitted across public or private networks. The ultimate goal of a digital forensics investigation is to preserve, identify, acquire and document digital evidence to be used in the court of law.

When we talk about digital forensics there are a lot of tools we can use, but in this project we are going to focus on one of the most important digital forensic software: Autopsy. Autopsy is a digital forensics platform and graphical interface to The Sleuth Kit and other digital forensics tools. It is used by law enforcement, military, and corporate examiners to investigate what happened on a computer. You can even use it to recover photos from your camera's memory card.

Autopsy was designed to be an end-to-end platform with modules that come with the software to make the analysis of data more portable. Apart from the included modules, you can add others that are available from third-parties and even more you are able to create your own personalized modules with Python or Java. Taking into account this huge characteristic, the main goal of this project is going to be the development of an Autopsy module to analyze any type of image metadata, but especially the one called IPTC; the current Autopsy modules are not able to do it.

As I explained above, the modules included in Autopsy by default are not able to analyse any type of image metadata apart from EXIF, this is an important lack, because many other types of image formats including other metadata standards are being used extensively nowadays.

EXIF, the Exchangeable Image File Format, is a standard that defines the formats of image, audio, and metadata tags used by cameras, phones and other digital recording devices. Depending on the file type and the authoring tool (the application or the capturing device), different types of metadata are recorded. For example: capture and last edited date and time stamps, GPS location coordinates, device information including manufacturer and model, capture information including lens type, focal range, aperture, shutter speed, flash settings, etc.

Taking into account this type of metadata is stored mostly automatically by the device, the probability of having EXIF data inside the image is high enough. Accordingly, having a tool able to analyze it should be a must.

On the other hand, the IPTC Photo Metadata Standard is the most widely used standard to describe photos because of its universal acceptance among new agencies, photographers, photo agencies, libraries, museums and other related industries. It structures and defines metadata properties that allow users to add precise and reliable

data about images. For example: creator of the image, description, people shown in the image, where and when it was taken, copyright information, contact details, etc.

In that case, most IPTC metadata have to be stored manually by the user, therefore, is less common to find IPTC data inside an image in comparison with EXIF. But, IPTC metadata is much more valuable than any type of metadata because of the information it provides, specially to the professional investigators. Knowing the IPTC information you have a very clear idea of the image description. Furthermore, fake news and fake images are growing very fast in our digital world, for that reason the importance of having IPTC metadata inside an image to describe it concretely is more important nowadays. This fact is well known by professional photographers: over the 80% of them add metadata to protect their images.

Due to the importance of IPTC, this Final Degree Thesis will be focused on the development of a new Python module able to analyze all type of metadata found in an image, specially the IPTC. This could be a notable advantage for Autopsy users when they want to go deep in the investigation of any image and its characteristics and find evidences to use them in a court of law.

Thankfully to the development of the module, investigators will be able to extract all the capable metadata stored inside the image while using the Autopsy program during their investigations and they will not be forced to run another external software only to analyze image metadata.

In the next figure we can see a diagram of the progress that any investigator follows in a professional digital forensic case and how it would be developed taking advantage of our module.

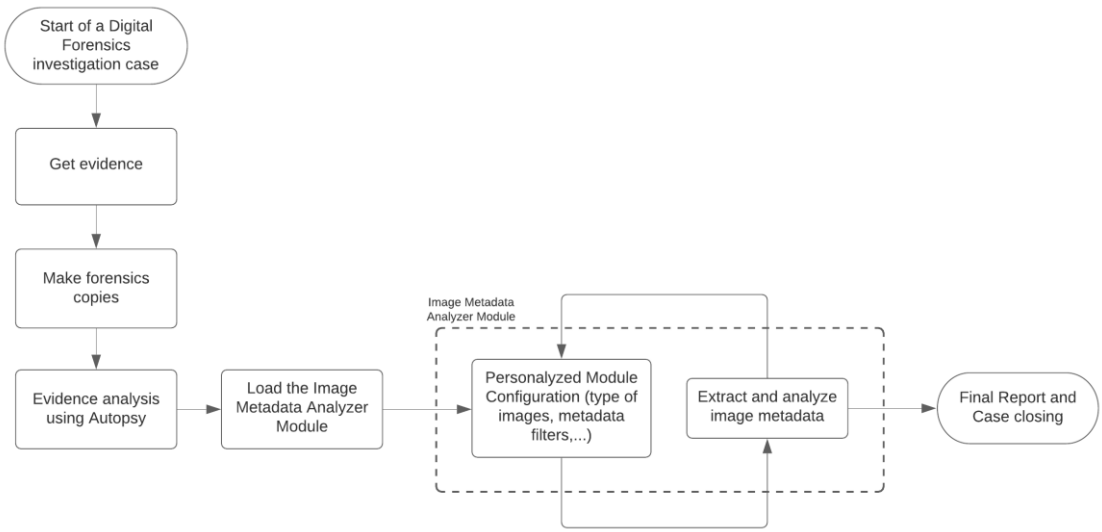


Figure 1. General flux diagram

The next chapters of this document are structured as follows:

First of all, a description about the state of the art of digital forensics will be explained together with a basic review of Autopsy software. Secondly, we will review the different metadata formats we can find in the literature and their nature. Next, we will continue the project understanding and studying how Autopsy modules work and creating a basic

"Hello World" Python module. Finally, the development of the Autopsy module for automated analysis of image metadata will be explained; we will also see how it works and the usefulness it has in real world digital forensics investigations.

1.1. Objectives

As we said, the main goal of the project is the development of an Autopsy module able to analyze IPTC metadata. But, this main objective can be decomposed in the next ones:

- Familiarization with the development of Python Autopsy modules.
- Development of a module for automated analysis of image metadata.
- Module testing, validation and documentation.

In the next section we are going to propose a Work Plan and a Gantt diagram to achieve these goals.

Apart from the general objectives, other different goals are set: Technical Objectives, Work Planning Objectives and Personal Objectives. These targets are as important as the main goal of the project.

1.1.1. Technical Objectives

- Learn the first ideas about cybersecurity and digital forensics world.
- Image metadata study: types of metadata we can find in an image, how it is structured, how can we visualize them, why they are so important in digital forensics investigations,…
- Being familiar with Autopsy tool and learn how to develop personalized modules.
- Expand the knowledge of Python programming language together with Jython and learn how to work with a specific Python IDE: SPYDER.

1.1.2. Work Planning Objectives

- Learn to do a final degree thesis, that is, plan and sequence it correctly, order the ideas clearly for a good comprehension of the message we want to transmit to the reader.
- Learn to do a presentation, summarizing the principal ideas with an adequate attitude and precisely vocabulary.
- Learn to search useful information in books or internet. Choose only necessary information and discard what is not important or irrelevant.

1.1.3. Personal Objectives

- Achieve a good knowledge and self-learning that help me grow in the professional and personal field.
- Value the effort, constancy and time investment made doing this project and try to send a paper of the work done to a forensics journal.
- Enjoy doing what I really like.

1.2. Work Plan

Background Research	WP ref: WP1
Major constituent: Research	Sheet 1 of 4
Short description: Being familiar with Digital Forensics' world, image metadata and Autopsy software.	Planned start date: 15.02.2021 Planned end date: 15.03.2021
Internal task T1: Research information about Digital Forensics. Internal task T2: Research information about the different types of image metadata we can found in an image. Internal task T3: Install Autopsy and understand how it works. Internal task T4: Write the Thesis introduction. Internal task T5: Write a review about the different metadata formats we can find in the literature and their nature. Internal task T6: Write a basic review about Autopsy software.	

Table 1. Work Package 1

Autopsy Modules	WP ref: WP2
Major constituent: Software	Sheet 2 of 4
Short description: How to develop modules in Autopsy, being familiar with the Sleuth Kit library and implementation of a basic "Hello World" module.	Planned start date: 16.03.2021 Planned end date: 04.04.2021
Internal task T1: Research information about the implementation of Python modules in Autopsy. Internal task T2: Implementation and testing of a basic Autopsy module. Internal task T3: Write a report about the work done.	

Table 2. Work Package 2

Development of the Module	WP ref: WP3
Major constituent: Software	Sheet 3 of 4
Short description: Development of a module for automated analysis of image metadata.	Planned start date: 05.04.2021 Planned end date: 08.05.2021
Internal task T1: Development of the Python module to analyze image metadata. Internal task T2: Module testing and validation.	

Internal task T3: Write a report about the work done.

Table 3. Work Package 3

Final Documentation	WP ref: WP4
Major constituent: Documentation	Sheet 4 of 4
Short description: Write and review the Final Degree Thesis documentation.	Planned start date: 09.05.2021 Planned end date: 20.06.2021
Internal task T1: Write and review the final documentation to present it to UPC and tribunals. Internal task T2: Write a paper of all the work done to present it to a congress.	

Table 4. Work Package 4

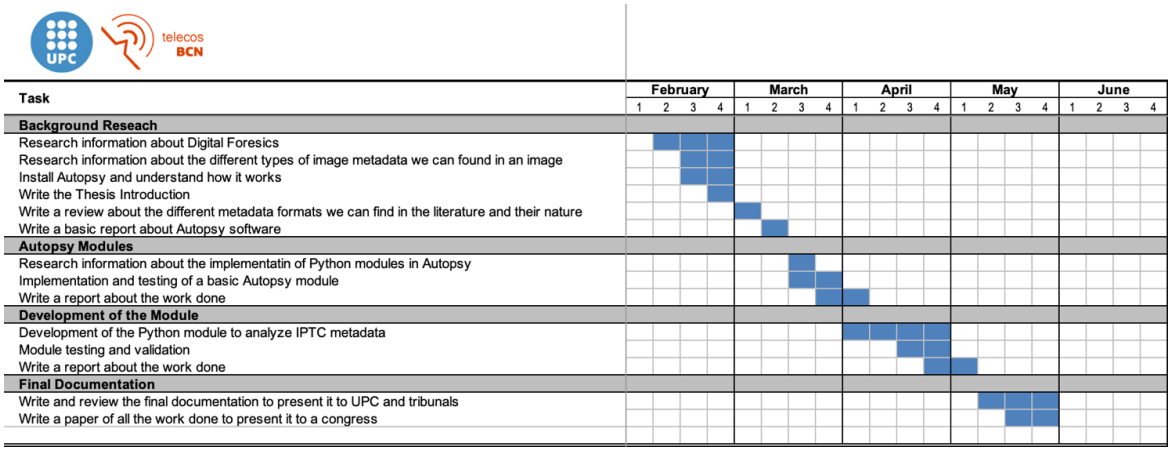


Figure 2. Initial Gantt Diagram

1.3. Eventualities and deviations from the initial plan

Despite that thankfully we did not have any considerable incident and eventuality, we have added one more work package to work in between the WP2 and WP3 called ExifTool:

ExifTool	WP ref: WP2.5
Major constituent: Software	Sheet 5
Short description: Study of the great software *ExifTool* to analyze image metadata. How can we integrate it to be capable of working with Autopsy?	Planned start date: 23.03.2021 Planned end date: 04.04.2021
Internal task T1: Research information about *ExifTool.* Internal task T2: Write a basic script about *ExifTool* with Python for the proper	

integration of it in an Autopsy module. Internal task T3: Write a report about the work done.

Table 5. Work Package 2.5

This new modification will affect the dates and consequently the Gantt diagram. It will be as follows:

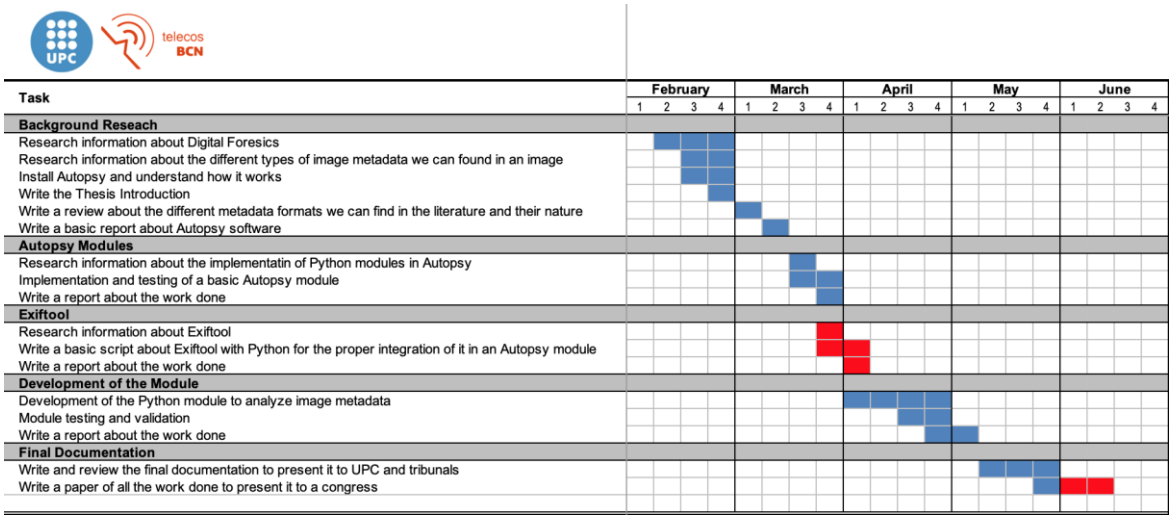


Figure 3. Updated Gantt Diagram

2. State of the art of the technology used in this thesis

2.1. Autopsy

The Sleuth Kit is a digital forensics open source tool which works with the command line and allows you to analyze disk images and recover files from them. Therefore, it is not user friendly and needs expert handling. For that reason, it is used behind the scenes of Autopsy: an easy to use, GUI-based program that allows you to efficiently analyze hard drives, smartphones, logical files,...

Figure 4. Autopsy's logo

The principal benefits that Autopsy provides are the user friendly interface, the clear detail reports provided at the end of the investigation and, last but not least, the plug-in architecture that allows you to find add-on modules or develop custom modules in Java or Python to personalize and add new features in your professional investigation case.

Besides that, some of the available forensics modules that allow the systematization of the analysis are:

- **Time analysis**: used to see when a computer was used or which events occurred before or after a given event.
- **Hash filtering**: used to identify files that are 'known' and can be ignored or are 'known bad' and raise awareness.
- **Keyword search**: extracts text from all kind of files to be easily searched.
- **Data carving**: carves files from unallocated space in the data source, helping in that way to discover files that were deleted.

In the next sections I will explain you a little example of how Autopsy works analyzing a Disk Image of a used Pen Drive but supposedly empty now.

2.1.1. Creating a new case

When running Autopsy, you can create a new case or open an existing case. In this example we are going to create a new one.

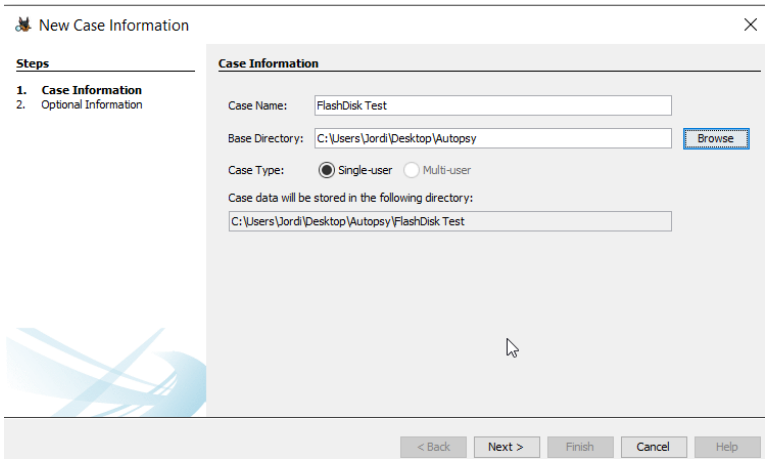


Figure 5. New Case Information in Autopsy

When creating a new case, it has to be provided a name for the case and a directory to store the case, as well as the name of the examiners and other optional information.

2.1.2. Adding the data source

After creating the new case and filling the Case Information, we are going to add the Data Source. We can see that we have different options of Data Source:

- **Disk Image or VM File**: A file that is byte-for-byte copy of a hard drive, media card or Virtual Machine.
- **Local Disk**: Local storage device. It can be a local drive, a USB-attached drive, etc.
- **Logical Files**: Files or folders that are on the computer without having to put them into a disk image.
- **Unallocated Space Image File**: Any type of file that does not contain a file system.
- **Autopsy Logical Imager Results**: The results from running the logical imager. The logical imager allows you to collect files from a live Windows computer.
- **XRY Text Export**: The results from exporting text files from XRY.

We are going to select the Disk Image we have created from the pen drive called *FlashDisk01.001*.

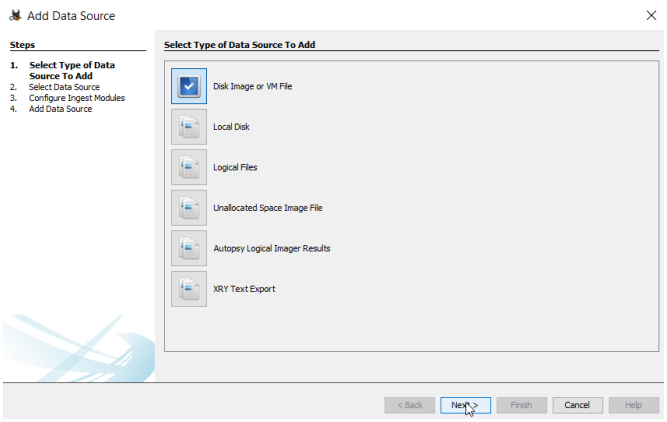


Figure 6. Selection of Data Source Type in Autopsy

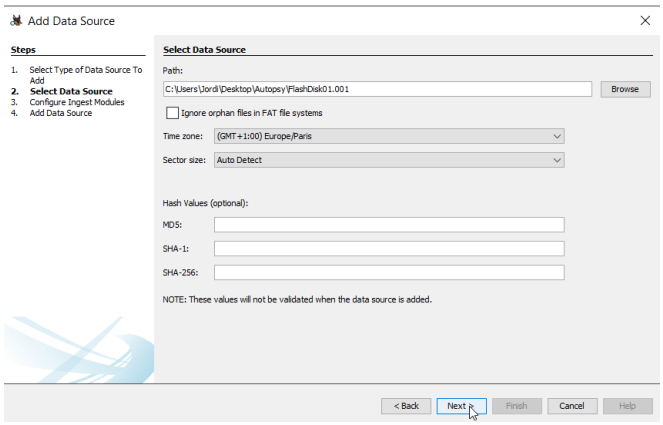


Figure 7. Selection of the Disk Image to analyze in Autopsy

2.1.3. Ingest Modules

Ingest Modules analyze the data in a data source (in our case the *FlashDisk01.001*), they are configured to find user content quickly and run in the background. They are used to post the results on the Blackboard, so you can find them in the "Results" area of the Tree Viewer explained in the next section.

There are different Ingest Modules available that Autopsy gives to us, the most important are: Recent Activity, Hash Lookup, File Type Identification, Extension Mismatch Detector, Embedded File Extraction, Picture Analyzer, Keyword Search, Email Parser, Encryption Detection, Interesting Files Identifier, Central Repository, PhotoRec Carver, Virtual Machine Extractor, Data Source Integrity, Drone Analyzer, iOS Analyzer, Android Analyzer and GPX Parser.

For further information see *Appendix I: Ingest Modules in Autopsy*.

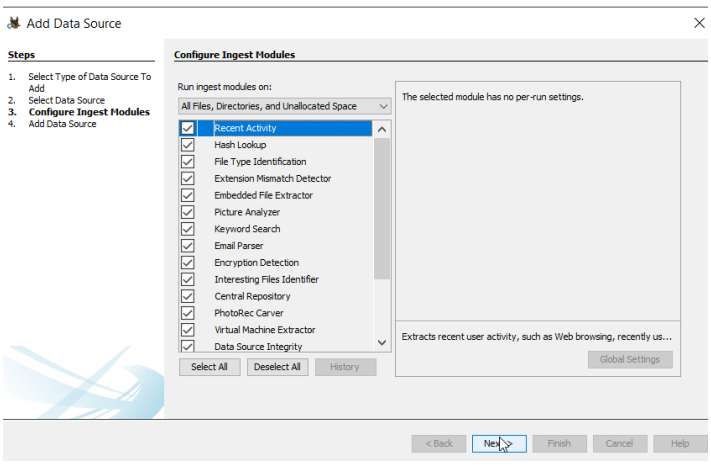


Figure 8. Configuration of Ingest Modules in Autopsy

2.1.4. User Interface Layout

As shown, the user interface is divided in five main areas: Tree Viewer, Result Viewer, Content Viewer, Keyword Search and Status Area.

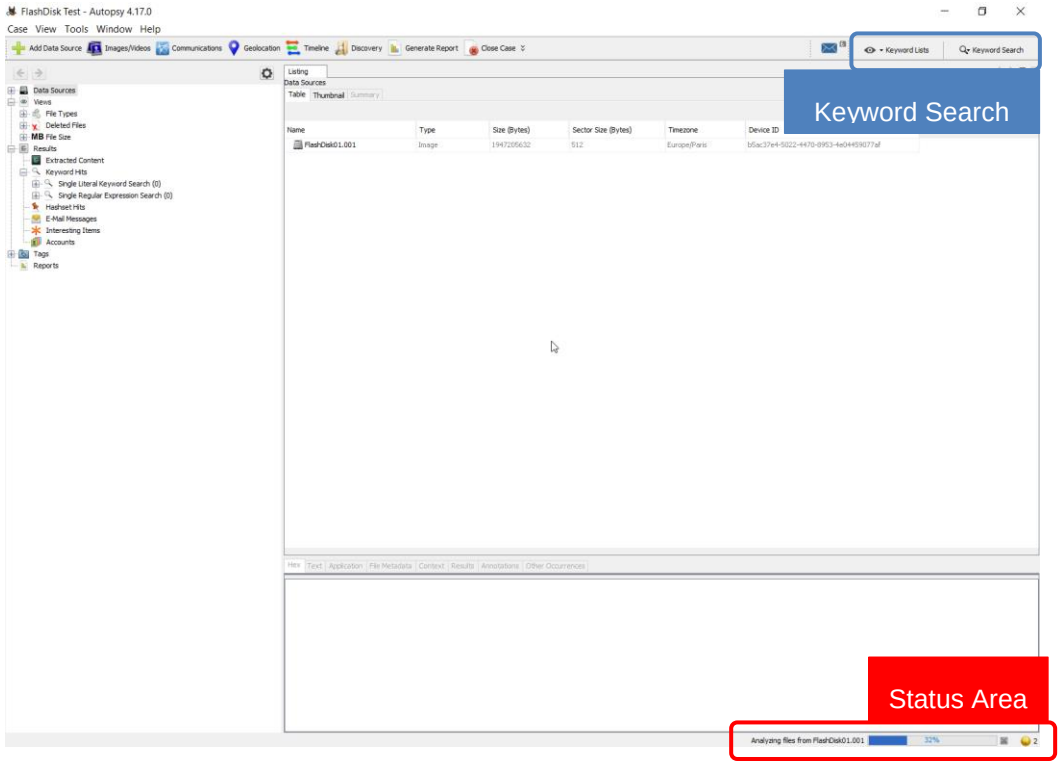


Figure 9. Autopsy User Interface Layout (I)

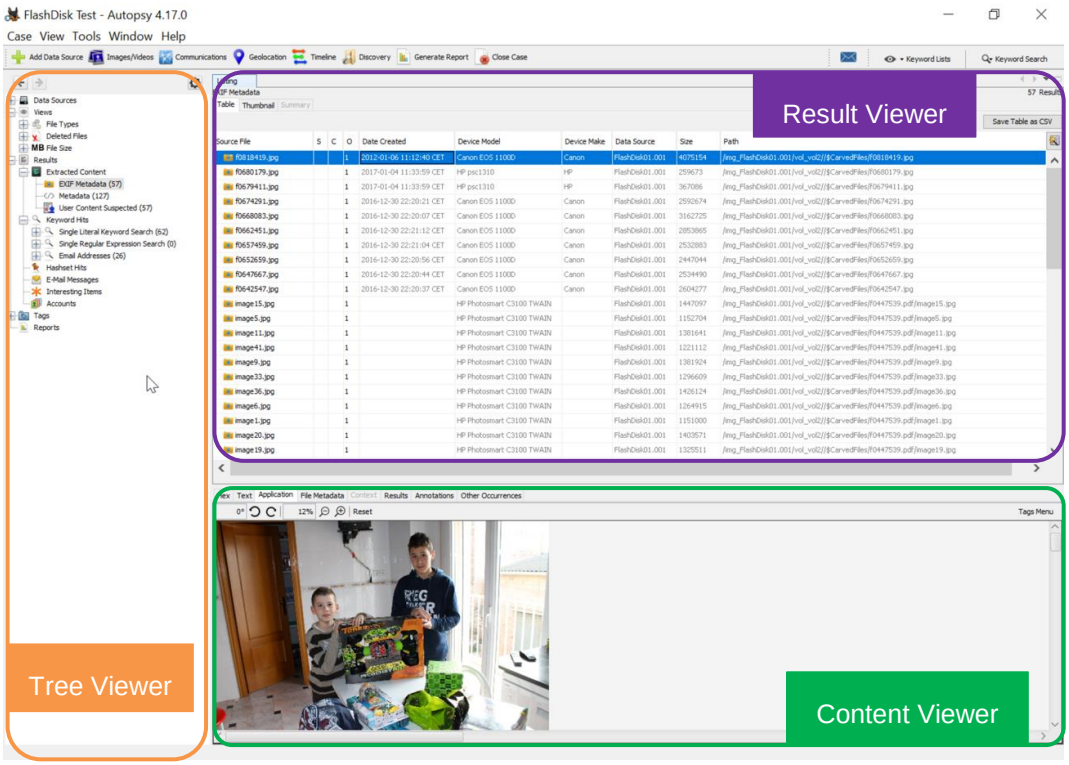


Figure 10. Autopsy User Interface Layout (II)

The **Keyword Search** allows the examiner to search for keywords in the data source. It uses the Keyword Search Module.

The **Status Area** is the region where can be seen the status of the data source analysis with a charger bar and a panel (when clicking on the yellow dot) with notifications.

The **Tree Viewer** is on the left side of the Autopsy user interface and it contains four main areas:

- Data Sources: shows the sources added to the case.
- Views: shows specific type of files from the data sources.
- Results: shows the results of the ingest modules.
- Reports: shows the reports generated.

The **Result Viewer** is on the top-right and shows the lists of files and their corresponding attributes. There are two ways of displaying the results:

- Table Results Viewer: displays the data catalog as a table with some details of each file (see image above).
- Thumbnail Results Viewer: shows the data as a table of thumbnail images. Mostly used when we want to see the image itself.

The **Content Viewer** is on the bottom-right side and displays the specific selected file in a variety of formats and options to describe the file such as Hexadecimal, Text, Application, File Metadata, Context, Results, Annotations and Other Occurrences.

We can also see that there are a bunch of files detected in the pen drive that remember, it was supposedly empty. The reason of that is the usage of the *Carver modules*, these modules have carved the deleted files from the unallocated space found in the data source.

2.1.5. Adding your own modules

As we said, one of the most important properties of Autopsy is the possibility to add your own personalized Ingest Modules either in Java or Python (Jython).

Autopsy is built on top of NetBeans Rich Client Platform, which makes easy to work with a plug-in environment. For that reason, if you want to develop a module in Java, you will have to download NetBeans and work from there.

On the other hand, Autopsy also uses Jython to enable Python scripting. Jython is like Python but converts the script into Java byte code to be perfectly understandable by Autopsy. Using it is very easy and allows you to access all of the Java services and classes that you need.

To add Java or Python Modules you have the following options in Autopsy:

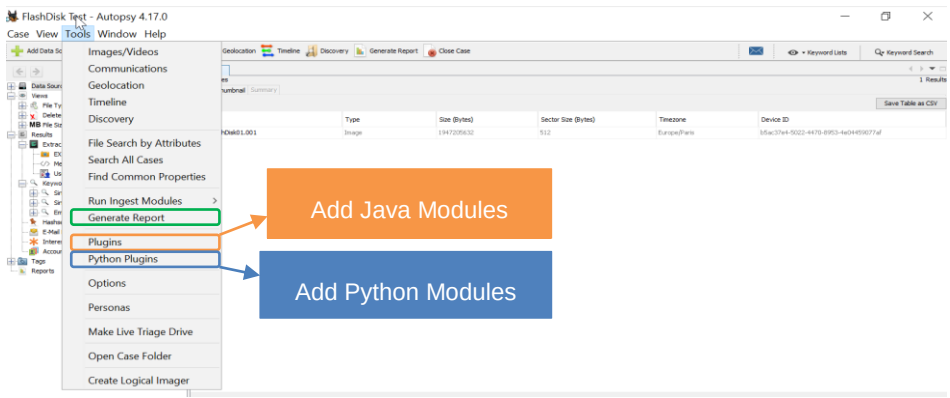


Figure 11. How to add Java/Python Modules in Autopsy

2.1.6. Generating a Report

In order to create a report, the user have to choose the “Tools” option and select “Generate a Report” (see *Figure 11*).

Different types of Report Modules can be chosen as shown in the *Figure 12*.

After selecting the type of report we have to choose which information we want to include in the report.

The examiner is able to select “All Results”, in that case you are going to include in the report all the results found by the Ingest Modules, however, you can then optionally choose which type of data from the Ingest Modules you want to include in the report (“Choose Result Types…” button).

If you choose “Tagged Results”, you can restrict the files and results that appear in the report to only those that are tagged with the tags you select.

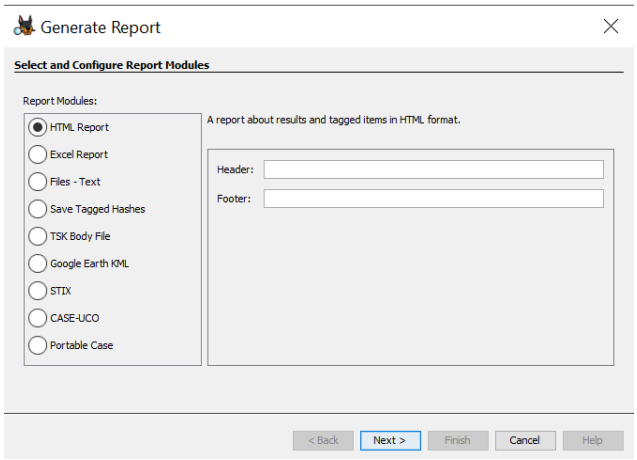


Figure 12. Select the type of Report in Autopsy

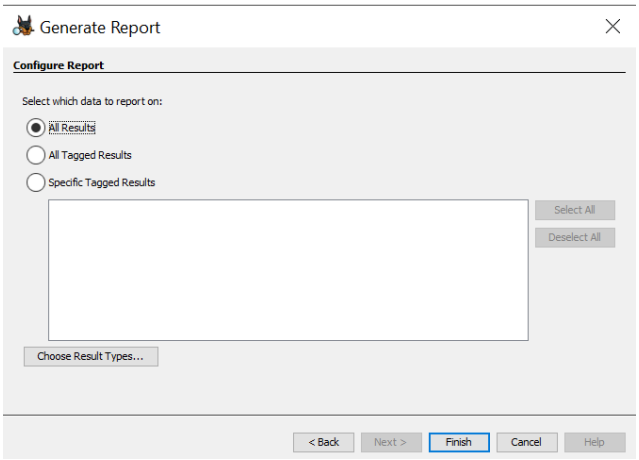


Figure 13. Configuration of the Report in Autopsy

2.2. Image Metadata

Image metadata is a set of data describing and providing information about an image. It allows information to be transported with an image file, in a way that can be understood by other software and human users.

There are three main categories of image metadata:

- **Technical metadata**: automatically generated by the camera. It includes camera details and settings such as camera brand and model, date and time when the image was created, GPS location where it was created, aperture, focal range, shutter speed, ISO number, flash settings, etc.
- **Descriptive metadata**: manually added by the photographer or someone managing the image. It includes the name of the image creator, keywords related to the image, captions, title and comments, among many other possibilities.
- **Administrative metadata**: manually added. It includes usage and licensing rights, copyright restrictions, contact information for the owner of the image, etc.

Taking into account these three main categories, several standardized formats of metadata were created, including: Exchangeable Image File Format (EXIF), Information Interchange Model (IPTC), Extensible Metadata Platform (XMP), Dublin Core Metadata Initiative (DCMI) and Picture Licensing Universal System (PLUS).

EXIF and IPTC are the most used metadata formats nowadays. This is because knowing the information in both standards, you are able to figure out almost all the knowledge available in the image: Technical, Descriptive and Administrative metadata.

During the next sections I will explain you in more detail the EXIF and IPTC metadata formats and how Autopsy or other programs extract this type of information.

2.2.1. Exchangeable Image File Format (EXIF)

The EXIF standard was managed by the Japan Electronics and Information Technology industries Association (JEITA) and Camera and Imaging Products Association (CIPA). In particular, it is a standard that defines a set of TIFF tags to describe images used by digital cameras (including smartphones), scanners and other systems handling image and sound files. The EXIF metadata can be found in TIFF, JPEG and PSD files. But, it is not used in JPEG 2000 or PNG. Furthermore, the major part of EXIF metadata is stored automatically by the device that took the photo; for that reason, this type of metadata is mostly technical.

The information we can find extracting Exchangeable Image File Format metadata is the following and it covers four big categories:

- Date and time information. Most digital cameras record the current date and time and save this in the metadata.
- Camera settings. This includes static information such as the camera brand and model, information that varies with each image such as orientation (rotation), aperture, shutter speed, focal length, flash settings, and ISO information, etc.
- Information about localization. Only available for digital cameras that have a connected GPS.
- Description and copyright information.

2.2.1.1. EXIF in Autopsy

The *Picture Analyzer* module of Autopsy extracts EXIF (Exchangeable Image File Format) information from ingested pictures. This information can contain geolocation data of the picture, time, date, camera model and settings (exposure values, resolution,…) and other information. The module also converts HEIC/HEIF images to JPG while maintaining their EXIF information, which will be processed and saved as it would for normal JPG images.

The discovered attributes are added to the blackboard and can be seen by the user accessing to Results > Extracted Content > EXIF Metadata, in the Directory Tree of Autopsy.

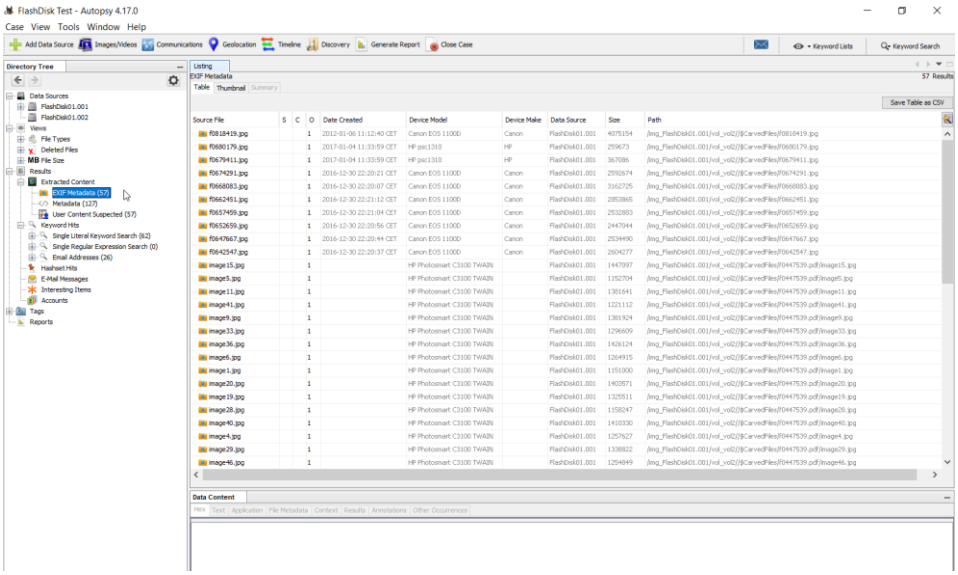


Figure 14. List of images that contain EXIF Metadata in Autopsy

In the *Figure 14* you can see all the images found that contain EXIF Metadata information carved from the same Flash Disk explained in the Chapter 2.

If we want to see the specific metadata available in an image, we have to click on the file we want to analyze and in the Data Content we will see the results.

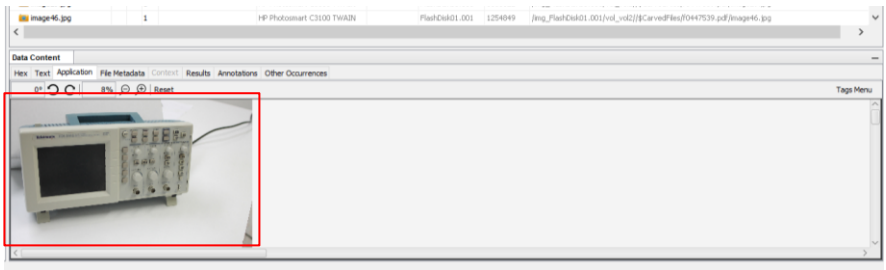


Figure 15. Data Content in an image that contain EXIF Metadata in Autopsy (I)

Figure 16. Data Content in an image that contain EXIF Metadata in Autopsy (II)

We can see in the screenshots above a preview of the image selected and the information extracted by the Picture Analyzer module; that is, the date the image was created (2016-12-30) and the time (22:20:21 CET), the device model (Canon EOS 1100D) and make (Canon).

2.2.1.2. Other programs to extract EXIF metadata

One of the most used and powerful tools to extract metadata is the *ExifTool* by Phil Harvey.

ExifTool is a platform-independent Perl library plus a command-line application for reading, writing and editing meta information in a wide variety of files. ExifTool supports many different metadata formats including EXIF, GPS, IPTC, XMP, JFIF, etc.

After installing the software, we are going to see how it works and the metadata it extracts from the same image seen in the section above.

```
MacBook-Pro-de-Jordi:3.Metadata JORDI$ exiftool f0674291.jpg
ExifTool Version Number         : 12.21
File Name                       : f0674291.jpg
Directory                       : .
File Size                       : 2.5 MiB
File Modification Date/Time     : 2021:03:02 09:51:08+01:00
File Access Date/Time           : 2021:03:02 10:11:25+01:00
File Inode Change Date/Time     : 2021:03:02 10:11:24+01:00
File Permissions                : -rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
Exif Byte Order                 : Little-endian (Intel, II)
Make                            : Canon
Camera Model Name               : Canon EOS 1100D
Orientation                     : Horizontal (normal)
X Resolution                    : 72
Y Resolution                    : 72
Resolution Unit                 : inches
Modify Date                     : 2016:12:30 22:20:21
Artist                          :
Y Cb Cr Positioning             : Co-sited
Copyright                       :
Exposure Time                   : 1/50
F Number                        : 3.5
Exposure Program                : Not Defined
ISO                             : 400
Sensitivity Type                : Recommended Exposure Index
Recommended Exposure Index      : 400
Exif Version                    : 0230
Date/Time Original              : 2016:12:30 22:20:21
Create Date                     : 2016:12:30 22:20:21
Components Configuration        : Y, Cb, Cr, -
Shutter Speed Value             : 1/49
Aperture Value                  : 3.5
Flash                           : Off, Did not fire
Focal Length                    : 28.0 mm
Macro Mode                      : Normal
Self Timer                      : Off
```

Figure 17. ExifTool extracted metadata for EXIF analysis

We can see that there is a large amount of metadata found in the image, while Autopsy only could extract few types of EXIF information. This is a clear lack for Autopsy because as investigators, we want to know as much as information as possible of the image analyzed and take advantage of any insignificant detail we can find.

But, we also have to take into account that what ExifTool did is extract all types of metadata located inside the image, not only EXIF metadata.

See *Appendix II: ExifTool analysis of f0674291.jpg* for further information about all the metadata in the image found by the ExifTool.

2.2.2. IPTC-IIM

International Press Telecommunications Council (IPTC) is a consortium of the world's major news agencies, news publishers and news industry vendors. It develops and promotes efficient technical standards to improve the management and exchange of information between content providers, intermediaries and consumers.

In 1991 they decided to create a new standard called Information Interchange Model (IIM). The Information Interchange Model is a file structure and set of metadata attributes that can be applied to text, images and other media types. Is the most widely used standard to describe photos and defines metadata properties that allow users to add precise and reliable data about images.

IIM metadata can be found into JPEG, TIFF, JPEG2000 or PNG image files. Other file formats such as GIF or PCX do not support IIM.

Although most EXIF metadata is introduced automatically by the device, IPTC metadata is introduced manually by the user. Therefore, you can find all type of descriptive properties in the image you could imagine such as who took the photo, where it was taken, who appear in the image, a title and description of the image, copyright information, contact details, etc.

2.2.2.1. IPTC in Autopsy

Unfortunately, there is still no module developed in Autopsy to extract IPTC metadata. A big lack as you can verify. There is a lot of information can be found in IPTC metadata that could be very useful and key for any digital forensics professional investigation. We are going to solve that shortage.

2.2.2.2. Other programs to extract IPTC metadata

Following the same line as we did during the explanation of EXIF metadata, the ExifTool can also extract IPTC metadata and for that reason we are going to see an example of how it works and all the types of metadata available. Furthermore, we are going to analyze the official image of the last IPTC Standard 2019.1 that includes all IPTC metadata fields defined.

Figure 18. IPTC Standard 2019.1 official image

```
MacBook-Pro-de-Jordi:3.Metadata JORDI$ exiftool iptc-image.jpg
ExifTool Version Number         : 12.21
File Name                       : iptc-image.jpg
Directory                       : .
File Size                       : 126 KiB
File Modification Date/Time     : 2021:03:02 10:01:14+01:00
File Access Date/Time           : 2021:03:02 11:06:46+01:00
File Inode Change Date/Time     : 2021:03:02 11:06:45+01:00
File Permissions                : -rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
Exif Byte Order                 : Big-endian (Motorola, MM)
Image Description               : The description aka caption (ref2019.1)
X Resolution                    : 72
Y Resolution                    : 72
Resolution Unit                 : inches
Artist                          : Creator1 (ref2019.1)
Y Cb Cr Positioning             : Centered
Copyright                       : Copyright (Notice) 2019.1 IPTC - www.iptc.org  (ref2019.1)
Current IPTC Digest             : 1a752785c1ca57e321cf228e3c3e8498
Object Attribute Reference      : A Genre (ref2019.1)
Object Name                     : The Title (ref2019.1)
Subject Reference               : IPTC:1ref2019.1, IPTC:2ref2019.1, IPTC:3ref2019.1
Keywords                        : Keyword1ref2019.1, Keyword2ref2019.1, Keyword3ref2019.1
Special Instructions            : An Instruction (ref2019.1)
Time Created                    : 19:01:00+00:00
By-line                         : Creator1 (ref2019.1)
By-line Title                   : Creator's Job Title  (ref2019.1)
Sub-location                    : Sublocation (Core) (ref2019.1)
Province-State                  : Province/State (Core) (ref2019.1
Country-Primary Location Code   : R19
Country-Primary Location Name   : Country (Core) (ref2019.1)
Original Transmission Reference : Job Id (ref2019.1)
Copyright Notice                : Copyright (Notice) 2019.1 IPTC - www.iptc.org  (ref2019.1)
Caption-Abstract                : The description aka caption (ref2019.1)
Writer-Editor                   : Description Writer (ref2019.1)
Application Record Version      : 4
XMP Toolkit                     : Image::ExifTool 11.74
Country Code                    : R19
Creator City                    : Creator's CI: City (ref2019.1)
Creator Country                 : Creator's CI: Country (ref2019.1)
Creator Address                 : Creator's CI: Address, line 1 (ref2019.1)
```

Figure 19. ExifTool extracted metadata for IPTC analysis

Also a large amount of metadata found, specially IPTC. Some examples are the Copyright information, title, country code, creator city, creator email, etc. See *Appendix III: ExifTool analysis of iptc-image.jpg* for more information about all the metadata found by the ExifTool.

Another interesting program to analyze IPTC metadata is the online tool *Get IPTC Photo Metadata* [7]. It is based on the *ExifTool* software and is an easy way to analyze metadata: you only have to upload the image on the website or paste the image URL (if we want to analyze a picture found on the internet), all this without installing any software.

In the next figures you can see the extracted metadata from the IPTC image shown before.

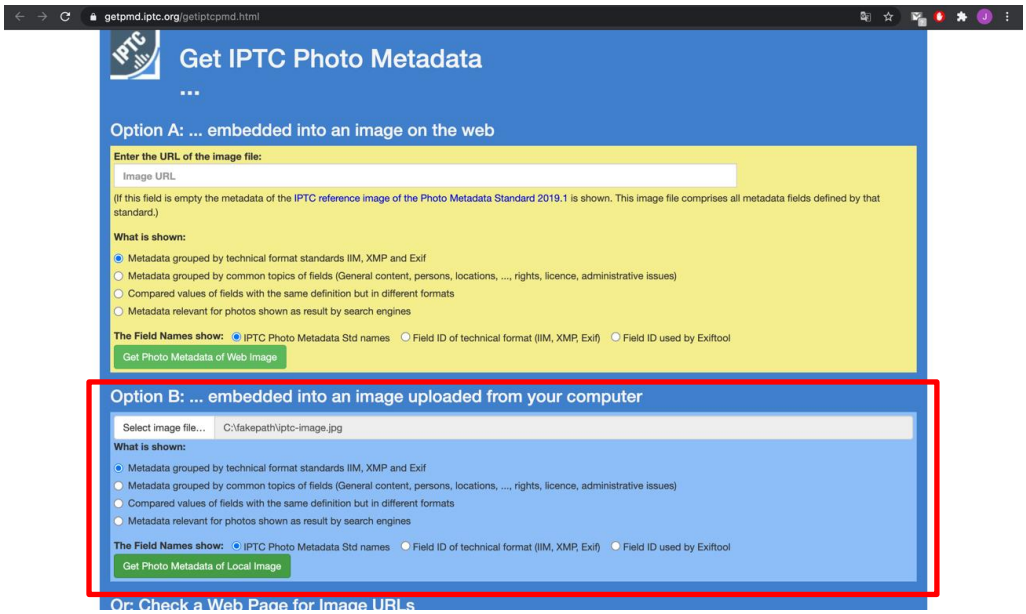


Figure 20. Get IPTC Photo Metadata main page

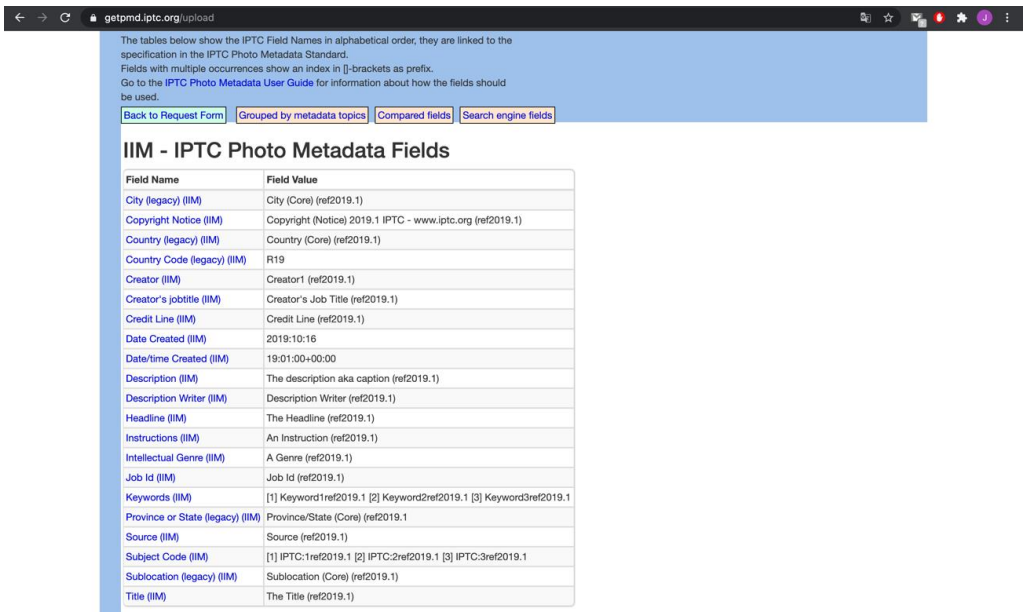


Figure 21. Extraction of IPTC data in the IPTC Photo Metadata website

3. Methodology and project development

3.1. Tools used

3.1.1. ExifTool

For the proper development of the project we are going to take advantage of the great potential of the software ExifTool developed by Phil Harvey. As we said in previous chapters, ExifTool is a platform-independent Perl library and command-line application for reading, writing and editing metadata. However, we are going to use it only for the purpose of reading all the metadata extracted from an image. ExifTool meets all our needs analysing all the metadata information in an image. There are other programs to analyse metadata but most of them are not platform independent and/or open source; the rest majority of image metadata analysis programs are based on ExifTool.

Taking into account that ExifTool software is based on Perl library and Autopsy modules are based on Python or Java, we are going to exploit the free software Python library *PyExifTool*. This library consists on an instance of Phil Harvey's ExifTool and provides a class that runs the command-line tool in batch mode and features methods to send commands to the program, including methods to extract metadata from one or more image files. This library is perfect to work together with Autopsy and ExifTool, therefore, the whole project will be related and developed with Python programming language.

3.1.2. Scientific Python Development Environment (SPYDER)

To work in a comfortable way and make great usage of Python, we have to choose an excellent Integrated Development Environment (IDE). In this case, we are going to use the Scientific Python Development Environment (SPYDER) integrated in the Anaconda environment. Spyder is a free and open source powerful scientific environment written in Python and designed by scientists, engineers and data analysts. It offers you advanced properties and characteristics such as editing, analysis, debugging, visualization of nice plots, IPython console and much more.

3.1.3. Example of usage

For an easy comprehension of the concepts already explained, a basic example of the PyExifTool library with the Spyder environment will be illustrated analyzing all the metadata found in the *Figure 22* with a simple Python script (*Figure 23*).

Figure 22. Image analyzed by the PyExifTool library testing script

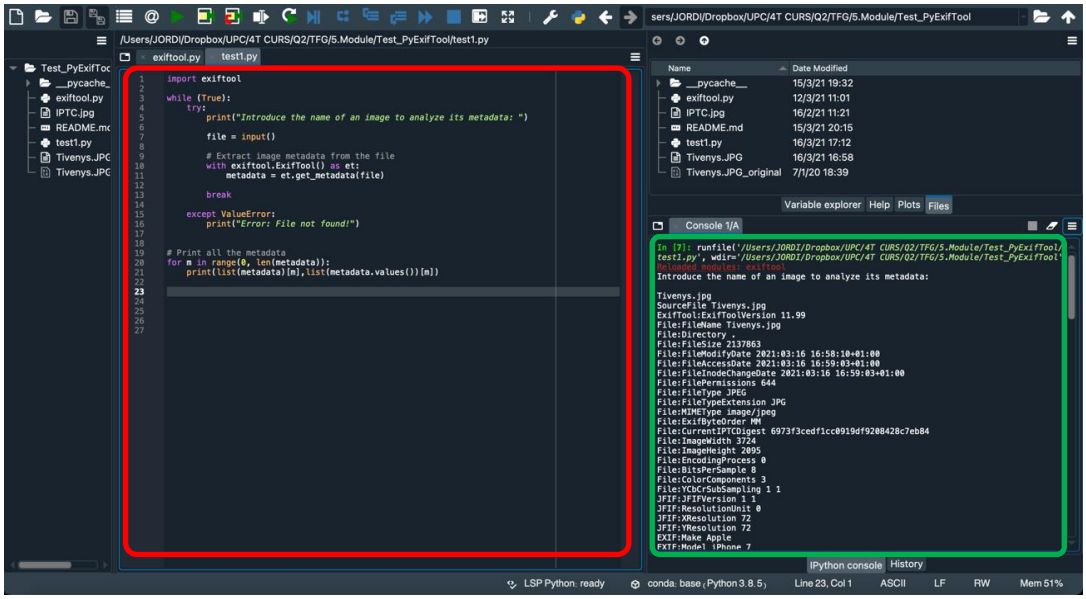


Figure 23. Spyder environment, script and output of the PyExifTool library testing script

In the figure above we can see the Spyder environment with the corresponding script and the output of the program, that is all the metadata found in the image (*see Appendix IV: Output of PyExifTool library testing script* and *Appendix V: PyExifTool testing script*).

The Python script is very basic. First of all, we import the library PyExifTool (its code is shown in *Appendix VI: PyExifTool library*). Secondly, we ask for an image to analyze, in that case the *Figure 22* (*Tivenys.jpg*). After that, and with the help of the class *ExifTool()* of the library, we extract all the image metadata found in the file with the method *get_matadata(file)* and print it in a pretty way in the Console.

An interesting thing to keep in mind is that the method *get_metadata(file)* prints the metadata and also shows the type of metadata that corresponds to the information found (EXIF, IPTC, JFIF, XMP,…).

3.2. Development of Autopsy File Ingest Modules

Before creating the module for automated analysis of image metadata, we are going to have a first contact with the creation of File Ingest Modules in Autopsy creating a basic module in Python (Jython). In that way, we will see how the creation of modules work.

Ingest Modules in Autopsy run on the data source added to a case. When you add a disk image to the case, you have a list of modules to run (see *Figure 8*). We are going to write one of those modules and include it in our case afterwards; this is basically the same we will do in the next chapter with the creation of the module to analyze image metadata.

The body of a File Ingest Module is based on two classes: the description of the module (see *Figure 24*) and the functionality of the module (see *Figure 25*).

In the *Figure 24* we can see the first class called *HelloWorldFileIngestModuleFactory* where the name and details of the module are described.

The next class is called *HelloWorldFileIngestModule*. In this class we will write the appropriate code that corresponds to the functionality we want the module do. This class is composed by the following methods:

- **startUp():** where the module is initialized. Here is where normally you write the code that can fail, in that way, throwing an exception in the startUp() method causes the entire module to stop.
- **process():** where we do our analysis. Each file is passed into this method so you can analyze it.
- **shutDown():** where sources are freed and summary messages are sent, if necessary.

We are going to write a module that analyzes every file title and sees if contains the string "HelloWorld". If this is accomplished, we will put the file in an artifact on the blackboard.

So, we create the method *process()* and do the analysis. First of all, we are going to skip non-files, we don't want to analyze them. Secondly, we start filtering the files by its name: if the file contains the string "HelloWorld" in his title, we create an artifact and an attribute and we add that attribute to the artifact. We have to take into account that when a module wants to store its results on the blackboard, it makes an artifact of the correct type (in that case our chosen artifact is TSK_INTERESTING_FILE_HIT) and then adds attributes to it (in our case we only add one attribute called TSK_SET_NAME).

Afterwards, we fire an event to notify that there is a new artifact. Accordingly, the UI is updated and refreshed with the new artifact when the module is executed.

Finally, in the *shutdown()* method, we send a message to the ingest inbox with the number of files that the module has found.

In the *Appendix VII: HelloWorld.py* you can see the full *HelloWorld.py* script.

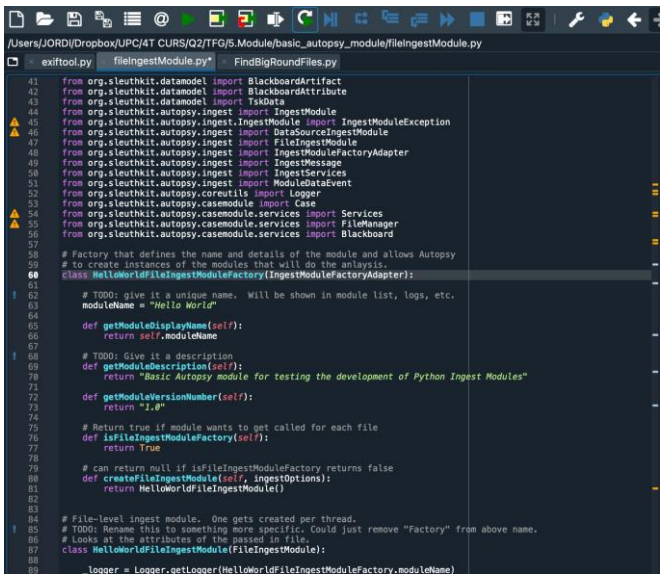


Figure 24. HelloWorld.py basic Autopsy module (I)

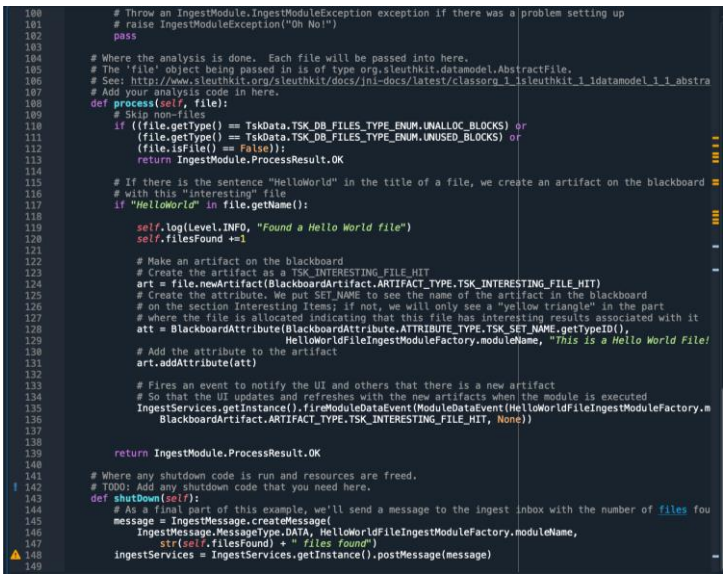


Figure 25. HelloWorld.py basic Autopsy module (II)

Once we have the code of the module, we are going to test it on Autopsy analyzing a folder (data source: Logical File) that contains different files, among them a text file called *HelloWorld.txt*. Therefore, we add the module as shown previously (see *Figure 11*) to the folder *python_modules* and we create another folder called *DemoScript*, in that folder we can add our *HelloWorld.py* module.

Once we have it, we are able to execute our script in Autopsy.

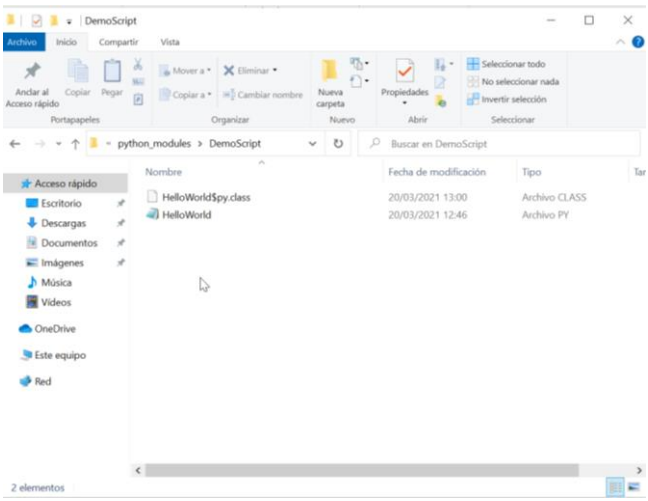


Figure 26. Addition of the HelloWorld.py module in the DemoScript folder

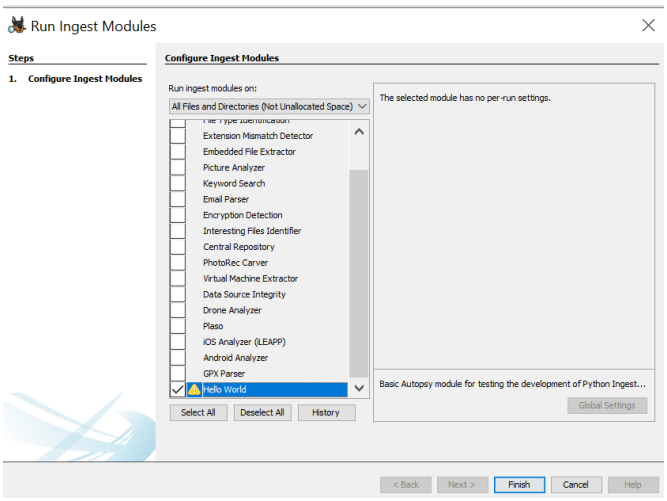


Figure 27. Selection of the Hello World module in Autopsy

The results are the following and we can see that the behavior is what we expected.

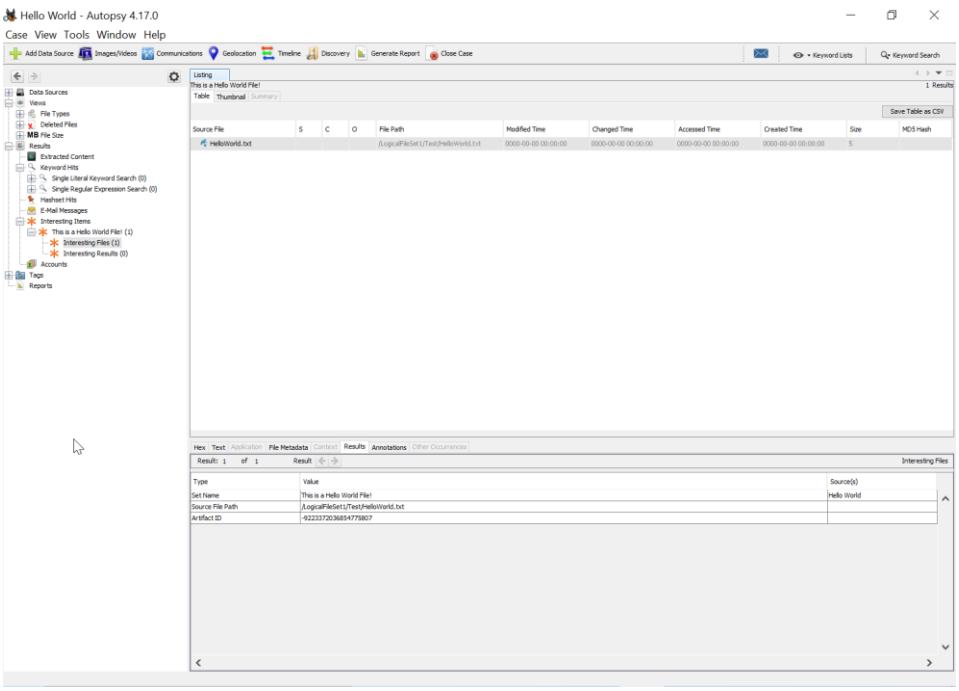


Figure 28. Hello World module results

4. Results

4.1. High Level module explanation

As we said during the thesis, our result will be a module, developed in Python language, able to analyze any type of metadata found in an image. Specially, the metadata that the *Picture Analyzer* Autopsy own module cannot analyze. In this way, Autopsy will show in the Tree Viewer a new artifact called *Image Metadata Analyzer* with all the analyzed images inside and the metadata found.

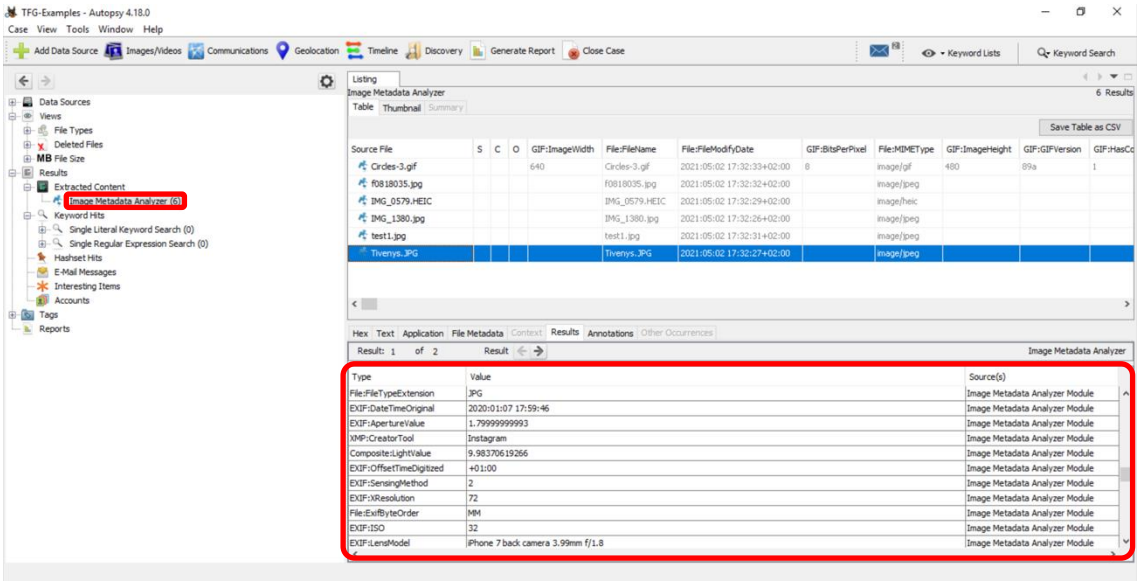


Figure 29. Image Metadata Analyzer preview

Apart from analyzing image metadata, that is a big advantage for digital forensics investigators, the module will also be able to filter the metadata in order to facilitate users to find specific images that contain specific metadata they want to search for their cases. The current available filters are the following:

Filter Type	***Text* Pattern**	***RegEx* Pattern**
Simple	*Name1*	*"[a-zA-Z0-9]"*
NOT	**NOT** *Name1*	*"(NOT)\s[a-zA-Z0-9]"*
AND	*Name1* **AND** *Name2* **AND** …	*"[a-zA-Z0-9]\s(AND)\s[a-zA-Z0-9]"*
OR	*Name1* **OR** *Name2* **OR** …	*"[a-zA-Z0-9]\s(OR)\s[a-zA-Z0-9]"*
==	*MetadataTag* **==** *MetadataValue*	*"[a-zA-Z0-9]\s(==)\s[a-zA-Z0-9]"*
!=	*MetadataTag* **!=** *MetadataValue*	*"[a-zA-Z0-9]\s(!=)\s[a-zA-Z0-9]"*
CONTAINS	*MetadataTag* **CONTAINS** *Name*	*"[a-zA-Z0-9]\s(CONTAINS)\s[a-zA-Z0-9]"*

DOES NOT CONTAIN	*MetadataTag* **DOES NOT CONTAIN** *Name*	*"[a-zA-Z0-9]\s(DOES\sNOT\sCONTAIN)\s[a-zA-Z0-9]"*

Table 6. Filters' description (I)

Filter Type	Description
Simple	Simple filter. The module searches if the word is in the metadata of any image analyzed in the case.
NOT	NOT filter. The module searches if the word is NOT in the metadata of any image analyzed in the case.
AND	AND filter. The module searches if all the words are inside the metadata of any image.
OR	OR filter. The module searches if at least one of the words is inside the metadata of any image.
==	Filter by specific metadata tag and value. The module searches if a specific metadata value is inside any image. Remember that the *MetadataTag* must match perfectly accordingly to the ExifTool tag names [16].
!=	Filter by specific metadata tag and value. The module searches if a specific metadata value is NOT inside any image. Remember that the *MetadataTag* must match perfectly accordingly to the ExifTool tag names [16].
CONTAINS	Filter by specific metadata tag. The module searches if any image contains the word in the metadata value of a specific *MetadataTag*. Remember that the *MetadataTag* must match perfectly accordingly to ExifTool tag names [16].
DOES NOT CONTAIN	Filter by specific metadata tag. The module searches if any image does NOT contain the word in the metadata value of a specific *MetadataTag*. Remember that the *MetadataTag* must match perfectly accordingly to the ExifTool tag names [16].

Table 7. Filters' description (II)

Once the filters are entered, the results will be shown as *Interesting Items*, in the Tree Viewer of Autopsy. Moreover, you will be able to see, in the comments of the Results, in which Metadata Tags the filters have been found.

For example, if the user want to know which are the images that were made with an *iPhone*, the module will search this specific word inside the metadata to filter and show only the images made with that specific device. After that, you will be able to see in which Metadata Tags the word has been found. This is also a big advantage that can facilitate and save a lot of time to investigators. Furthermore, you can insert more than one filter simultaneously to make your image search more precise (see *Figure 30).*

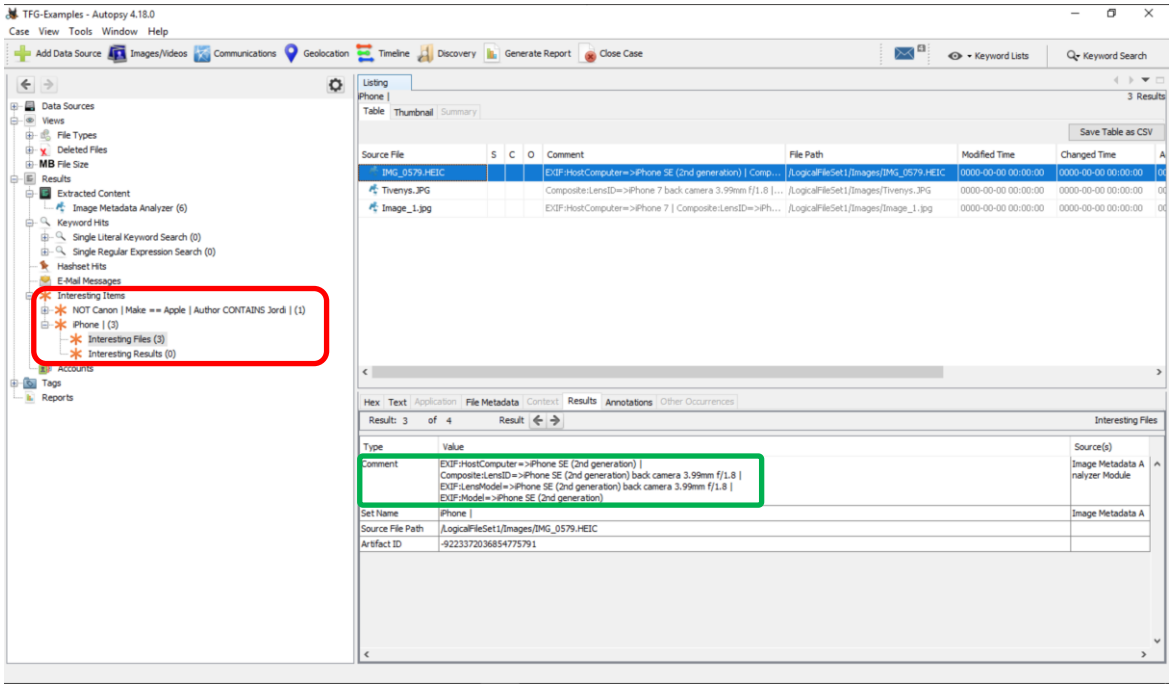


Figure 30. Image Metadata Filter preview

See *Appendix VIII: How to install the Image Metadata Analyzer Module* to learn how to install the module in your computer.

4.1.1. Flux Diagram

In the following figure you can see the flux diagram about the functioning of our module.

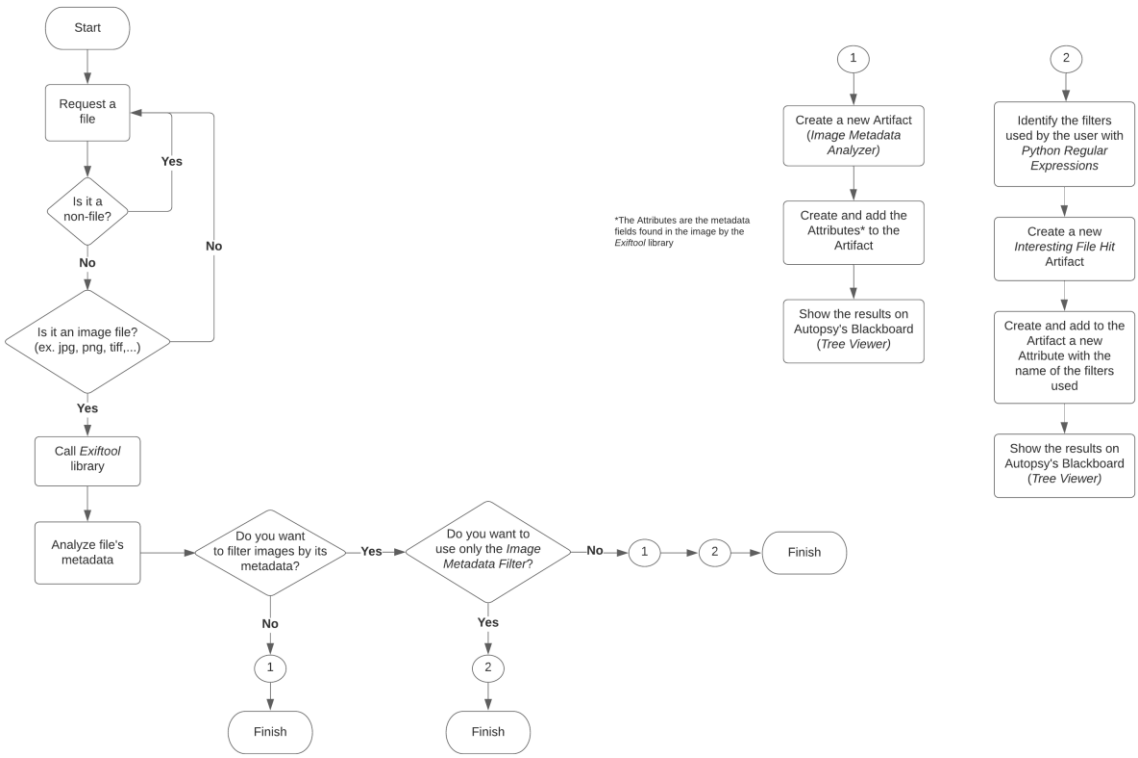


Figure 31. Module's flux diagram

Firstly, a file is requested and we check that is an image file such as JPG, PNG, TIFF, GIF or HEIC. If not, we discard it and request the following file.

Once we have filtered the file by its type, we call the specific method of the ExifTool library to analyze the metadata of the image.

Secondly, the metadata found is stored in a variable and since then, the program continue running following the specific path the user want to take:

- **Analyze only Image Metadata**. The user selects on the Module GUI the type of metadata and image files he/she wants to analyze and the module analyze them (see *Figure 32*).
- **Use only the Image Metadata Filter**. When we only want to filter images by its metadata, we have to select the type of image files we want to analyze, write the appropriate filters on the text field and select the "Use only the Image Metadata Filter" box. In this way, we will only make use of the filter mode. It is not necessary to select the type of metadata we want to analyze because in this mode, we analyze all the metadata available (see *Figure 33*).
- **Analyze Image Metadata and also make use of the Image Metadata Filter**. When we want to analyze the image metadata and also use the filter, we have to deselect the "Use only the Image Metadata Filter" box and fill the other options according to the user goals.

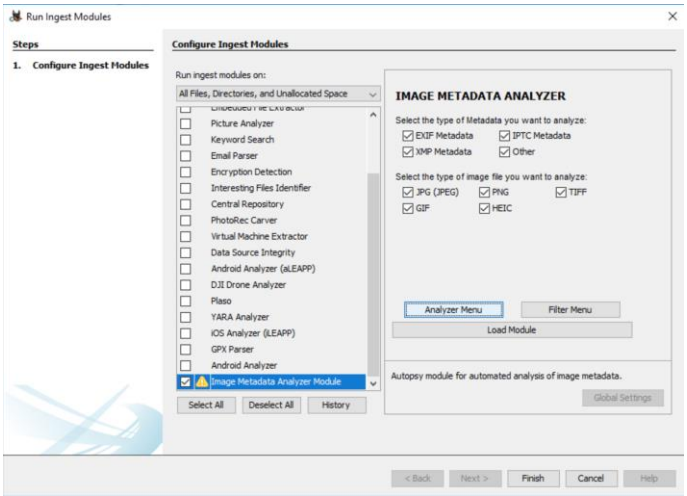


Figure 32. Image Metadata Analyzer Module GUI (I)

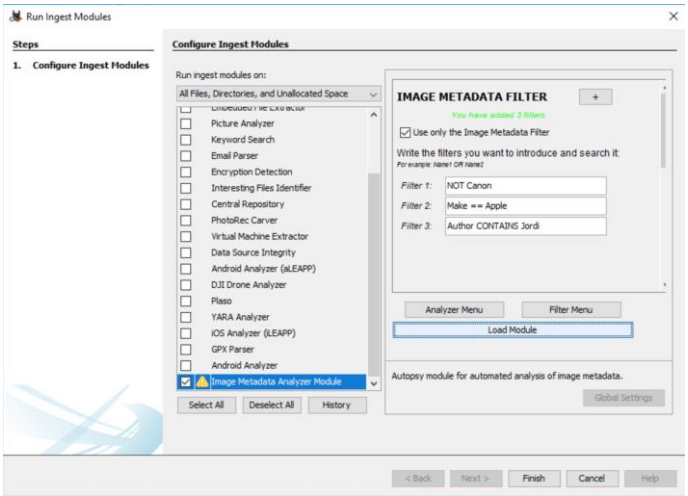


Figure 33. Image Metadata Analyzer Module GUI (II)

Finally, when the path is selected, the module creates the specific Artifact and Attributes accordingly to the path followed. The results will be shown in the *Tree Viewer* of the Autopsy User Interface as we have seen in *Figure 29* and *Figure 30*.

CAUTION: The "Use only Image Metadata Filter" box is created to avoid images to be analyzed every time the module is loaded. If the box is NOT selected, the images would be repeatedly added in the *Image Metadata Analyzer* section of the Directory Tree every time the module is executed and it would be an absolutely mess. When we select the box, we are saying that we do not want to analyze image metadata anymore (maybe, because we already did it before) and we only want to make use of the filter.

RECOMMENDATIONS: When we want to analyze and also filter images, the best option is to first analyze only one time the metadata of the images and once we have it, load

again the module but this time only using the filter mode ("Use only the Image Metadata Filter" box selected). See *Appendix IX: Example of Module usage*.

The code of the module is shown in *Appendix X: ImageAnalyzerGUI.py* and *Appendix XI: ImageAnalyzerLib.py*

4.2. Figure of merit

The creation of our module enables efficient and automated analysis of image metadata, including an optimized filter to find the exact images the user is searching for.

In this section, I will show you the efficiency that the module has analyzing and filtering image metadata. In that way, we will see the advantages that digital forensics investigators are going to have when using the module in their professional cases.

We have to take into account that the tests have been done with a *Macbook Pro 2015 15"*, accordingly, the results and speeds are directly related with the characteristics of the computer.

4.2.1. Image Metadata Analyzer

Firstly, we are going to test the basic functionality of the module, that is, only using the Image Metadata Analyzer Mode. For that reason, we do different analysis and test up to 500 images (approximately 900 MB), in that way we see how the module reacts to that amount of data.

In the next figures we can see the results. On the one hand, the *Figure 34* shows us the amount of time that the module takes to analyze a specific number of images. We notice that the behavior is linear: if the number of images grow, the time will proportionally grow.

On the other hand, in the *Figure 35*, there is the time that takes the module to analyze different amount of images' Megabytes. The behavior is also linear as in the first graph: when the number of MB analyzed grow, the time will also grow.

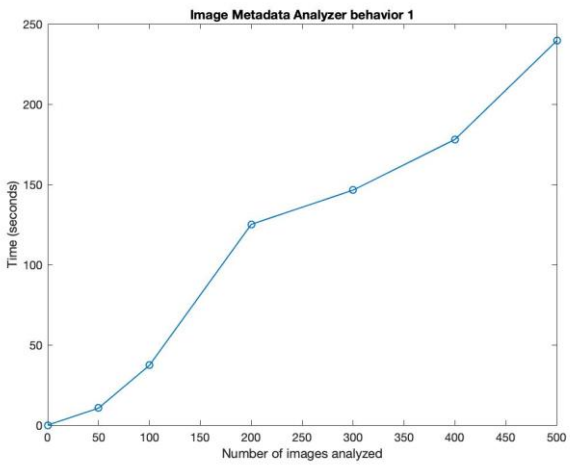


Figure 34. Image Metadata Analyzer behavior (I)

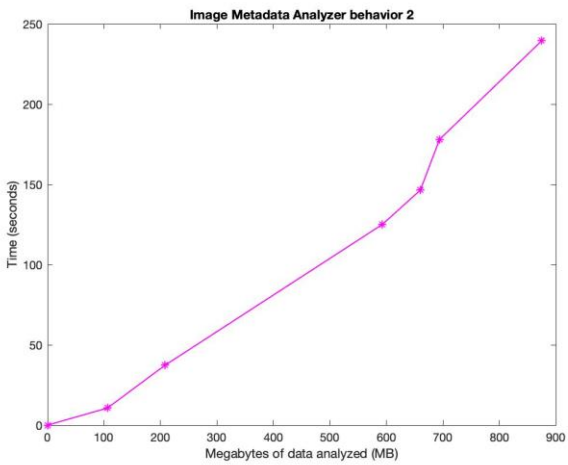


Figure 35. Image Metadata Filter behavior (II)

4.2.2. Image Metadata Filter

Secondly, we test the other functionality of our module: Image Filtering. In that case, we have the image metadata analysed and now we only want to make use of the filter. We are going to analyse the same amount of images analysed before and see if our module is efficient enough.

On the one hand, in the *Figure 36*, we can see how the time taken to analyse a specific number of images using filters do not vary so much, it is almost constant. That is a very good information because the time that takes to filter a specific number of images will not change too much when increasing the number of filters used.

On the other hand, watching the *Figure 37* we confirm the behavior explained before. The time taken to analyze a number of images is practically the same when we raise the number of filters used and logically, it is directly proportional with the number of images analyzed: if the number of images analyzed grow, the time will also increase.

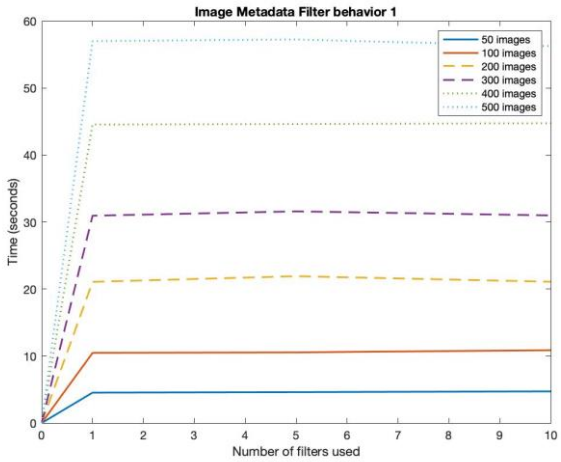


Figure 36. Image Metadata Filter behaviour (I)

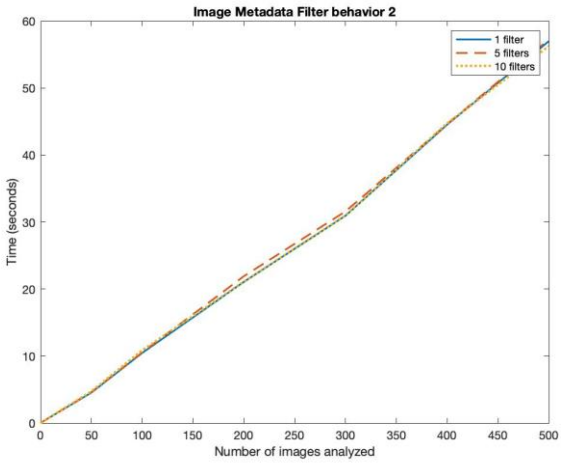


Figure 37. Image Metadata Filter behaviour (II)

All in all, with the figures explained above we conclude that the module is efficient enough. The module only takes near 4 minutes to analyze the image metadata of approximately 1 GB of data. But, the biggest advantage is when we have all the metadata analyzed and we only want to make use of the filter, the time taken to analyze and filter the images is incredibly fast and constant when adding new filters; in 1 minute we have filtered 500 images (1 GB of data approximately). Accordingly, when using the module you do not have to be scared of adding new filters, the module will respond to it perfectly and the difference of time lost using 1 filter or 10 filters will be insignificant.

5. Budget

We choose the Full Costing method to calculate the expenses of the project during a period of 4 months. For that reason, we are going to take into account the Direct Costs and Indirect Costs generated.

5.1. Direct Costs

During the project we have used several free open-source tools such as: Autopsy, SPYDER, ExifTool and VirtualBox.

For that reason, there will not be direct costs related to software licenses. The only direct costs of the project are going to be the computer used and the junior engineer salary:

- **Computer:** A Macbook Pro 2015 15'' will be used for the proper development of the project. It has a cost of 1.500 €.
- **Junior Engineer:** The developer of the project. We consider a gross salary of approximately 25.000€ per year.

	Cost (initial value)	Years of life	Final value	Depreciable Value
Computer	1.500 €	10	150 €	**1.485 €**

Table 8. Direct Costs (I)

	Gross Salary x hour	Hours working in a day	Hours working in a month (only week days)	Gross Salary in 4 months
Junior Engineer	11 €	4 hours	80 hours	**3.520 €**

Table 9. Direct Costs (II)

We take into account that the Depreciable Value (amortization) is calculated as follows:

$$Depreciable\ Value = \frac{Initial\ value - Final\ value}{Years\ of\ life} = \frac{1500€ - 150€}{10} = 1485\ €$$

5.2. Indirect Costs

The Indirect Costs of the project is only the Project Manager salary.

	Gross Salary x hour	Hours working in a month (only week days)	Gross Salary in 4 months
Project Manager	20 €	8 hours	**640 €**

Table 10. Indirect Costs

All in all, the **Total Budget** of our project during a period of 4 months and taking into account the Full Costing method (Direct Costs + Indirect Costs) is:

Direct Costs	5.005 €
Indirect Costs	640 €
TOTAL Budget	**5.645 €**

Table 11. Full Costing Total Budget

6. Conclusions and future development

6.1. Conclusions

Digital forensics is a very important field for society, any little contribution you can do is a major improvement and a big advantage that investigators can take in a professional case. This thesis has tried to help investigators to have an efficient and automated tool to analyse image metadata and reveal clear evidences in a court of law while using the same software they are used to employ in other investigations: Autopsy.

Based on the contributions done in this thesis, we have detected an important gap in the Autopsy framework and we have solved it creating a very demanded feature consisting in an Autopsy plug-in module capable of analyzing image metadata and enable the user to perform the filtering of image data bases based on a set of configurable rules. For that reason, investigators are not forced to run another external software to execute this functionality, this feature is integrated into the Autopsy framework through this new available open source plug-in.

According to the results, we have shown that the Image Metadata Analyzer Module works perfectly in real cases. Furthermore, the usability of the filters is very powerful and makes the software more user friendly and comfortable to work with. For that reason, the time you can spend using this module instead of other programs that are not specifically automated for the analysis of all the image metadata found in a picture is huge, and since now, we had not known about any other Autopsy plug-in that fits with these needs.

Taking into account that my knowledge about cybersecurity and digital forensics before starting this thesis was non-existent, I can say that after 4 months of individual hard work the learnings have been enormous and I am very happy with all the results achieved, not only the academic results, also the personal learnings. For that reason, this project have encouraged me to not stop and continue learning about the topic and the cybersecurity field.

Covid-19 pandemic has altered the way the final degree thesis was used to be done, for that reason, the project was done fully from home, with no face-to-face meetings and learnings. Despite these difficulties and keeping in mind that I had no prior knowledge about the topic, the project objectives have been perfectly and successfully achieved. Moreover, I want to make a special mention and thank my project supervisor Josep Pegueroles, who always has listened, advised and helped me.

It was not easy, as any big challenge you face in your life, there are always big steps, but also huge falls. Even so, the secret is always to fall forward and continue working hard, pursuing your goals and doing what you are passionate about.

6.2. Future Development

The metadata field in digital forensics is very big as well as important, in this thesis we only have focused on a little section, that is metadata in pictures. But, any digital file contains metadata inside that can help investigators to find clear evidences in their professional cases too. For that reason, the module can continue growing in that way, creating a general metadata analyser module able to examine any digital file (for example: audio files, text files, video files,...) and obtain all the metadata found.

Furthermore, as I am not a professional digital forensic investigator, the module could have weaknesses; accordingly, any feedback got of any user will be taken into account for further versions of the module. In that way, we will be able to make a module that fits in perfectly with any digital forensics investigation case, taking metadata analysis into the next level.

Bibliography

Reference List: Journal paper [1], Investigation Paper [2] [5], Master Degree Thesis [3], Online Reference [4] [6] [7] [8] [9] [12] [14] [15] [16] [17] [18], Article [10], OSDFCon Webinar [11] and Online Course [13].

[1] Cedeño Mata, Michelle and Miñano Belvis. “Cybersecurity & Digital Forensics Clinic”.

[2] Benallal Akhdar Mohamed-Reda and Pablo Sánchez Carmona. “Estudio del estado del arte en procedimientos forenses digitales”.

[3] Álvaro Borreguero Beltrán. “A Forensics Approach to Blockchain”.

[4] IPTC standard. https://iptc.org/standards/photo-metadata/.

[5] Metadata Working Group. “Guidelines for Handling Image Metadata”. Version 2.0, November 2010.

[6] Phil Harvey, Exiftool software. https://exiftool.org/.

[7] Get IPTC Photo Metadata. https://getpmd.iptc.org/getiptcpmd.html.

[8] Autopsy Documentation. http://sleuthkit.org/autopsy/docs/user-docs/4.18.0/.

[9] Sleuth Kit Library. http://sleuthkit.org/sleuthkit/docs/jni-docs/4.6.0/index.html.

[10] Christine Deschaseaux. “State of Image Metadata in 2018”. https://blog.imatag.com/state-of-image-metadata-in-2018.

[11] María Andrea Vignau. “Computer Autopsies: Use Free Forensic Software”.

[12] Sven Marnach, PyExiftool library. http://smarnach.github.io/pyexiftool/.

[13] DFIR.Science. “Introduction to Digital Forensics”. https://www.youtube.com/playlist?list=PLJu2iQtpGvv-2LtysuTTka7dHt9GKUbxD.

[14] Photo Metadata. https://www.photometadata.org/.

[15] EXIF Metadata. https://en.wikipedia.org/wiki/Exif.

[16] Exiftool Tag Names. https://exiftool.org/TagNames/

[17] Jython software. https://www.jython.org/download

[18] Image Metadata Analyzer Module github repository. https://github.com/jdom05/TFG.git

Appendix I: Autopsy Ingest Modules

These are the most important ingest modules available in Autopsy and a little description about its functionality. For further information visit [8].

- **Recent Activity**: Allows the examiner to see what activity has occurred in the last seven days of usage.
- **Hash Lookup**: Calculates MD5 hash values for files and looks up hash values in a database to determine if the file is 'known bad', 'known' or 'unknown'.
- **File Type Identification**: Identifies files based on their internal signatures.
- **Extension Mismatch Detector**: Detects files that have an extension not traditionally associated with the file's detected type and someone may be trying to hide.
- **Embedded File Extraction**: Expands archive files to enable Autopsy to analyze all files on the system. It enables keyword search and hash lookup to analyze files inside of archives.
- **Picture Analyzer**: Extracts EXIF (Exchangeable Image File Format) information from ingested pictures.
- **Keyword Search**: Supports manual text searching.
- **Email Parser**: Allows the examiner to identify email based communications.
- **Encryption Detection**: Searches for files that could be encrypted using both a general entropy calculation and more specialized tests for certain file types.
- **Interesting Files Identifier**: Allows the examiner to search for files or directories in a data source and flag them when are found.
- **Central Repository**: Allows a user to find matching artifacts both across cases and across data sources in the same case.
- **PhotoRec Carver**: Carves files from unallocated space in the data source.
- **Virtual Machine Extractor**: Adds a virtual machine found in the data source to the case as a new data source so the user is able to analyze it.
- **Data Source Integrity**: Verifies the hashes associated with the data source (if any) or if there are no hashes associated the module calculates them.
- **Drone Analyzer**: Allows you to analyze files from the internal SD card of a drone.
- **iOS Analyzer**: Runs *iLEAPP*, that is an external module able to analyze iOS logs, events and Plist files (Properties file used by macOS applications).
- **Android Analyzer**: Allows the examiner to analyze SQLite and other files from an Android device.
- **GPX Parser**: Allows you to import GPS data from a GPX file.

Appendix II: ExifTool analysis of f0674291.jpg

```
MacBook-Pro-de-Jordi:3.Metadata JORDI$ exiftool f0674291.jpg
ExifTool Version Number         : 12.21
File Name                       : f0674291.jpg
Directory                       : .
File Size                       : 2.5 MiB
File Modification Date/Time     : 2021:03:02 09:51:08+01:00
File Access Date/Time           : 2021:03:02 10:11:25+01:00
File Inode Change Date/Time     : 2021:03:02 10:11:24+01:00
File Permissions                : -rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
Exif Byte Order                 : Little-endian (Intel, II)
Make                            : Canon
Camera Model Name               : Canon EOS 1100D
Orientation                     : Horizontal (normal)
X Resolution                    : 72
Y Resolution                    : 72
Resolution Unit                 : inches
Modify Date                     : 2016:12:30 22:20:21
Artist                          :
Y Cb Cr Positioning             : Co-sited
Copyright                       :
Exposure Time                   : 1/50
F Number                        : 3.5
Exposure Program                : Not Defined
ISO                             : 400
Sensitivity Type                : Recommended Exposure Index
Recommended Exposure Index      : 400
Exif Version                    : 0230
Date/Time Original              : 2016:12:30 22:20:21
Create Date                     : 2016:12:30 22:20:21
Components Configuration        : Y, Cb, Cr, -
Shutter Speed Value             : 1/49
Aperture Value                  : 3.5
Flash                           : Off, Did not fire
Focal Length                    : 28.0 mm
Macro Mode                      : Normal
Self Timer                      : Off
Quality                         : Fine
Canon Flash Mode                : Off
Continuous Drive                : Single
Focus Mode                      : AI Focus AF
Record Mode                     : JPEG
Canon Image Size                : Large
Easy Mode                       : Flash Off
Digital Zoom                    : None
Contrast                        : Normal
Saturation                      : Normal
Camera ISO                      : Auto
Metering Mode                   : Evaluative
Focus Range                     : Not Known
Canon Exposure Mode             : Easy
Lens Type                       : Canon EF 28-80mm f/3.5-5.6
Max Focal Length                : 80 mm
Min Focal Length                : 28 mm
Focal Units                     : 1/mm
Max Aperture                    : 3.6
Min Aperture                    : 23
Flash Activity                  : 0
Flash Bits                      : (none)
Zoom Source Width               : 0
Zoom Target Width               : 0
Manual Flash Output             : n/a
Color Tone                      : Normal
Auto ISO                        : 100
Base ISO                        : 400
Measured EV                     : 7.25
Target Aperture                 : 3.6
```

```
Target Exposure Time            : 1/51
Exposure Compensation           : 0
White Balance                   : Auto
Slow Shutter                    : None
Shot Number In Continuous Burst : 0
Optical Zoom Code               : n/a
Camera Temperature              : 18 C
Flash Guide Number              : 0
Flash Exposure Compensation     : 0
Auto Exposure Bracketing        : Off
AEB Bracket Value               : 0
Control Mode                    : Camera Local Control
Measured EV 2                   : 7.625
Bulb Duration                   : 0
Camera Type                     : EOS High-end
ND Filter                       : n/a
Canon Image Type                : Canon EOS 1100D
Canon Firmware Version          : Firmware Version 1.0.4
Flash Metering Mode             : Off
Camera Orientation              : Horizontal (normal)
Focus Distance Upper            : 0 m
Focus Distance Lower            : 0 m
Firmware Version                : 1.0.4
File Index                      : 8627
Directory Index                 : 100
Contrast Standard               : 0
Sharpness Standard              : 3
Saturation Standard             : 0
Color Tone Standard             : 0
Contrast Portrait               : 0
Sharpness Portrait              : 2
Saturation Portrait             : 0
Color Tone Portrait             : 0
Contrast Landscape              : 0
Sharpness Landscape             : 4
Saturation Landscape            : 0
Color Tone Landscape            : 0
Contrast Neutral                : 0
Sharpness Neutral               : 0
Saturation Neutral              : 0
Color Tone Neutral              : 0
Contrast Faithful               : 0
Sharpness Faithful              : 0
Saturation Faithful             : 0
Color Tone Faithful             : 0
Contrast Monochrome             : 0
Sharpness Monochrome            : 3
Filter Effect Monochrome        : None
Toning Effect Monochrome        : None
Contrast Auto                   : 0
Sharpness Auto                  : 3
Saturation Auto                 : 0
Color Tone Auto                 : 0
Filter Effect Auto              : n/a
Toning Effect Auto              : n/a
Contrast User Def 1             : 0
Sharpness User Def 1            : 3
Saturation User Def 1           : 0
Color Tone User Def 1           : 0
Filter Effect User Def 1        : None
Toning Effect User Def 1        : None
Contrast User Def 2             : 0
Sharpness User Def 2            : 3
Saturation User Def 2           : 0
Color Tone User Def 2           : 0
Filter Effect User Def 2        : None
Toning Effect User Def 2        : None
Contrast User Def 3             : 0
Sharpness User Def 3            : 3
Saturation User Def 3           : 0
Color Tone User Def 3           : 0
Filter Effect User Def 3        : None
Toning Effect User Def 3        : None
```

```
User Def 1 Picture Style        : Standard
User Def 2 Picture Style        : Standard
User Def 3 Picture Style        : Standard
Canon Model ID                  : EOS Rebel T3 / 1100D / Kiss X50
Thumbnail Image Valid Area      : 0 159 7 112
AF Area Mode                    : Auto
Num AF Points                   : 9
Valid AF Points                 : 9
Canon Image Width               : 4272
Canon Image Height              : 2848
AF Image Width                  : 4272
AF Image Height                 : 2848
AF Area Widths                  : 106 106 106 149 183 149 106 106 106
AF Area Heights                 : 148 148 148 102 190 102 148 148 148
AF Area X Positions             : -1123 -674 -674 0 0 0 674 674 1123
AF Area Y Positions             : 0 333 -333 626 0 -626 333 -333 0
AF Points In Focus              : 4
AF Points Selected              : 0,1,2,3,4,5,6,7,8
Original Decision Data Offset   : 0
Bracket Mode                    : Off
Bracket Value                   : 0
Bracket Shot Number             : 0
Raw Jpg Size                    : Large
Long Exposure Noise Reduction 2 : Off
WB Bracket Mode                 : Off
WB Bracket Value AB             : 0
WB Bracket Value GM             : 0
Live View Shooting              : Off
Flash Exposure Lock             : Off
Internal Serial Number          : YA0241473
Dust Removal Data               : (Binary data 1024 bytes, use -b option to extract)
Crop Left Margin                : 0
Crop Right Margin               : 0
Crop Top Margin                 : 0
Crop Bottom Margin              : 0
Exposure Level Increments       : 1/3 Stop
Flash Sync Speed Av             : Auto
Long Exposure Noise Reduction   : Off
High ISO Noise Reduction        : Standard
Highlight Tone Priority         : Disable
AF Assist Beam                  : Emits
Shutter-AE Lock                 : AF/AE lock
Set Button When Shooting        : Normal (disabled)
Flash Button Function           : Raise built-in flash
LCD Display At Power On         : Display
Aspect Ratio                    : 3:2
Cropped Image Width             : 4272
Cropped Image Height            : 2848
Cropped Image Left              : 0
Cropped Image Top               : 0
Tone Curve                      : Standard
Sharpness                       : 3
Sharpness Frequency             : n/a
Sensor Red Level                : 0
Sensor Blue Level               : 0
White Balance Red               : 0
White Balance Blue              : 0
Color Temperature               : 5200
Picture Style                   : Standard
Digital Gain                    : 0
WB Shift AB                     : 0
WB Shift GM                     : 0
Measured RGGB                   : 398 1024 1024 597
VRD Offset                      : 0
Sensor Width                    : 4352
Sensor Height                   : 2874
Sensor Left Border              : 72
Sensor Top Border               : 23
Sensor Right Border             : 4343
Sensor Bottom Border            : 2870
Black Mask Left Border          : 0
Black Mask Top Border           : 0
Black Mask Right Border         : 0
```

```
Black Mask Bottom Border        : 0
Color Data Version              : 9 (60D/1100D)
WB RGGB Levels As Shot          : 2709 1024 1024 1618
Color Temp As Shot              : 5450
WB RGGB Levels Auto             : 2709 1024 1024 1618
Color Temp Auto                 : 5450
WB RGGB Levels Measured         : 2707 1024 1023 1617
Color Temp Measured             : 5450
WB RGGB Levels Daylight         : 2341 1024 1024 1533
Color Temp Daylight             : 5200
WB RGGB Levels Shade            : 2745 1024 1024 1314
Color Temp Shade                : 7000
WB RGGB Levels Cloudy           : 2539 1024 1024 1411
Color Temp Cloudy               : 6000
WB RGGB Levels Tungsten         : 1631 1024 1024 2320
Color Temp Tungsten             : 3200
WB RGGB Levels Fluorescent      : 2020 1024 1024 2131
Color Temp Fluorescent          : 3732
WB RGGB Levels Kelvin           : 2341 1024 1024 1533
Color Temp Kelvin               : 5189
WB RGGB Levels Flash            : 2621 1024 1024 1402
Color Temp Flash                : 6211
Average Black Level             : 2047 2047 2047 2047
Raw Measured RGGB               : 114490 292254 295581 169095
Per Channel Black Level         : 2047 2047 2047 2047
Specular White Level            : 15094
Linearity Upper Margin          : 10000
Picture Style User Def          : Standard; Standard; Standard
Picture Style PC                : None; None; None
Custom Picture Style File Name  :
Vignetting Corr Version         : 0
Peripheral Lighting             : Off
Distortion Correction           : Off
Chromatic Aberration Corr       : Off
Peripheral Lighting Value       : 0
Distortion Correction Value     : 0
Original Image Width            : 4272
Original Image Height           : 2848
Peripheral Lighting Setting     : Off
Peripheral Illumination Corr    : Off
Auto Lighting Optimizer         : Standard
Ambience Selection              : Standard
User Comment                    :
Sub Sec Time                    : 00
Sub Sec Time Original           : 00
Sub Sec Time Digitized          : 00
Flashpix Version                : 0100
Color Space                     : sRGB
Exif Image Width                : 4272
Exif Image Height               : 2848
Interoperability Index          : R98 - DCF basic file (sRGB)
Interoperability Version        : 0100
Focal Plane X Resolution        : 4720.441989
Focal Plane Y Resolution        : 4786.554622
Focal Plane Resolution Unit     : inches
Custom Rendered                 : Normal
Exposure Mode                   : Auto
Scene Capture Type              : Standard
Owner Name                      :
Serial Number                   : 063061003567
Lens Info                       : 28-80mm f/?
Lens Model                      : EF28-80mm f/3.5-5.6
Lens Serial Number              : 0000000000
Compression                     : JPEG (old-style)
Thumbnail Offset                : 9052
Thumbnail Length                : 8720
Rating                          : 0
Image Width                     : 4272
Image Height                    : 2848
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:2:2 (2 1)
```

```
Drive Mode                      : Single-frame Shooting
File Number                     : 100-8627
Lens                            : 28.0 - 80.0 mm
Shooting Mode                   : Flash Off
WB RGGB Levels                  : 2709 1024 1024 1618
Aperture                        : 3.5
Blue Balance                    : 1.580078
Image Size                      : 4272x2848
Lens ID                         : Canon EF 28-80mm f/3.5-5.6
Megapixels                      : 12.2
Red Balance                     : 2.645508
Scale Factor To 35 mm Equivalent: 1.6
Shutter Speed                   : 1/50
Create Date                     : 2016:12:30 22:20:21.00
Date/Time Original              : 2016:12:30 22:20:21.00
Modify Date                     : 2016:12:30 22:20:21.00
Thumbnail Image                 : (Binary data 8720 bytes, use -b option to extract)
Lens                            : 28.0 - 80.0 mm (35 mm equivalent: 44.0 - 125.8 mm)
Circle Of Confusion             : 0.019 mm
Depth Of Field                  : inf (0.00 m - inf)
Field Of View                   : 44.5 deg
Focal Length                    : 28.0 mm (35 mm equivalent: 44.0 mm)
Hyperfocal Distance             : 11.73 m
Light Value                     : 7.3
```

Appendix III: ExifTool analysis of iptc-image.jpg

```
MacBook-Pro-de-Jordi:3.Metadata JORDI$ exiftool iptc-image.jpg
ExifTool Version Number         : 12.21
File Name                       : iptc-image.jpg
Directory                       : .
File Size                       : 126 KiB
File Modification Date/Time     : 2021:03:02 10:01:14+01:00
File Access Date/Time           : 2021:03:02 11:06:46+01:00
File Inode Change Date/Time     : 2021:03:02 11:06:45+01:00
File Permissions                : -rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
Exif Byte Order                 : Big-endian (Motorola, MM)
Image Description               : The description aka caption (ref2019.1)
X Resolution                    : 72
Y Resolution                    : 72
Resolution Unit                 : inches
Artist                          : Creator1 (ref2019.1)
Y Cb Cr Positioning             : Centered
Copyright                       : Copyright (Notice) 2019.1 IPTC - www.iptc.org  (ref2019.1)
Current IPTC Digest             : 1a752785c1ca57e321cf228e3c3e8498
Object Attribute Reference      : A Genre (ref2019.1)
Object Name                     : The Title (ref2019.1)
Subject Reference               : IPTC:1ref2019.1, IPTC:2ref2019.1, IPTC:3ref2019.1
Keywords                        : Keyword1ref2019.1, Keyword2ref2019.1, Keyword3ref2019.1
Special Instructions            : An Instruction (ref2019.1)
Time Created                    : 19:01:00+00:00
By-line                         : Creator1 (ref2019.1)
By-line Title                   : Creator's Job Title  (ref2019.1)
Sub-location                    : Sublocation (Core) (ref2019.1)
Province-State                  : Province/State (Core) (ref2019.1
Country-Primary Location Code   : R19
Country-Primary Location Name   : Country (Core) (ref2019.1)
Original Transmission Reference : Job Id (ref2019.1)
Copyright Notice                : Copyright (Notice) 2019.1 IPTC - www.iptc.org  (ref2019.1)
Caption-Abstract                : The description aka caption (ref2019.1)
Writer-Editor                   : Description Writer (ref2019.1)
Application Record Version      : 4
XMP Toolkit                     : Image::ExifTool 11.74
Country Code                    : R19
Creator City                    : Creator's CI: City (ref2019.1)
Creator Country                 : Creator's CI: Country (ref2019.1)
Creator Address                 : Creator's CI: Address, line 1 (ref2019.1)
Creator Postal Code             : Creator's CI: Postcode (ref2019.1)
Creator Region                  : Creator's CI: State/Province (ref2019.1)
Creator Work Email              : Creator's CI: Email@1, Email@2 (ref2019.1)
Creator Work Telephone          : Creator's CI: Phone # 1, Phone # 2 (ref2019.1)
Creator Work URL                : http://www.Creators.CI/WebAddress/ref2019.1
Intellectual Genre              : A Genre (ref2019.1)
Location                        : Sublocation (Core) (ref2019.1)
Scene                           : IPTC-Scene-Code1 (ref2019.1), IPTC-Scene-Code2 (ref2019.1)
Subject Code                    : 1ref2019.1, 2ref2019.1, 3ref2019.1
About Cv Term Cv Id             : http://example.com/cv/about/ref2019.1
About Cv Term Id                : http://example.com/cv/about/ref2019.1/code987
About Cv Term Name              : CV-Term Name 1 (ref2019.1)
About Cv Term Refined About     : http://example.com/cv/refinements2/ref2019.1/codeX145
Additional Model Information    : Additional Model Info (ref2019.1)
Artwork Circa Date Created      : AO Circa Date: between 1550 and 1600 (ref2019.1)
Artwork Content Description     : AO Content Description 1 (ref2019.1)
Artwork Contribution Description: AO Contribution Description 1 (ref2019.1)
Artwork Copyright Notice        : AO Copyright Notice 1 (ref2019.1)
Artwork Creator                 : AO Creator Name 1a (ref2019.1), AO Creator Name 1b (ref2019.1)
Artwork Creator ID              : AO Creator Id 1a (ref2019.1), AO Creator Id 1b (ref2019.1)
Artwork Copyright Owner ID      : AO Current Copyright Owner ID 1 (ref2019.1)
Artwork Copyright Owner Name    : AO Current Copyright Owner Name 1 (ref2019.1)
Artwork Licensor ID             : AO Current Licensor ID 1 (ref2019.1)
Artwork Licensor Name           : AO Current Licensor Name 1 (ref2019.1)
Artwork Date Created            : 1916:10:26 12:13:14+00:00
Artwork Physical Description    : AO Physical Description 1 (ref2019.1)
```

```
Artwork Source                  : AO Source 1 (ref2019.1)
Artwork Source Inventory No     : AO Source Inventory No 1 (ref2019.1)
Artwork Source Inv URL          : AO Source Inventory URL (ref2019.1)
Artwork Style Period            : AO Style Baroque (ref2019.1), AO Style Italian Baroque
(ref2019.1)
Artwork Title                   : AO Title 1 (ref2019.1)
Digital Image GUID              : http://example.com/imageGUIDs/TestGUID12345/ref2019.1
Digital Source Type             : http://cv.iptc.org/newscodes/digitalsourcetype/softwareImage
Embedded Encoded Rights Expr    : The Encoded Rights Expression (ref2019.1)
Embedded Encoded Rights Expr Type: IANA Media Type of ERE (ref2019.1)
Embedded Encoded Rights Expr Lang ID: http://example.org/RELids/id4711/ref2019.1
Event                           : An Event (ref2019.1)
Genre Cv Id                     : http://example.com/cv/genre/ref2019.1
Genre Cv Term Id                : http://example.com/cv/genre/ref2019.1/code1369
Genre Cv Term Name              : Genre CV-Term Name 1 (ref2019.1)
Genre Cv Term Refined About     : http://example.com/cv/genrerefinements2/ref2019.1/codeY864
Image Region Name               : Listener 1, Listener 2, Speaker 1
Image Region Organisation In Image Name: Organisation name no 1 in region persltr2 (ref2019.1),
Organisation name no 1 in region persltr2 (ref2019.1), Organisation name no 1 in region persltr3
(ref2019.1)
Image Region Person In Image    : Person name no 1 in region persltr2 (ref2019.1), Person name no 1
in region persltr3 (ref2019.1), Person name no 1 in region persltr1 (ref2019.1)
Image Region Boundary H         : 0.385
Image Region Boundary Shape     : Rectangle, Circle, Polygon
Image Region Boundary Unit      : Relative, Relative, Relative
Image Region Boundary W         : 0.127
Image Region Boundary X         : 0.31, 0.59
Image Region Boundary Y         : 0.18, 0.426
Image Region Ctype Name         : Region Boundary Content Type Name (ref2019.1), Region Boundary
Content Type Name (ref2019.1), Region Boundary Content Type Name (ref2019.1)
Image Region Ctype Identifier   : https://example.org/rctype/type2019.1a,
https://example.org/rctype/type2019.1b, https://example.org/rctype/type2019.1a,
https://example.org/rctype/type2019.1b, https://example.org/rctype/type2019.1a,
https://example.org/rctype/type2019.1b
Image Region ID                 : persltr2, persltr3, persltr1
Image Region Role Name          : Region Boundary Content Role Name (ref2019.1), Region Boundary
Content Role Name (ref2019.1), Region Boundary Content Role Name (ref2019.1)
Image Region Role Identifier    : https://example.org/rrole/role2019.1a,
https://example.org/rrole/role2019.1b, https://example.org/rrole/role2019.1a,
https://example.org/rrole/role2019.1b, https://example.org/rrole/role2019.1a,
https://example.org/rrole/role2019.1b
Image Region Boundary Rx        : 0.068
Image Region Boundary Vertices X: 0.05, 0.148, 0.375
Image Region Boundary Vertices Y: 0.713, 0.041, 0.863
Linked Encoded Rights Expr      : http://example.org/linkedrightsexpression/id986/ref2019.1
Linked Encoded Rights Expr Type : IANA Media Type of ERE (ref2019.1)
Linked Encoded Rights Expr Lang ID: http://example.org/RELids/id4712/ref2019.1
Location Created City           : City (Location created1) (ref2019.1)
Location Created Country Code   : R17
Location Created Country Name   : CountryName (Location created1) (ref2019.1)
Location Created Location Id    : Location Id (Location created1) (ref2019.1)
Location Created Province State : Province/State (Location created1) (ref2019.1)
Location Created Sublocation    : Sublocation (Location created1) (ref2019.1)
Location Created World Region   : Worldregion (Location created1) (ref2019.1)
Location Shown City             : City (Location shown1) (ref2019.1), City (Location shown2)
(ref2019.1)
Location Shown Country Code     : R17, R17
Location Shown Country Name     : CountryName (Location shown1) (ref2019.1), CountryName (Location
shown2) (ref2019.1)
Location Shown Location Id      : Location Id 1a(Location shown1) (ref2019.1), Location Id
1b(Location shown1) (ref2019.1), Location Id 2a(Location shown2) (ref2019.1), Location Id
2b(Location shown2) (ref2019.1)
Location Shown Province State   : Province/State (Location shown1) (ref2019.1), Province/State
(Location shown2) (ref2019.1)
Location Shown Sublocation      : Sublocation (Location shown1) (ref2019.1), Sublocation (Location
shown2) (ref2019.1)
Location Shown World Region     : Worldregion (Location shown1) (ref2019.1), Worldregion (Location
shown2) (ref2019.1)
Max Avail Height                : 14
Max Avail Width                 : 20
Model Age                       : 25, 27, 30
Organisation In Image Code      : Organisation Code 1 (ref2019.1), Organisation Code 2 (ref2019.1),
Organisation Code 3 (ref2019.1)
```

```
Organisation In Image Name      : Organisation Name 1 (ref2019.1), Organisation Name 2 (ref2019.1),
Organisation Name 3 (ref2019.1)
Person In Image                 : Person Shown 1 (ref2019.1), Person Shown 2 (ref2019.1)
Person In Image Cv Term Cv Id   : http://example.com/cv/test99/ref2019.1
Person In Image Cv Term Id      : http://example.com/cv/test99/code987/ref2019.1
Person In Image Cv Term Name    : Person Characteristic Name 1 (ref2019.1)
Person In Image Cv Term Refined About: http://example.com/cv/refinements987/codeY765/ref2019.1
Person In Image Description     : Person Description 1 (ref2019.1)
Person In Image Id              : http://wikidata.org/item/Q123456789/ref2019.1,
http://freebase.com/m/987654321/ref2019.1
Person In Image Name            : Person Name 1 (ref2019.1)
Product In Image Description    : Product Description 1 (ref2019.1)
Product In Image GTIN           : 123456782019.1
Product In Image Name           : Product Name 1 (ref2019.1)
Registry Entry Role             : Registry Entry Role ID 1 (ref2019.1), Registry Entry Role ID 2
(ref2019.1)
Registry Item ID                : Registry Image ID 1 (ref2019.1), Registry Image ID 2 (ref2019.1)
Registry Organisation ID        : Registry Organisation ID 1 (ref2019.1), Registry Organisation ID
2 (ref2019.1)
Creator                         : Creator1 (ref2019.1), Creator2 (ref2019.1)
Description                     : The description aka caption (ref2019.1)
Rights                          : Copyright (Notice) 2019.1 IPTC - www.iptc.org  (ref2019.1)
Subject                         : Keyword1ref2019.1, Keyword2ref2019.1, Keyword3ref2019.1
Title                           : The Title (ref2019.1)
Authors Position                : Creator's Job Title  (ref2019.1)
Caption Writer                  : Description Writer (ref2019.1)
City                            : City (Core) (ref2019.1)
Country                         : Country (Core) (ref2019.1)
Credit                          : Credit Line (ref2019.1)
Date Created                    : 2019:10:16 19:01:00+00:00
Headline                        : The Headline (ref2019.1)
Instructions                    : An Instruction (ref2019.1)
Source                          : Source (ref2019.1)
State                           : Province/State (Core) (ref2019.1)
Transmission Reference          : Job Id (ref2019.1)
Copyright Owner ID              : Copyright Owner Id 1 (ref2019.1), Copyright Owner Id 2
(ref2019.1)
Copyright Owner Name            : Copyright Owner Name 1 (ref2019.1), Copyright Owner Name 2
(ref2019.1)
Image Creator ID                : Image Creator Id 1 (ref2019.1), Image Creator Id 2 (ref2019.1)
Image Creator Name              : Image Creator Name 1 (ref2019.1), Image Creator Name 2
(ref2019.1)
Image Creator Image ID          : Image Creator Image ID (ref2019.1)
Image Supplier ID               : Image Supplier Id (ref2019.1)
Image Supplier Name             : Image Supplier Name (ref2019.1)
Image Supplier Image ID         : Image Supplier Image ID (ref2019.1)
Licensor City                   : Licensor City 1 (ref2019.1), Licensor City 2 (ref2019.1)
Licensor Country                : Licensor Country 1 (ref2019.1), Licensor Country 2 (ref2019.1)
Licensor Email                  : Licensor Email 1 (ref2019.1), Licensor Email 2 (ref2019.1)
Licensor Extended Address       : Licensor Ext Addr 1 (ref2019.1), Licensor Ext Addr 2 (ref2019.1)
Licensor ID                     : Licensor ID 1 (ref2019.1), Licensor ID 2 (ref2019.1)
Licensor Name                   : Licensor Name 1 (ref2019.1), Licensor Name 2 (ref2019.1)
Licensor Postal Code            : Licensor Postcode 1 (ref2019.1), Licensor Postcode 2 (ref2019.1)
Licensor Region                 : Licensor Region 1 (ref2019.1), Licensor Region 2 (ref2019.1)
Licensor Street Address         : Licensor Street Addr 1 (ref2019.1), Licensor Street Addr 2
(ref2019.1)
Licensor Telephone 1            : Licensor Phone1 1 (ref2019.1), Licensor Phone1 2 (ref2019.1)
Licensor Telephone 2            : Licensor Phone2 1 (ref2019.1), Licensor Phone2 2 (ref2019.1)
Licensor URL                    : Licensor URL 1 (ref2019.1), Licensor URL 2 (ref2019.1)
Minor Model Age Disclosure      : Age 25 or Over
Model Release ID                : Model Release ID 1 (ref2019.1), Model Release ID 2 (ref2019.1)
Model Release Status            : Not Applicable
Property Release ID             : Property Release ID 1 (ref2019.1), Property Release ID 2
(ref2019.1)
Property Release Status         : Not Applicable
Rating                          : 1.0
Usage Terms                     : Rights Usage Termns (ref2019.1)
Web Statement                   : http://www.WebStatementOfRights.org/2019.1
DCT Encode Version              : 100
APP14 Flags 0                   : (none)
APP14 Flags 1                   : (none)
Color Transform                 : YCbCr
Image Width                     : 1000
```

```
Image Height                    : 500
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:4:4 (1 1)
Image Size                      : 1000x500
Megapixels                      : 0.500
Date/Time Created               : 2019:10:16 19:01:00+00:00
Date/Time Original              : 2019:10:16 19:01:00+00:00
```

Appendix IV: Output of PyExifTool library testing script

```
Introduce the name of an image to analyze its metadata:

Tivenys.jpg
SourceFile Tivenys.jpg
ExifTool:ExifToolVersion 11.99
File:FileName Tivenys.jpg
File:Directory .
File:FileSize 2137863
File:FileModifyDate 2021:03:16 16:58:10+01:00
File:FileAccessDate 2021:03:16 16:59:03+01:00
File:FileInodeChangeDate 2021:03:16 16:59:03+01:00
File:FilePermissions 644
File:FileType JPEG
File:FileTypeExtension JPG
File:MIMEType image/jpeg
File:ExifByteOrder MM
File:CurrentIPTCDigest 6973f3cedf1cc0919df9208428c7eb84
File:ImageWidth 3724
File:ImageHeight 2095
File:EncodingProcess 0
File:BitsPerSample 8
File:ColorComponents 3
File:YCbCrSubSampling 1 1
JFIF:JFIFVersion 1 1
JFIF:ResolutionUnit 0
JFIF:XResolution 72
JFIF:YResolution 72
EXIF:Make Apple
EXIF:Model iPhone 7
EXIF:Orientation 1
EXIF:XResolution 72
EXIF:YResolution 72
EXIF:ResolutionUnit 2
EXIF:Software Instagram
EXIF:ModifyDate 2020:01:07 17:59:46
EXIF:ExposureTime 0.01
EXIF:FNumber 1.8
EXIF:ExposureProgram 2
EXIF:ISO 32
EXIF:DateTimeOriginal 2020:01:07 17:59:46
EXIF:CreateDate 2020:01:07 17:59:46
EXIF:OffsetTime +01:00
EXIF:OffsetTimeOriginal +01:00
EXIF:OffsetTimeDigitized +01:00
EXIF:ShutterSpeedValue 0.0099989999964845
EXIF:ApertureValue 1.79999999993144
EXIF:BrightnessValue 5.405560449
EXIF:ExposureCompensation 0
EXIF:MeteringMode 5
EXIF:Flash 16
EXIF:FocalLength 3.99
EXIF:SubjectArea 1861 1047 2048 922
EXIF:SubSecTimeOriginal 323
EXIF:SubSecTimeDigitized 323
EXIF:ColorSpace 1
EXIF:ExifImageWidth 3724
EXIF:ExifImageHeight 2095
EXIF:SensingMethod 2
EXIF:SceneType 1
EXIF:ExposureMode 0
EXIF:WhiteBalance 0
EXIF:FocalLengthIn35mmFormat 33
EXIF:LensInfo 3.99000001 3.99000001 1.8 1.8
EXIF:LensMake Apple
EXIF:LensModel iPhone 7 back camera 3.99mm f/1.8
MakerNotes:RunTimeFlags 1
MakerNotes:RunTimeValue 26935085311125
MakerNotes:RunTimeEpoch 0
MakerNotes:RunTimeScale 1000000000
```

```
MakerNotes:AccelerationVector -0.9902680512 -0.01313300897 0.1290269494
XMP:XMPToolkit Image::ExifTool 12.21
XMP:Creator Jordi Domenech Fons
XMP:DateCreated 2020:01:07 17:59:46.323
XMP:CreateDate 2020:01:07 17:59:46.323
XMP:CreatorTool Instagram
XMP:ModifyDate 2020:01:07 17:59:46
IPTC:CodedCharacterSet %G
IPTC:EnvelopeRecordVersion 4
IPTC:ApplicationRecordVersion 2
IPTC:DigitalCreationTime 17:59:46
IPTC:DigitalCreationDate 2020:01:07
IPTC:DateCreated 2020:01:07
IPTC:TimeCreated 17:59:46
IPTC:City Tivenys
Photoshop:IPTCDigest 5506f87b54ab2687ff9a954d70603f29
Composite:RunTimeSincePowerUp 26935.085311125
Composite:Aperture 1.8
Composite:ImageSize 3724 2095
Composite:Megapixels 7.80178
Composite:ScaleFactor35efl 8.27067669172932
Composite:ShutterSpeed 0.01
Composite:SubSecCreateDate 2020:01:07 17:59:46.323+01:00
Composite:SubSecDateTimeOriginal 2020:01:07 17:59:46.323+01:00
Composite:SubSecModifyDate 2020:01:07 17:59:46+01:00
Composite:DateTimeCreated 2020:01:07 17:59:46
Composite:DigitalCreationDateTime 2020:01:07 17:59:46
Composite:CircleOfConfusion 0.00363286605785387
Composite:FOV 57.2209676644142
Composite:FocalLength35efl 33
Composite:HyperfocalDistance 2.43457916123253
Composite:LightValue 9.98370619265935
```

Appendix V: PyExifTool testing script

```
1.  import exiftool
2.  import re
3.
4.  while (True):
5.      try:
6.          print("Introduce the name of an image to analyze its metadata: ")
7.
8.          file = input()
9.          # Extract image metadata from the file
10.         with exiftool.ExifTool() as et:
11.             metadata = et.get_metadata(file)
12.
13.         break
14.
15.     except ValueError:
16.         print("Error: File not found!")
17.
18. # Print all the metadata
19. for m in range(0, len(metadata)):
20.     print(list(metadata)[m],list(metadata.values())[m])
21.
```

Appendix VI: PyExifTool Library

```
# -*- coding: utf-8 -*-
# PyExifTool <http://github.com/smarnach/pyexiftool>
# Copyright 2012 Sven Marnach

# This file is part of PyExifTool.
#
# PyExifTool is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the licence, or
# (at your option) any later version, or the BSD licence.
#
# PyExifTool is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
#
# See COPYING.GPL or COPYING.BSD for more details.

"""
PyExifTool is a Python library to communicate with an instance of Phil
Harvey's excellent ExifTool_ command-line application.  The library
provides the class :py:class:`ExifTool` that runs the command-line
tool in batch mode and features methods to send commands to that
program, including methods to extract meta-information from one or
more image files.  Since ``exiftool`` is run in batch mode, only a
single instance needs to be launched and can be reused for many
queries.  This is much more efficient than launching a separate
process for every single query.

.. _ExifTool: http://www.sno.phy.queensu.ca/~phil/exiftool/

The source code can be checked out from the github repository with

::

    git clone git://github.com/smarnach/pyexiftool.git

Alternatively, you can download a tarball_.  There haven't been any
releases yet.

.. _tarball: https://github.com/smarnach/pyexiftool/tarball/master

PyExifTool is licenced under GNU GPL version 3 or later.

Example usage::

    import exiftool

    files = ["a.jpg", "b.png", "c.tif"]
    with exiftool.ExifTool() as et:
        metadata = et.get_metadata_batch(files)
    for d in metadata:
        print("{:20.20} {:20.20}".format(d["SourceFile"],
                                         d["EXIF:DateTimeOriginal"]))
"""

from __future__ import unicode_literals

import sys
import subprocess
import os
import json
import warnings
import codecs

```

```
65. try:        # Py3k compatibility
66.     basestring
67. except NameError:
68.     basestring = (bytes, str)
69.
70. executable = "exiftool"
71. """The name of the executable to run.
72.
73. If the executable is not located in one of the paths listed in the
74. ``PATH`` environment variable, the full path should be given here.
75. """
76.
77. # Sentinel indicating the end of the output of a sequence of commands.
78. # The standard value should be fine.
79. sentinel = b"{ready}"
80.
81. # The block size when reading from exiftool.  The standard value
82. # should be fine, though other values might give better performance in
83. # some cases.
84. block_size = 4096
85.
86. # This code has been adapted from Lib/os.py in the Python source tree
87. # (sha1 265e36e277f3)
88. def _fscodec():
89.     encoding = sys.getfilesystemencoding()
90.     errors = "strict"
91.     if encoding != "mbcs":
92.         try:
93.             codecs.lookup_error("surrogateescape")
94.         except LookupError:
95.             pass
96.         else:
97.             errors = "surrogateescape"
98.
99.     def fsencode(filename):
100.          """
101.          Encode filename to the filesystem encoding with 'surrogateescape' error
102.          handler, return bytes unchanged. On Windows, use 'strict' error handler if
103.          the file system encoding is 'mbcs' (which is the default encoding).
104.          """
105.          if isinstance(filename, bytes):
106.              return filename
107.          else:
108.              return filename.encode(encoding, errors)
109.
110.      return fsencode
111.
112.  fsencode = _fscodec()
113.  del _fscodec
114.
115.  class ExifTool(object):
116.      """Run the `exiftool` command-line tool and communicate to it.
117.
118.      You can pass the file name of the ``exiftool`` executable as an
119.      argument to the constructor.  The default value ``exiftool`` will
120.      only work if the executable is in your ``PATH``.
121.
122.      Most methods of this class are only available after calling
123.      :py:meth:`start()`, which will actually launch the subprocess.  To
124.      avoid leaving the subprocess running, make sure to call
125.      :py:meth:`terminate()` method when finished using the instance.
126.      This method will also be implicitly called when the instance is
127.      garbage collected, but there are circumstance when this won't ever
128.      happen, so you should not rely on the implicit process
129.      termination.  Subprocesses won't be automatically terminated if
130.      the parent process exits, so a leaked subprocess will stay around
131.      until manually killed.
132.
```

```
133.      A convenient way to make sure that the subprocess is terminated is
134.      to use the :py:class:`ExifTool` instance as a context manager::
135.
136.          with ExifTool() as et:
137.              ...
138.
139.      .. warning:: Note that there is no error handling.  Nonsensical
140.         options will be silently ignored by exiftool, so there's not
141.         much that can be done in that regard.  You should avoid passing
142.         non-existent files to any of the methods, since this will lead
143.         to undefied behaviour.
144.
145.      .. py:attribute:: running
146.
147.         A Boolean value indicating whether this instance is currently
148.         associated with a running subprocess.
149.      """
150.
151.      def __init__(self, executable_=None):
152.          if executable_ is None:
153.              self.executable = executable
154.          else:
155.              self.executable = executable_
156.          self.running = False
157.
158.      def start(self):
159.          """Start an ``exiftool`` process in batch mode for this instance.
160.
161.          This method will issue a ``UserWarning`` if the subprocess is
162.          already running.  The process is started with the ``-G`` and
163.          ``-n`` as common arguments, which are automatically included
164.          in every command you run with :py:meth:`execute()`.
165.          """
166.          if self.running:
167.              warnings.warn("ExifTool already running; doing nothing.")
168.              return
169.          with open(os.devnull, "w") as devnull:
170.              self._process = subprocess.Popen(
171.                  [self.executable, "-stay_open", "True",  "-@", "-",
172.                   "-common_args", "-G", "-n"],
173.                  stdin=subprocess.PIPE, stdout=subprocess.PIPE,
174.                  stderr=devnull)
175.          self.running = True
176.
177.      def terminate(self):
178.          """Terminate the ``exiftool`` process of this instance.
179.
180.          If the subprocess isn't running, this method will do nothing.
181.          """
182.          if not self.running:
183.              return
184.          self._process.stdin.write(b"-stay_open\nFalse\n")
185.          self._process.stdin.flush()
186.          self._process.communicate()
187.          del self._process
188.          self.running = False
189.
190.      def __enter__(self):
191.          self.start()
192.          return self
193.
194.      def __exit__(self, exc_type, exc_val, exc_tb):
195.          self.terminate()
196.
197.      def __del__(self):
198.          self.terminate()
199.
200.      def execute(self, *params):
```

```
201.          """Execute the given batch of parameters with ``exiftool``.
202.
203.          This method accepts any number of parameters and sends them to
204.          the attached ``exiftool`` process.  The process must be
205.          running, otherwise ``ValueError`` is raised.  The final
206.          ``-execute`` necessary to actually run the batch is appended
207.          automatically; see the documentation of :py:meth:`start()` for
208.          the common options.  The ``exiftool`` output is read up to the
209.          end-of-output sentinel and returned as a raw ``bytes`` object,
210.          excluding the sentinel.
211.
212.          The parameters must also be raw ``bytes``, in whatever
213.          encoding exiftool accepts.  For filenames, this should be the
214.          system's filesystem encoding.
215.
216.          .. note:: This is considered a low-level method, and should
217.             rarely be needed by application developers.
218.          """
219.          if not self.running:
220.              raise ValueError("ExifTool instance not running.")
221.          self._process.stdin.write(b"\n".join(params + (b"-execute\n",)))
222.          self._process.stdin.flush()
223.          output = b""
224.          fd = self._process.stdout.fileno()
225.          while not output[-32:].strip().endswith(sentinel):
226.              output += os.read(fd, block_size)
227.          return output.strip()[:-len(sentinel)]
228.
229.      def execute_json(self, *params):
230.          """Execute the given batch of parameters and parse the JSON output.
231.
232.          This method is similar to :py:meth:`execute()`.  It
233.          automatically adds the parameter ``-j`` to request JSON output
234.          from ``exiftool`` and parses the output.  The return value is
235.          a list of dictionaries, mapping tag names to the corresponding
236.          values.  All keys are Unicode strings with the tag names
237.          including the ExifTool group name in the format <group>:<tag>.
238.          The values can have multiple types.  All strings occurring as
239.          values will be Unicode strings.  Each dictionary contains the
240.          name of the file it corresponds to in the key ``"SourceFile"``.
241.
242.          The parameters to this function must be either raw strings
243.          (type ``str`` in Python 2.x, type ``bytes`` in Python 3.x) or
244.          Unicode strings (type ``unicode`` in Python 2.x, type ``str``
245.          in Python 3.x).  Unicode strings will be encoded using
246.          system's filesystem encoding.  This behaviour means you can
247.          pass in filenames according to the convention of the
248.          respective Python version – as raw strings in Python 2.x and
249.          as Unicode strings in Python 3.x.
250.          """
251.          params = map(fsencode, params)
252.          return json.loads(self.execute(b"-j", *params).decode("utf-8"))
253.
254.      def get_metadata_batch(self, filenames):
255.          """Return all meta-data for the given files.
256.
257.          The return value will have the format described in the
258.          documentation of :py:meth:`execute_json()`.
259.          """
260.          return self.execute_json(*filenames)
261.
262.      def get_metadata(self, filename):
263.          """Return meta-data for a single file.
264.
265.          The returned dictionary has the format described in the
266.          documentation of :py:meth:`execute_json()`.
267.          """
268.          return self.execute_json(filename)[0]
```

```
269.
270.      def get_tags_batch(self, tags, filenames):
271.          """Return only specified tags for the given files.
272.
273.          The first argument is an iterable of tags.  The tag names may
274.          include group names, as usual in the format <group>:<tag>.
275.
276.          The second argument is an iterable of file names.
277.
278.          The format of the return value is the same as for
279.          :py:meth:`execute_json()`.
280.          """
281.          # Explicitly ruling out strings here because passing in a
282.          # string would lead to strange and hard-to-find errors
283.          if isinstance(tags, basestring):
284.              raise TypeError("The argument 'tags' must be "
285.                              "an iterable of strings")
286.          if isinstance(filenames, basestring):
287.              raise TypeError("The argument 'filenames' must be "
288.                              "an iterable of strings")
289.          params = ["-" + t for t in tags]
290.          params.extend(filenames)
291.          return self.execute_json(*params)
292.
293.      def get_tags(self, tags, filename):
294.          """Return only specified tags for a single file.
295.
296.          The returned dictionary has the format described in the
297.          documentation of :py:meth:`execute_json()`.
298.          """
299.          return self.get_tags_batch(tags, [filename])[0]
300.
301.      def get_tag_batch(self, tag, filenames):
302.          """Extract a single tag from the given files.
303.
304.          The first argument is a single tag name, as usual in the
305.          format <group>:<tag>.
306.
307.          The second argument is an iterable of file names.
308.
309.          The return value is a list of tag values or ``None`` for
310.          non-existent tags, in the same order as ``filenames``.
311.          """
312.          data = self.get_tags_batch([tag], filenames)
313.          result = []
314.          for d in data:
315.              d.pop("SourceFile")
316.              result.append(next(iter(d.values()), None))
317.          return result
318.
319.      def get_tag(self, tag, filename):
320.          """Extract a single tag from a single file.
321.
322.          The return value is the value of the specified tag, or
323.          ``None`` if this tag was not found in the file.
324.          """
325.          return self.get_tag_batch(tag, [filename])[0]
326.
```

Appendix VII: HelloWorld.py

```
1.  import jarray
2.  import inspect
3.  from java.lang import System
4.  from java.util.logging import Level
5.  from org.sleuthkit.datamodel import SleuthkitCase
6.  from org.sleuthkit.datamodel import AbstractFile
7.  from org.sleuthkit.datamodel import ReadContentInputStream
8.  from org.sleuthkit.datamodel import BlackboardArtifact
9.  from org.sleuthkit.datamodel import BlackboardAttribute
10. from org.sleuthkit.datamodel import TskData
11. from org.sleuthkit.autopsy.ingest import IngestModule
12. from org.sleuthkit.autopsy.ingest.IngestModule import IngestModuleException
13. from org.sleuthkit.autopsy.ingest import DataSourceIngestModule
14. from org.sleuthkit.autopsy.ingest import FileIngestModule
15. from org.sleuthkit.autopsy.ingest import IngestModuleFactoryAdapter
16. from org.sleuthkit.autopsy.ingest import IngestMessage
17. from org.sleuthkit.autopsy.ingest import IngestServices
18. from org.sleuthkit.autopsy.ingest import ModuleDataEvent
19. from org.sleuthkit.autopsy.coreutils import Logger
20. from org.sleuthkit.autopsy.casemodule import Case
21. from org.sleuthkit.autopsy.casemodule.services import Services
22. from org.sleuthkit.autopsy.casemodule.services import FileManager
23. from org.sleuthkit.autopsy.casemodule.services import Blackboard
24.
25. # Factory that defines the name and details of the module and allows Autopsy
26. # to create instances of the modules that will do the anlaysis.
27. class HelloWorldFileIngestModuleFactory(IngestModuleFactoryAdapter):
28.
29.     # TODO: give it a unique name.  Will be shown in module list, logs, etc.
30.     moduleName = "Hello World"
31.
32.     def getModuleDisplayName(self):
33.         return self.moduleName
34.
35.     # TODO: Give it a description
36.     def getModuleDescription(self):
37.         return "Basic Autopsy module for testing the development of Python Ingest
    Modules"
38.
39.     def getModuleVersionNumber(self):
40.         return "1.0"
41.
42.     # Return true if module wants to get called for each file
43.     def isFileIngestModuleFactory(self):
44.         return True
45.
46.     # can return null if isFileIngestModuleFactory returns false
47.     def createFileIngestModule(self, ingestOptions):
48.         return HelloWorldFileIngestModule()
49.
50. # File-level ingest module.  One gets created per thread.
51. # Looks at the attributes of the passed in file.
52. class HelloWorldFileIngestModule(FileIngestModule):
53.
54.     _logger = Logger.getLogger(HelloWorldFileIngestModuleFactory.moduleName)
55.
56.     def log(self, level, msg):
57.         self._logger.logp(level, self.__class__.__name__, inspect.stack()[1][3], msg)
58.
59.     # Where any setup and configuration is done
60.     # 'context' is an instance of org.sleuthkit.autopsy.ingest.IngestJobContext.
61.     # See: http://sleuthkit.org/autopsy/docs/api-
    docs/latest/classorg_1_1sleuthkit_1_1autopsy_1_1ingest_1_1_ingest_job_context.html
62.     # Add any setup code that you need here.
```

```
63.     def startUp(self, context):
64.         self.filesFound = 0
65.
66.         # Throw an IngestModule.IngestModuleException exception if there was a
    problem setting up
67.         # raise IngestModuleException("Oh No!")
68.         pass
69.
70.     # Where the analysis is done.  Each file will be passed into here.
71.     # The 'file' object being passed in is of type
    org.sleuthkit.datamodel.AbstractFile.
72.     # See: http://www.sleuthkit.org/sleuthkit/docs/jni-
    docs/latest/classorg_1_1sleuthkit_1_1datamodel_1_1_abstract_file.html
73.     # Add your analysis code in here.
74.     def process(self, file):
75.         # Skip non-files
76.         if ((file.getType() == TskData.TSK_DB_FILES_TYPE_ENUM.UNALLOC_BLOCKS) or
77.             (file.getType() == TskData.TSK_DB_FILES_TYPE_ENUM.UNUSED_BLOCKS) or
78.             (file.isFile() == False)):
79.             return IngestModule.ProcessResult.OK
80.
81.         # If there is the sentence "HelloWorld" in the title of a file, we create an
    artifact on the blackboard
82.         # with this "interesting" file
83.         if "HelloWorld" in file.getName():
84.
85.             self.log(Level.INFO, "Found a Hello World file")
86.             self.filesFound +=1
87.
88.             # Make an artifact on the blackboard
89.             # Create the artifact as a TSK_INTERESTING_FILE_HIT
90.             art =
    file.newArtifact(BlackboardArtifact.ARTIFACT_TYPE.TSK_INTERESTING_FILE_HIT)
91.             # Create the attribute. We put SET_NAME to see the name of the artifact
    in the blackboard
92.             # on the section Interesting Items; if not, we will only see a "yellow
    triangle" in the part
93.             # where the file is allocated indicating that this file has interesting
    results associated with it
94.             att =
    BlackboardAttribute(BlackboardAttribute.ATTRIBUTE_TYPE.TSK_SET_NAME.getTypeID(),
95.                                         HelloWorldFileIngestModuleFactory.moduleName,
    "This is a Hello World File!")
96.             # Add the attribute to the artifact
97.             art.addAttribute(att)
98.
99.             # Fires an event to notify the UI and others that there is a new artifact
100.              # So that the UI updates and refreshes with the new artifacts when the
    module is executed
101.
    IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(HelloWorldFileIngest
    ModuleFactory.moduleName,
102.                 BlackboardArtifact.ARTIFACT_TYPE.TSK_INTERESTING_FILE_HIT, None))
103.
104.         return IngestModule.ProcessResult.OK
105.
106.     # Where any shutdown code is run and resources are freed.
107.     # TODO: Add any shutdown code that you need here.
108.     def shutDown(self):
109.         # As a final part of this example, we'll send a message to the ingest inbox
    with the number of files found (in this thread)
110.         message = IngestMessage.createMessage(
111.             IngestMessage.MessageType.DATA,
    HelloWorldFileIngestModuleFactory.moduleName,
112.                 str(self.filesFound) + " files found")
113.         ingestServices = IngestServices.getInstance().postMessage(message)
114.
```

Appendix VIII: Installation procedure of the Image Metadata Analyzer Module

For the proper installation of the module you will have to install first of all the *Jython* Installer on your computer from the following URL [17].

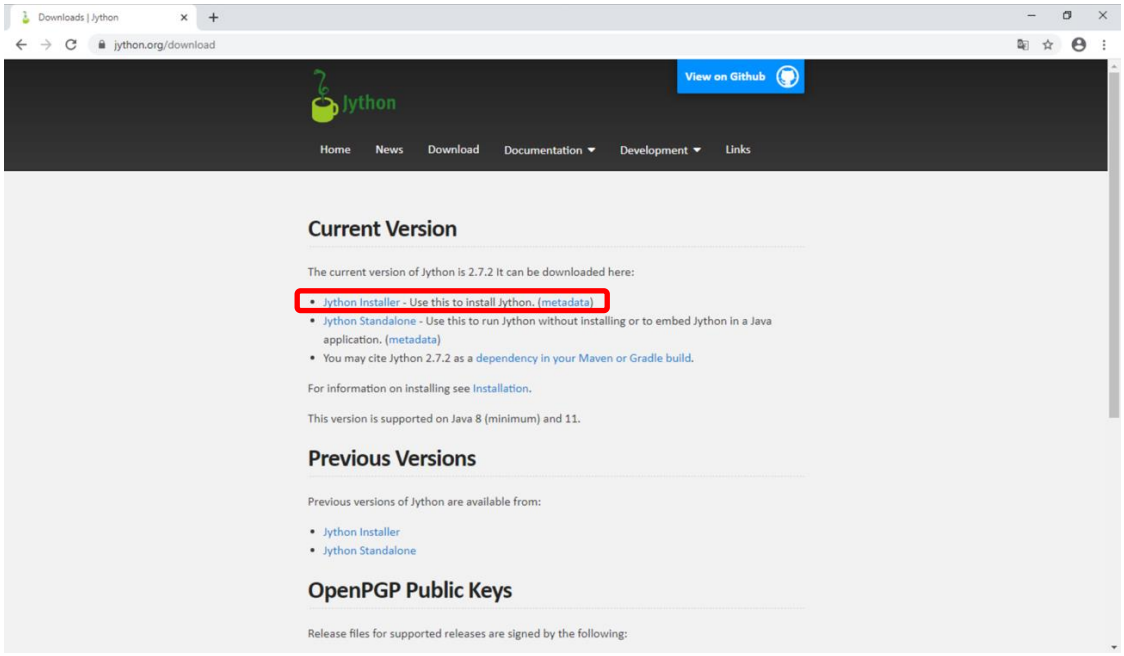


Figure 38. Jython Download

We only have to follow the installation procedure. In my case, I installed the *Standard* version on the *C:\jython2.7.2* folder of my computer.

Figure 39. Jython Installation (I)

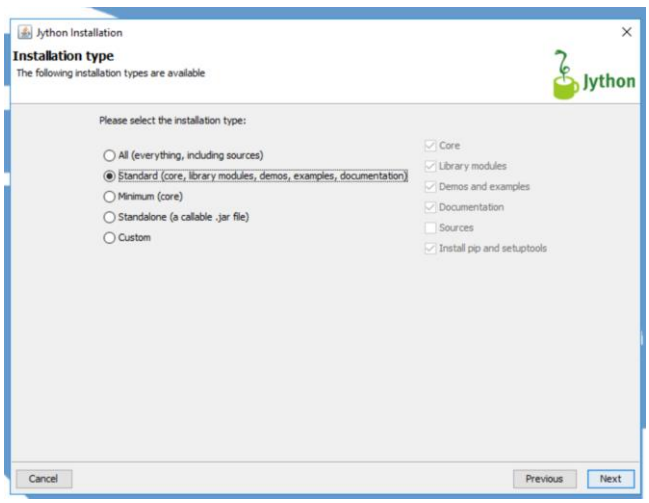


Figure 40. Jython Installation (II)

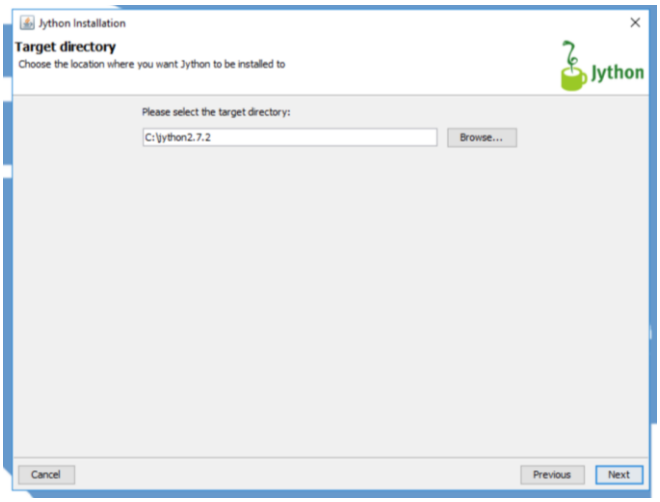


Figure 41. Jython Installation (III)

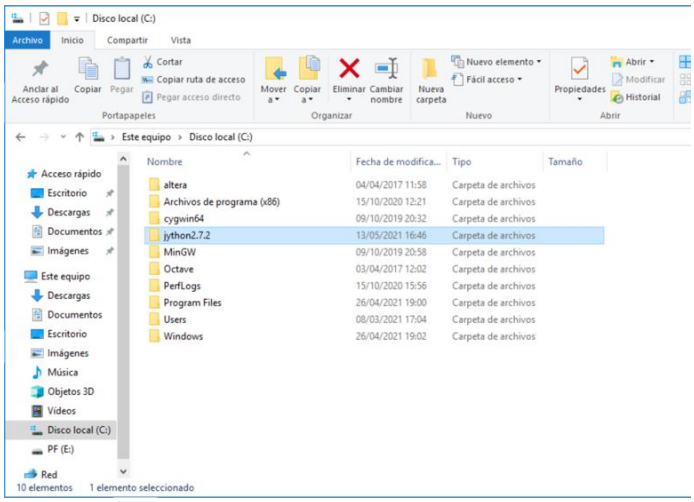


Figure 42. Jython Installation (IV)

Once the installation of *Jython* is done, we are going to install the *ExifTool* command-line tool developed by Phil Harvey [6].

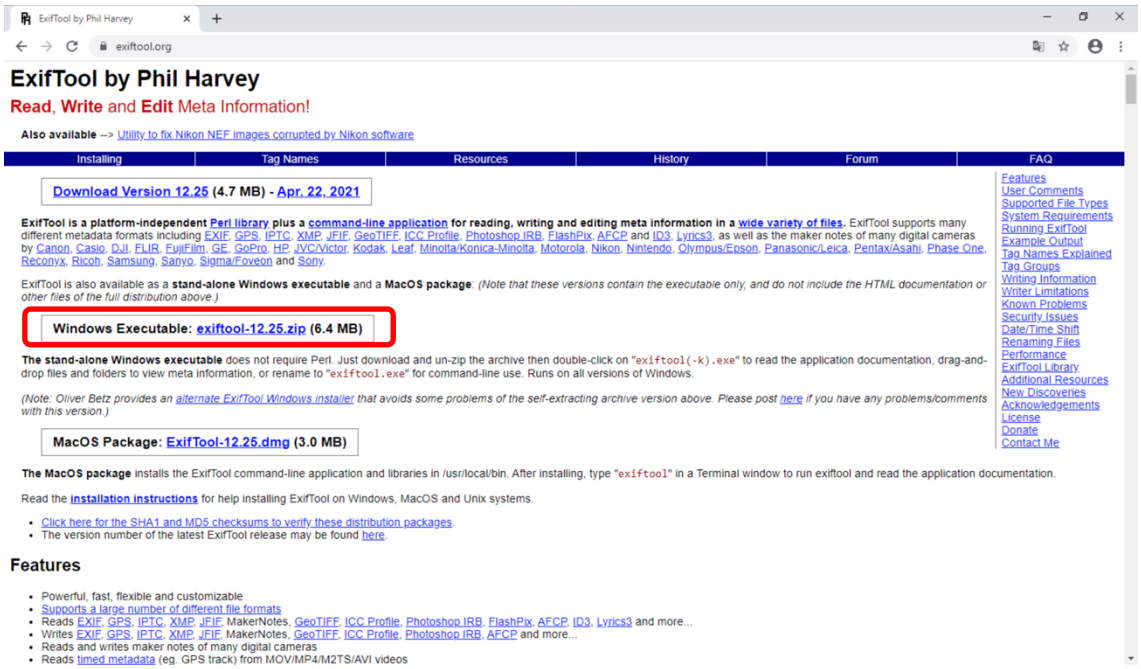


Figure 43. ExifTool Download

A zip file will be downloaded with the *.exe* inside. We will only need to copy and paste it on the proper directory: ***C:\Windows*** .

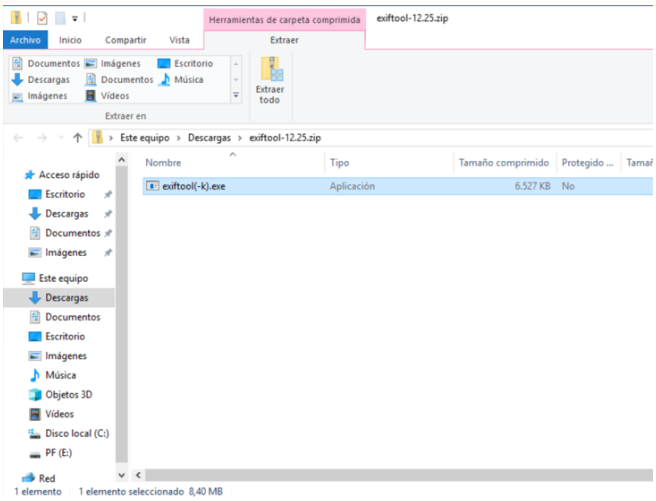


Figure 44. ExifTool Installation (I)

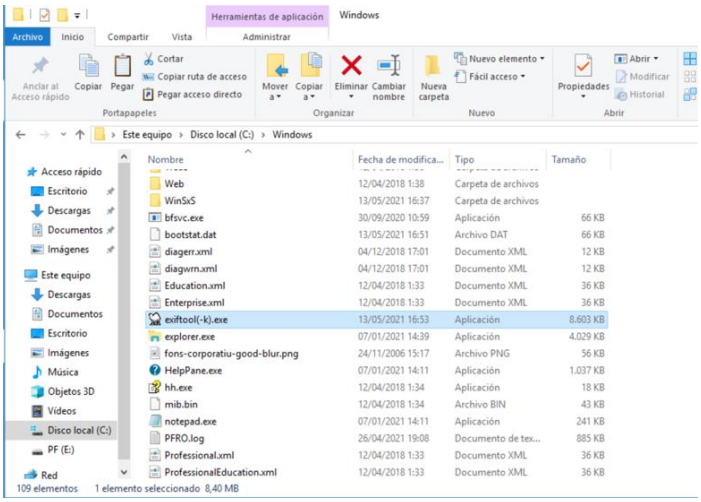


Figure 45. ExifTool Installation (II)

Once the executable is copied, we will have to rename it to “exiftool.exe” for command-line use: our module uses the *ExifTool* software as a command-line.

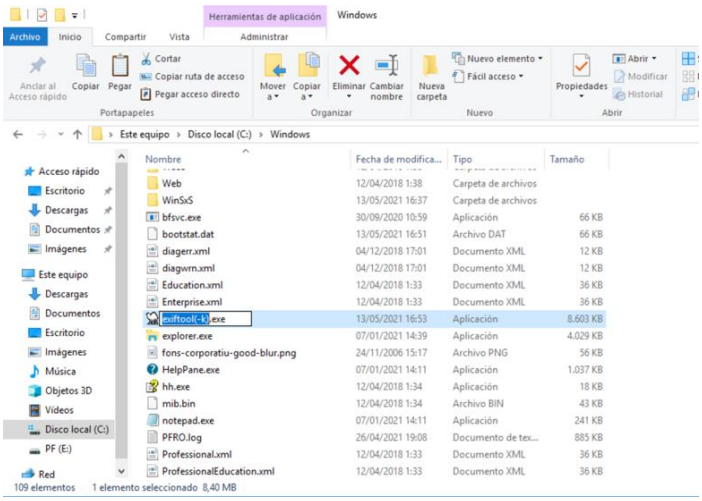


Figure 46. ExifTool Installation (III)

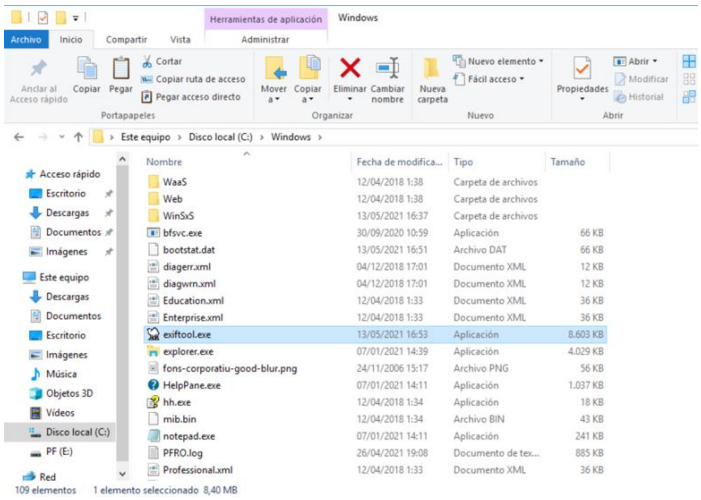


Figure 47. ExifTool Installation (IV)

After installing both *Jython* and *ExifTool* we are ready to install the Image Metadata Analyzer Module in Autopsy. For doing so, we will download the module code from my following public repository in GitHub as a ZIP file [18].

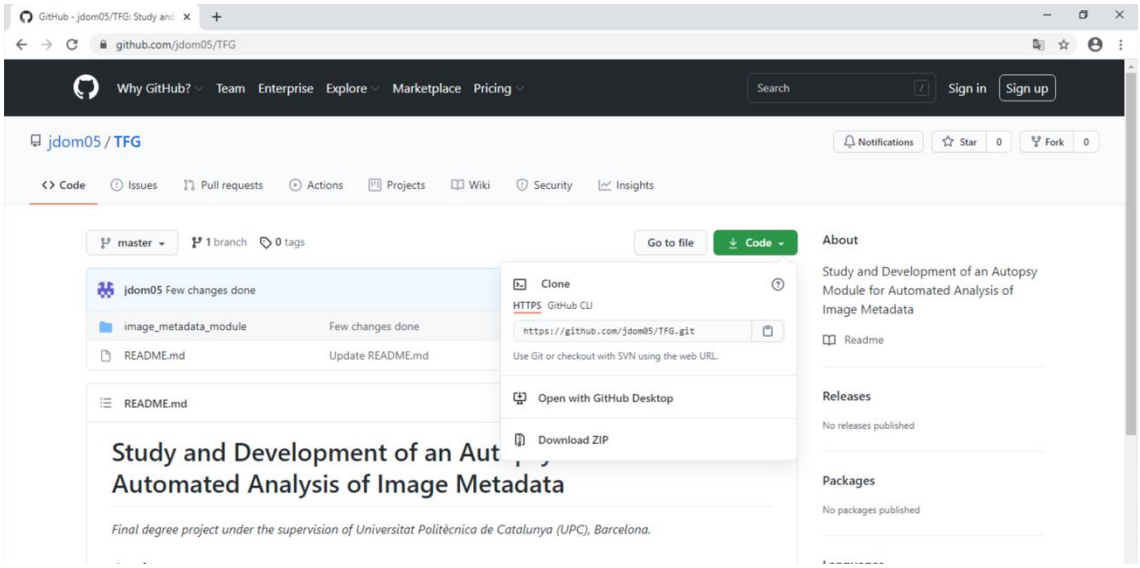


Figure 48. Image Metadata Analyzer Module github repository

When we have downloaded it, we only have to copy the folder “image_metadata_module”, that contains the module code, to the “python_module” folder of Autopsy (*Tools* > *Python Plugins*, see *Figure 50* and *Figure 51)*. Look at chapter 2 and 3 to learn how to add new Python modules in Autopsy.

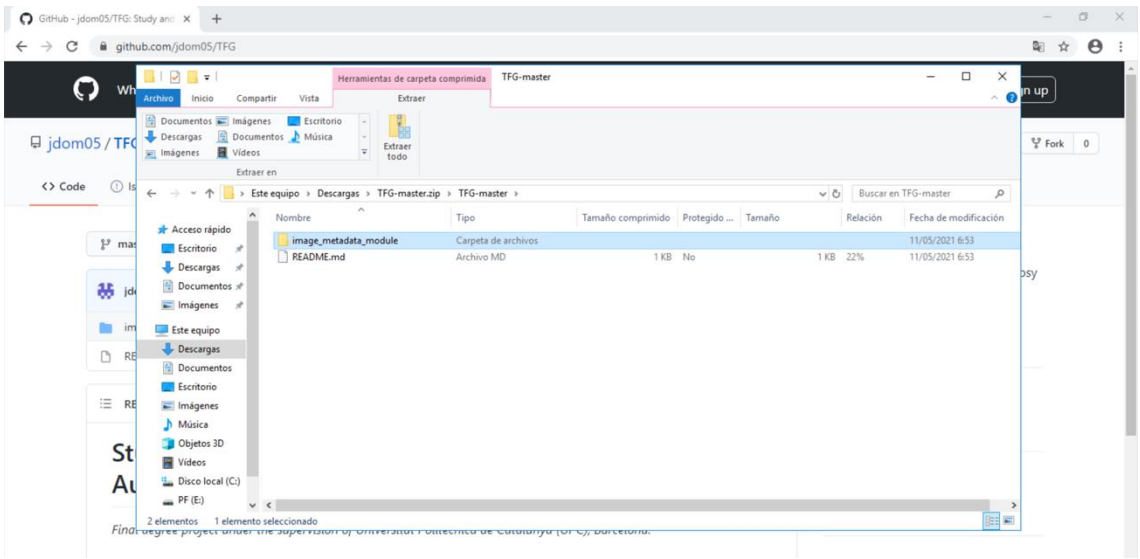


Figure 49. Image Metadata Analyzer Module Download

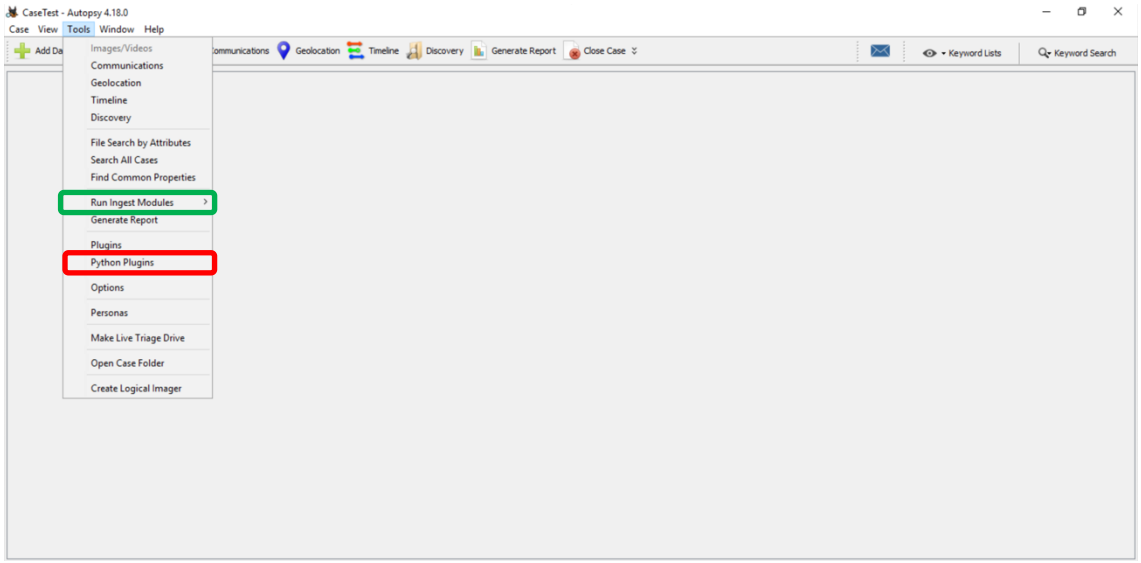


Figure 50. Image Metadata Analyzer Module Installation (I)

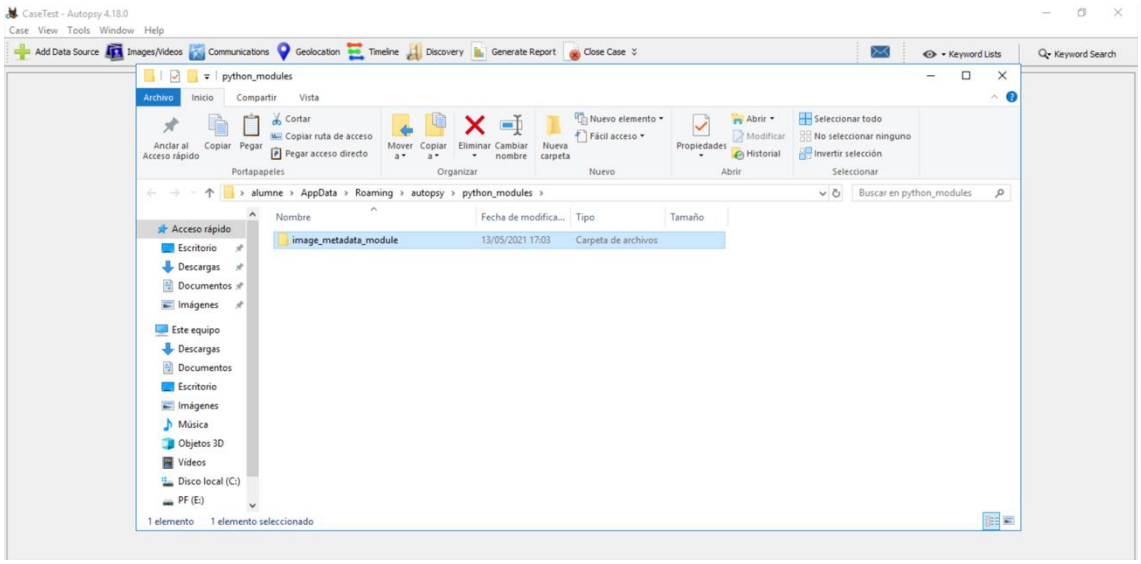


Figure 51. Image Metadata Analyzer Module Installation (II)

Once the Image Metadata Analyzer Module is installed in the proper path, we are finally able to use it in our case! For doing so, we go to *Tools* > *Run Ingest Modules* (see *Figure 50)* and select the Data Source we want the module run. We can search the module in the pull of modules that Autopsy has and we will find it as ***Image Metadata Analyzer Module***.

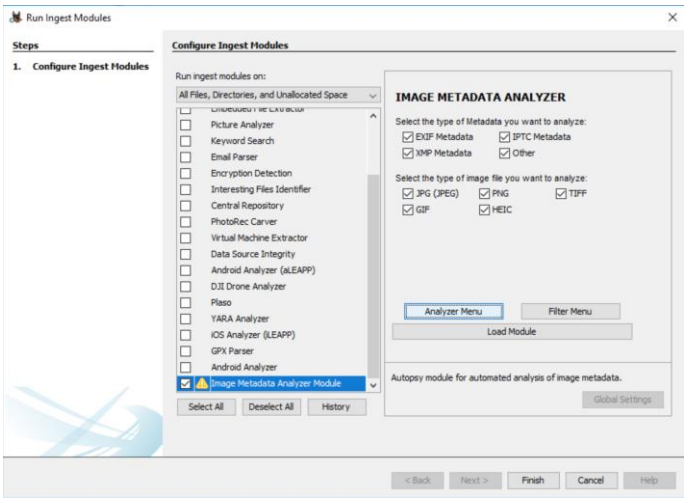


Figure 52. Image Metadata Analyzer Module GUI (I)

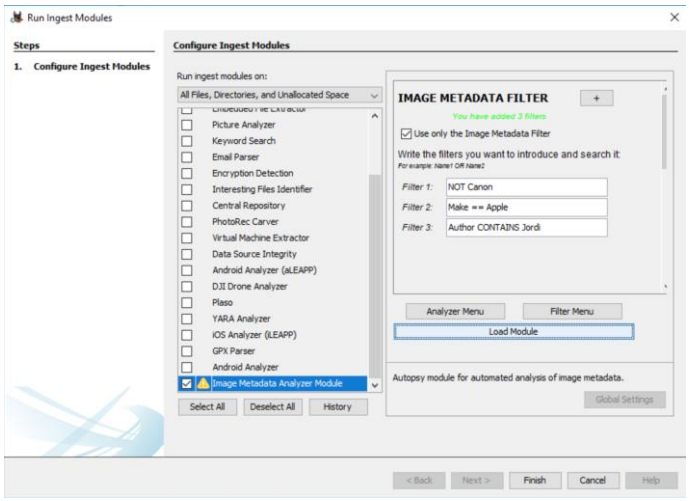


Figure 53. Image Metadata Analyzer Module GUI (II)

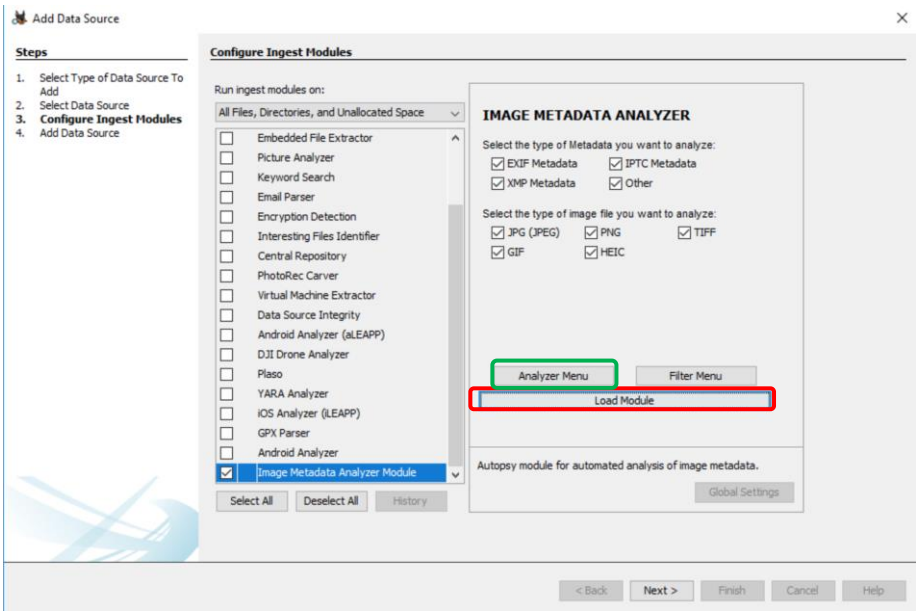
CAUTION: The module works with ***Jython 2.7.2***, ***ExifTool 12.25*** and ***Autopsy 4.18.0***. It is not checked if it also runs with further versions of the programs.

Appendix IX: Usage of the Image Metadata Analyzer Module

Once we have downloaded the module, we are ready to use it. In this section, I will show you a basic example of how to properly employ it. We are going to analyze a folder that contains 6 different images.

First of all, we start analyzing only the metadata of each image. Accordingly, we select in the Analyzer Menu of the GUI the type of metadata we want to analyze and the type of image file; we select all the boxes because we want to analyze all the available metadata of the 5 different type of images (jpg, png, tiff, gif and heic). After that, we load the module just clicking on the button “Load Module”.

CAUTION: It is important to click on the “Load Module” button every time we want to load the module. If we do not do it, the changes will not be saved and the module will not be executed correctly.



Figue 54. Analyzer Menu GUI

After loading the module, the results of the analyzed images will appear on the main page of Autopsy.

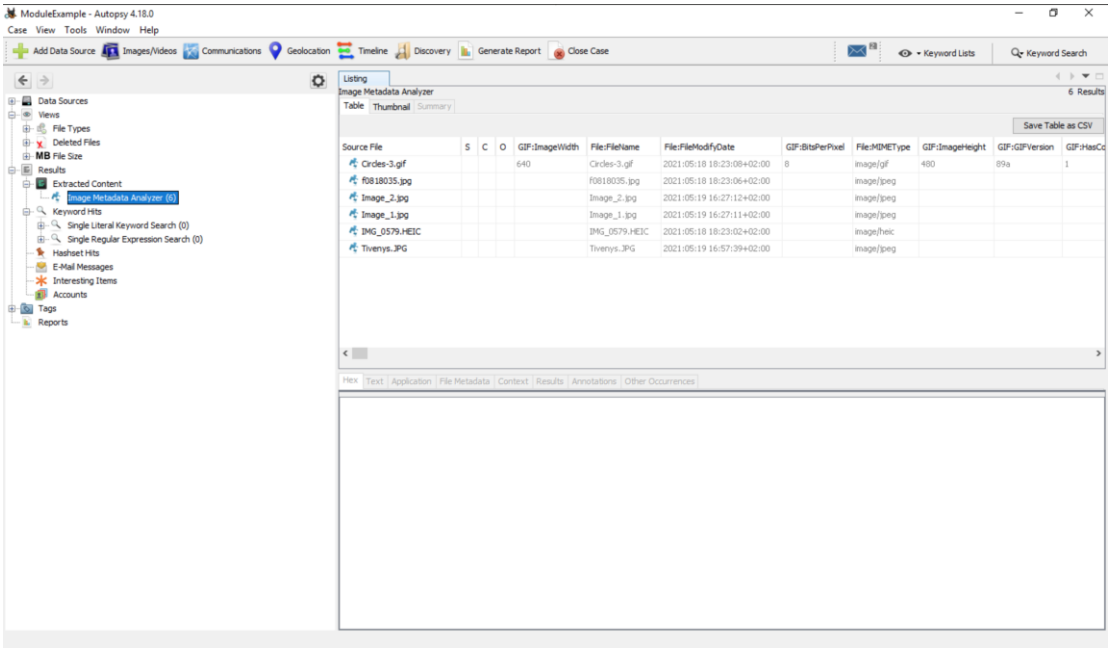


Figure 55. Analyzer Mode output in Autopsy

Secondly, once the metadata of each image is analyzed, we are going to filter images by its metadata. To do so, we load again the module but this time using the “Filter Menu”.

In the Filter Menu we will select the box “Use only the Image Metadata Filter”, because we have analyzed the image metadata before and we do not want to do it again. After that, we can add as much filters as we want (clicking on the “+” button) and we load the module again (“Load Module” button).

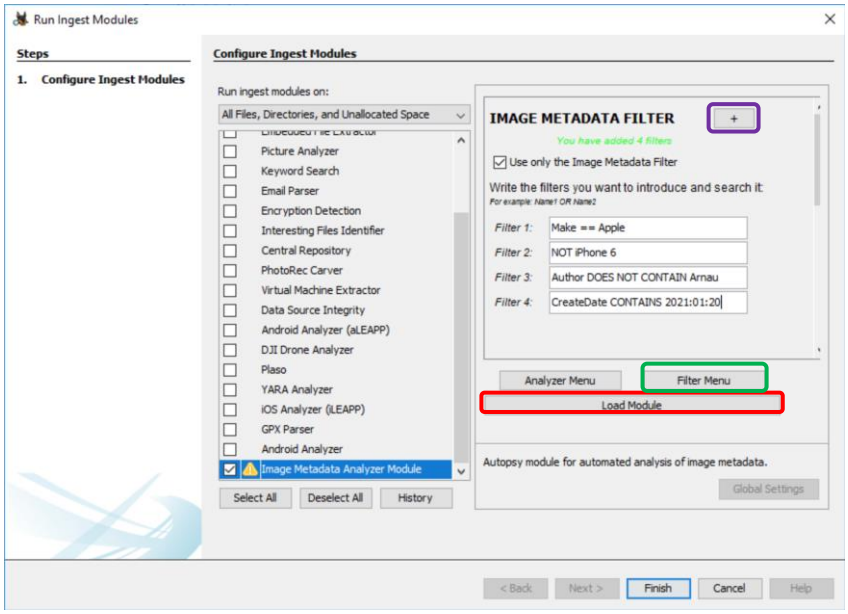


Figure 56. Filter Menu GUI (I)

The results will be shown as Interesting Items in the Tree Viewer of Autopsy.

You can see that the name of Interesting Items is the filter used. Apart from that, in the images filtered there is a section of comments that specifies you the metadata fields where the filters have been found and its value.

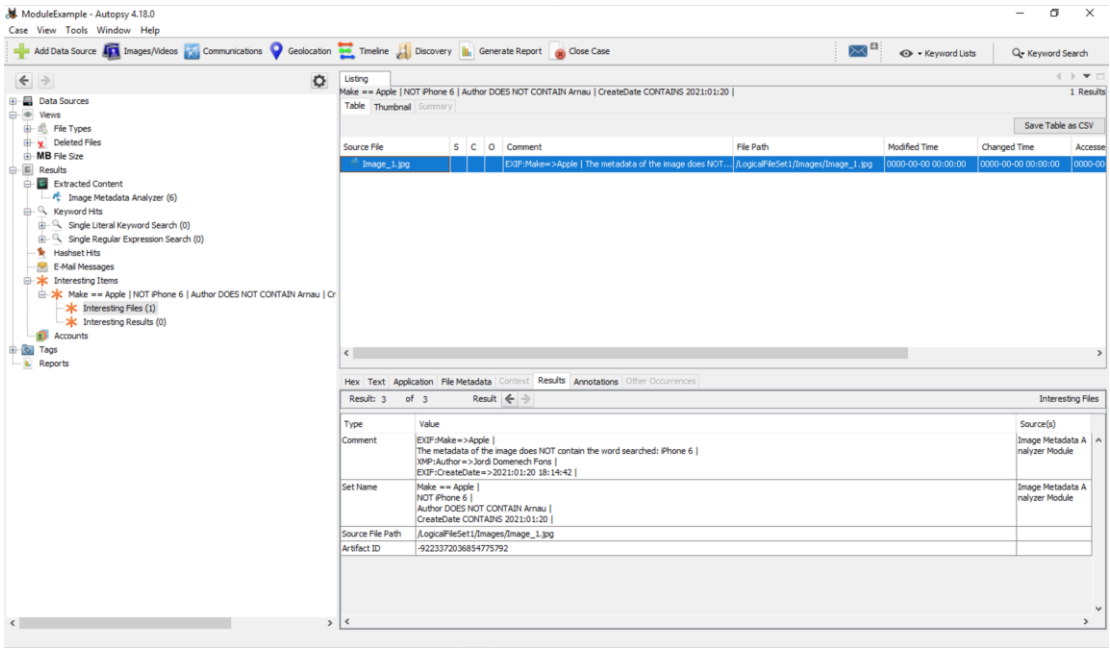


Figure 57. Filter Mode output in Autopsy (I)

If we want to insert new filters we only have to follow the same procedure making use only of the Image Metadata Filter in the Filter Menu:

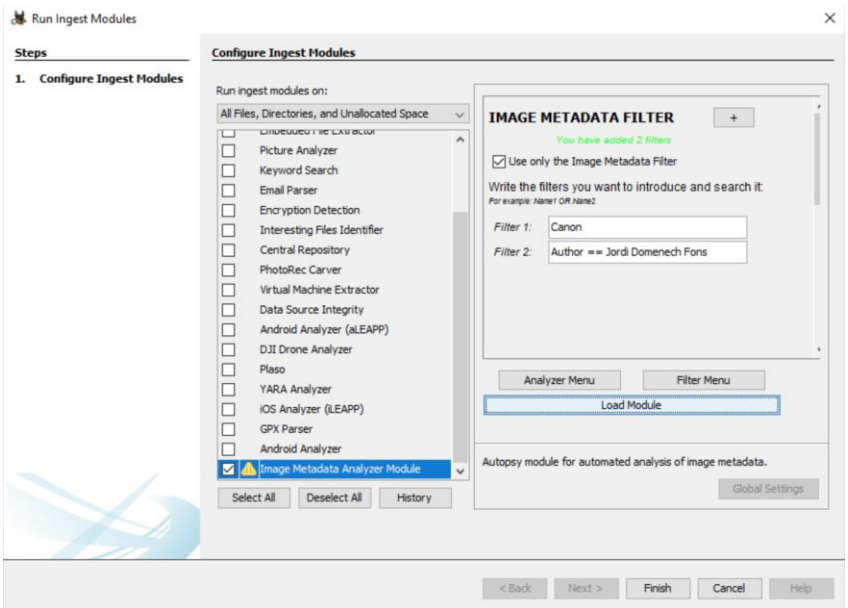


Figure 58. Filter Menu GUI (II)

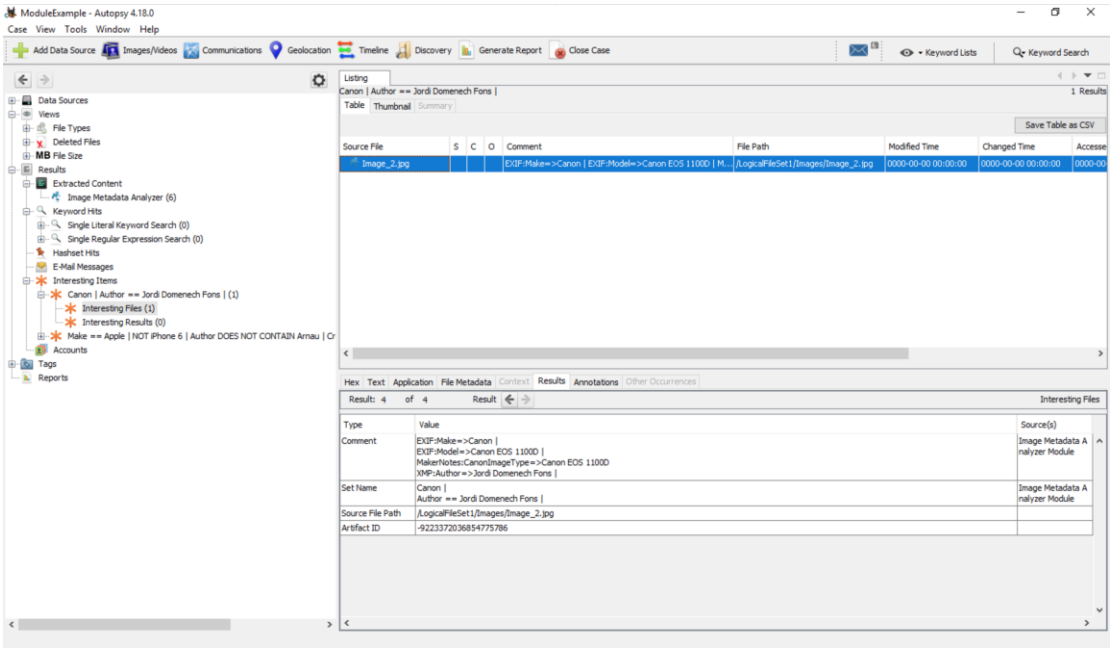


Figure 59. Filter Mode output in Autopsy (II)

Appendix X: ImageAnalyzerGUI.py

```
1.  #!/usr/bin/env python3
2.  # -*- coding: utf-8 -*-
3.  """
4.  Created on Tue Mar 30 09:16:46 2021
5.
6.  @author: JORDI
7.  """
8.
9.  # Sample module in the public domain. Feel free to use this as a template
10. # for your modules (and you can remove this header and take complete credit
11. # and liability)
12. #
13. # Contact: Brian Carrier [carrier <at> sleuthkit [dot] org]
14. #
15. # This is free and unencumbered software released into the public domain.
16. #
17. # Anyone is free to copy, modify, publish, use, compile, sell, or
18. # distribute this software, either in source code form or as a compiled
19. # binary, for any purpose, commercial or non-commercial, and by any
20. # means.
21. #
22. # In jurisdictions that recognize copyright laws, the author or authors
23. # of this software dedicate any and all copyright interest in the
24. # software to the public domain. We make this dedication for the benefit
25. # of the public at large and to the detriment of our heirs and
26. # successors. We intend this dedication to be an overt act of
27. # relinquishment in perpetuity of all present and future rights to this
28. # software under copyright law.
29. #
30. # THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
31. # EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
32. # MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
33. # IN NO EVENT SHALL THE AUTHORS BE LIABLE FOR ANY CLAIM, DAMAGES OR
34. # OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
35. # ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
36. # OTHER DEALINGS IN THE SOFTWARE.
37.
38.
39. # Ingest module for Autopsy with GUI
40. #
41. # Difference between other modules in this folder is that it has a GUI
42. # for user options.  This is not needed for very basic modules. If you
43. # don't need a configuration UI, start with the other sample module.
44. #
45. # Search for TODO for the things that you need to change
46. # See http://sleuthkit.org/autopsy/docs/api-docs/4.6.0/index.html for documentation
47.
48.
49. import jarray
50. import inspect
51. from java.lang import System
52. from java.util.logging import Level
53. from java.awt import Panel, BorderLayout, EventQueue, GridLayout, GridBagLayout,
    GridBagConstraints, Font, Color, CardLayout, Component, FlowLayout, Dimension
54. from javax.swing import JFrame, JLabel, JButton, JTextField, JComboBox, JTextField,
    JProgressBar, JMenuBar, JMenuItem, JTabbedPane, JPasswordField, JCheckBox,
    SwingConstants, BoxLayout, JPanel, JScrollPane
55. from org.sleuthkit.autopsy.casemodule import Case
56. from org.sleuthkit.autopsy.casemodule.services import Services
57. from org.sleuthkit.autopsy.ingest import DataSourceIngestModule
58. from org.sleuthkit.autopsy.ingest import FileIngestModule
59. from org.sleuthkit.autopsy.ingest import GenericIngestModuleJobSettings
60. from org.sleuthkit.autopsy.ingest import IngestMessage
61. from org.sleuthkit.autopsy.ingest import IngestModule
```

```
62. from org.sleuthkit.autopsy.ingest import ModuleDataEvent
63. from org.sleuthkit.autopsy.ingest.IngestModule import IngestModuleException
64. from org.sleuthkit.autopsy.ingest import IngestModuleFactoryAdapter
65. from org.sleuthkit.autopsy.ingest import IngestModuleIngestJobSettings
66. from org.sleuthkit.autopsy.ingest import IngestModuleIngestJobSettingsPanel
67. from org.sleuthkit.autopsy.ingest import IngestServices
68. from org.sleuthkit.autopsy.ingest import IngestModuleGlobalSettingsPanel
69. from org.sleuthkit.datamodel import BlackboardArtifact
70. from org.sleuthkit.datamodel import BlackboardAttribute
71. from org.sleuthkit.datamodel import ReadContentInputStream
72. from org.sleuthkit.datamodel import TskData
73. from org.sleuthkit.datamodel import SleuthkitCase
74. from org.sleuthkit.datamodel import AbstractFile
75. from org.sleuthkit.autopsy.coreutils import Logger
76. from org.sleuthkit.autopsy.coreutils import PlatformUtil
77. from java.lang import IllegalArgumentException
78.
79. from neededLib.exiftool import ExifTool
80. from neededLib import ImageAnalyzerLib
81. import re
82.
83. PATH_WINDOWS = "C:\WINDOWS\exiftool.exe" # WINDOWS path to the exiftool executable
84. PATH_MACOS = "/usr/local/bin/exiftool" # MACOS path to the exiftool executable
85.
86. filterCounter = 0
87. imageFound = False
88.
89.
90. #==========================================================================================#
91. #                    -- MODULE'S BASIC CONFIGURATION RELATED CLASS --                      #
92. #==========================================================================================#
93.
94. class ImageMetadataFileIngestModuleWithUIFactory(IngestModuleFactoryAdapter):
95.
96.     def __init__(self):
97.         self.settings = None
98.
99.     # Give it a unique name.  Will be shown in module list, logs, etc.
100.      moduleName = "Image Metadata Analyzer Module"
101.
102.      def getModuleDisplayName(self):
103.          return self.moduleName
104.
105.      # Give it a description
106.      def getModuleDescription(self):
107.          return "Autopsy module for automated analysis of image metadata."
108.
109.      def getModuleVersionNumber(self):
110.          return "1.0"
111.
112.      def getDefaultIngestJobSettings(self):
113.          return GenericIngestModuleJobSettings()
114.
115.
116.      #============== start of Panel GUI basic configuration methods ===============#
117.
118.      # Keep enabled only if you need ingest job-specific settings UI
119.      def hasIngestJobSettingsPanel(self):
120.          return True
121.
122.      # Note that you must use GenericIngestModuleJobSettings instead of making a
    custom settings class.
123.      def getIngestJobSettingsPanel(self, settings):
124.          if not isinstance(settings, GenericIngestModuleJobSettings):
125.              raise IllegalArgumentException("Expected settings argument to be
    instanceof GenericIngestModuleJobSettings")
126.          self.settings = settings
127.          return ImageMetadataFileIngestModuleWithUISettingsPanel(self.settings)
```

```
128.
129.      def isFileIngestModuleFactory(self):
130.          return True
131.
132.      def createFileIngestModule(self, ingestOptions):
133.          return ImageMetadataFileIngestModuleWithUI(self.settings)
134.
135.      #================ end of Panel GUI basic configuration methods ===============#
136.
137.
138.
139.  #============================================================================#
140.  #                  -- MODULE'S MAIN FUNCTIONALITY RELATED CLASS --           #
141.  #============================================================================#
142.
143.  # File-level ingest module.  One gets created per thread.
144.  # Looks at the attributes of the passed in file.
145.  class ImageMetadataFileIngestModuleWithUI(FileIngestModule):
146.
147.      _logger =
    Logger.getLogger(ImageMetadataFileIngestModuleWithUIFactory.moduleName)
148.
149.      def log(self, level, msg):
150.          self._logger.logp(level, self.__class__.__name__, inspect.stack()[1][3],
    msg)
151.
152.      # Autopsy will pass in the settings from the UI panel
153.      def __init__(self, settings):
154.          self.context = None
155.          self.local_settings = settings
156.
157.      # Where any setup and configuration is done
158.      # Add any setup code that you need here.
159.      def startUp(self, context):
160.          self.filesAnalyzed = 0
161.
162.          # We start the logs that gives us information about if a box of the GUI has
    been set or not
163.          ImageAnalyzerLib.startUpLogs(self)
164.
165.          pass
166.
167.      # Where the analysis is done. Each file will be passed into here.
168.      # Add your analysis code in here.
169.      def process(self, file):
170.
171.         # Skip non-files
172.          if ((file.getType() == TskData.TSK_DB_FILES_TYPE_ENUM.UNALLOC_BLOCKS) or
173.              (file.getType() == TskData.TSK_DB_FILES_TYPE_ENUM.UNUSED_BLOCKS) or
174.              (file.isFile() == False)):
175.              return IngestModule.ProcessResult.OK
176.
177.
178.          # At the moment we will only analyze 5 different type of files:
179.              # - JPG (JPEG)
180.              # - PNG (does not support EXIF metadata)
181.              # - TIFF
182.              # - GIF (does not support IPTC metadata)
183.              # - HEIC (supports EXIF and XMP, but IPTC ??)
184.          # If the file corresponds with one of the 5 types and the box is selected
    to analyze that specific file, we start the analysis
185.          if (
186.              (file.getName().lower().endswith(".jpg") and
    self.local_settings.getSetting("jpg") == "true") or
187.              (file.getName().lower().endswith(".png") and
    self.local_settings.getSetting("png") == "true") or
188.              (file.getName().lower().endswith(".tiff") and
    self.local_settings.getSetting("tiff") == "true") or
```

```
189.                (file.getName().lower().endswith(".gif") and
    self.local_settings.getSetting("gif") == "true") or
190.                (file.getName().lower().endswith(".heic") and
    self.local_settings.getSetting("heic") == "true")
191.                ):
192.
193.                self.log(Level.INFO, "Found an image file with path: " +
    file.getLocalAbsPath())
194.                #self.log(Level.INFO, "Found a JPG file with path: " +
    file.getUniquePath())
195.
196.                self.log(Level.INFO, "Platform used: " + PlatformUtil.getOSName())
197.
198.
199.                # Check the platform the user is using
200.                if "Mac" in PlatformUtil.getOSName():
201.                    PATH = PATH_MACOS
202.
203.                    # Metadata Error Test
204.                    #PATH = PATH_WINDOWS
205.
206.                elif "Windows" in PlatformUtil.getOSName():
207.                    PATH = PATH_WINDOWS
208.
209.
210.                # Analyze the image metadata
211.                # If there is an error while analyzing the metadata, we will not run
    the module
212.                try:
213.
214.                    with ExifTool(PATH) as et:
215.                        # metadata is a dictionary (key:value pair)
216.                        # key => metadata type (tag)
217.                        # value => metadata value (value of the tag)
218.                        metadata = et.get_metadata(file.getLocalAbsPath())
219.
220.                        #metadata = et.get_metadata(file.getUniquePath())
221.
222.                except:
223.
224.                    ImageAnalyzerLib.postInformationForTheUser(self,
    ImageMetadataFileIngestModuleWithUIFactory.moduleName,
225.                                                               Level.SEVERE,
    IngestMessage.MessageType.ERROR,
226.                                                               "Error reading metadata
    of current file " + file.getName())
227.
228.                    return IngestModule.ProcessResult.ERROR
229.
230.
231.                # A list with the tags of all the metadata found in the image
232.                imageMetadataTagList = list(metadata)
233.
234.                # A list with the values of all the metadata found in the image
235.                imageMetadataValueList = list(metadata.values())
236.
237.
238.                self.filesAnalyzed += 1
239.
240.                self.log(Level.INFO, "Files analyzed: " + str(self.filesAnalyzed))
241.
242.
243.                #================== start of IMAGE METADATA ANALYSIS =================#
244.
245.                # Use blackboard class to index blackboard artifacts for keyword search
246.                blackboard = Case.getCurrentCase().getServices().getBlackboard()
247.
```

```
248.             # If the Filter Mode checkbox is selected we will not add the analyzed
    images with its metadata in the artifact we have in the blackboard
249.             # Because if we only wanted to execute filters, we would have repited
    images (in the Image Metadata Analyzer artifact) every time we execute the module
250.             if (not self.local_settings.getSetting("filter") == "true"):
251.
252.                 # Creating a custom Artifact
253.                 artId = blackboard.getOrAddArtifactType("TSK_IMAGE_METADATA",
    "Image Metadata Analyzer") #Blackboard Artifact Type
254.                 artifact = file.newArtifact(artId.getTypeID())
255.
256.
257.                 for m in range(0, len(metadata)):
258.
259.                     # Check the GUI box "EXIF"
260.                     if (self.local_settings.getSetting("exif") == "true") and
    (imageMetadataTagList[m].startswith("EXIF")):
261.
262.                         # Check the value of the metadata: if it is not a string,
    convert it to string for a correct printing
263.                         # metadataValue is the str of each metadata value
264.                         metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
265.
266.                         # Add Attribute to the Artifact
267.                         try:
268.
269.                             ImageAnalyzerLib.addNewMetadataAttribute(blackboard,
    artifact, artId, ImageMetadataFileIngestModuleWithUIFactory.moduleName,
270.
    imageMetadataTagList[m], metadataValue)
271.                         except:
272.
273.                             ImageAnalyzerLib.postInformationForTheUser(self,
    ImageMetadataFileIngestModuleWithUIFactory.moduleName,
274.
    Level.WARNING, IngestMessage.MessageType.ERROR,
275.                                                                    "Error
    adding EXIF Attribute " + imageMetadataTagList[m] + " to the Artifact!")
276.
277.                     # Check the GUI box "IPTC"
278.                     if (self.local_settings.getSetting("iptc") == "true") and
    (imageMetadataTagList[m].startswith("IPTC")):
279.
280.                         # Check the value of the metadata: if it is not a string,
    convert it to string for a correct printing
281.                         # metadataValue is the str of each metadata value
282.                         metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
283.
284.                          # Add Attribute to the Artifact
285.                         try:
286.
287.                             ImageAnalyzerLib.addNewMetadataAttribute(blackboard,
    artifact, artId, ImageMetadataFileIngestModuleWithUIFactory.moduleName,
288.
    imageMetadataTagList[m], metadataValue)
289.                         except:
290.
291.                             ImageAnalyzerLib.postInformationForTheUser(self,
    ImageMetadataFileIngestModuleWithUIFactory.moduleName,
292.
    Level.WARNING, IngestMessage.MessageType.ERROR,
293.                                                                    "Error
    adding IPTC Attribute " + imageMetadataTagList[m] + " to the Artifact!")
294.
295.                     # Check the GUI box "XMP"
```

```
296.                     if (self.local_settings.getSetting("xmp") == "true") and
    (imageMetadataTagList[m].startswith("XMP")):
297.
298.                         # Check the value of the metadata: if it is not a string,
    convert it to string for a correct printing
299.                         # metadataValue is the str of each metadata value
300.                         metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
301.
302.                          # Add Attribute to the Artifact
303.                         try:
304.
305.                             ImageAnalyzerLib.addNewMetadataAttribute(blackboard,
    artifact, artId, ImageMetadataFileIngestModuleWithUIFactory.moduleName,
306.
    imageMetadataTagList[m], metadataValue)
307.                         except:
308.
309.                             ImageAnalyzerLib.postInformationForTheUser(self,
    ImageMetadataFileIngestModuleWithUIFactory.moduleName,
310.
    Level.WARNING, IngestMessage.MessageType.ERROR,
311.                                                                         "Error
    adding XMP Attribute " + imageMetadataTagList[m] + " to the Artifact!")
312.
313.                     # Add other metadata information found such as "File:",
    "Composite:", etc.
314.                     if self.local_settings.getSetting("other") == "true" and not
    (imageMetadataTagList[m].startswith("EXIF") or
315.
    imageMetadataTagList[m].startswith("IPTC") or
316.
    imageMetadataTagList[m].startswith("XMP")):
317.
318.                         # Check the value of the metadata: if it is not a string,
    convert it to string for a correct printing
319.                         # metadataValue is the str of each metadata value
320.                         metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
321.
322.                         # Add Attribute to the Artifact
323.                         try:
324.
325.                             ImageAnalyzerLib.addNewMetadataAttribute(blackboard,
    artifact, artId, ImageMetadataFileIngestModuleWithUIFactory.moduleName,
326.
    imageMetadataTagList[m], metadataValue)
327.                         except:
328.
329.                             ImageAnalyzerLib.postInformationForTheUser(self,
    ImageMetadataFileIngestModuleWithUIFactory.moduleName,
330.
    Level.WARNING, IngestMessage.MessageType.ERROR,
331.                                                                         "Error
    adding Other Attribute " + imageMetadataTagList[m] + " to the Artifact!")
332.
333.
334.             #================== end of IMAGE METADATA ANALYSIS ==================#
335.
336.
337.             #================ start of IMAGE METADATA FILTERING ==================#
338.
339.             # Different levels:
340.             #   - 1st level => basic one condition filter
341.             #   - 2nd level => "AND" filter ("" + AND + "")
342.             #   - 3rd level => "OR" filter ("" + OR + "")
343.             #   - 4th level => "NOT" filter (NOT + "")
```

```
344.                #   - 5th level => filter by specific metadata attribute (Author ==
    Jordi)
345.                #   - 6th level => filter by specific metadata attribute (Author !=
    Jordi)
346.                #   - 7th level => filter by specific metadata attribute (Author
    CONTAIN Jordi)
347.                #   - 8th level => filter by specific metadata attribute (Author DOES
    NOT CONTAIN Jordi)
348.                # TODO: - 9th level => "AND", "OR", "NOT" mixed filter
349.
350.                validation = 0
351.
352.                pattern_1 = "[a-zA-Z0-9]"
353.
354.                pattern_2 = "[a-zA-Z0-9]\s(AND)\s[a-zA-Z0-9]"
355.
356.                pattern_3 = "[a-zA-Z0-9]\s(OR)\s[a-zA-Z0-9]"
357.
358.                pattern_4 = "(NOT)\s[a-zA-Z0-9]"
359.
360.                pattern_5 = "[a-zA-Z0-9]\s(==)\s[a-zA-Z0-9]"
361.
362.                pattern_6 = "[a-zA-Z0-9]\s(!=)\s[a-zA-Z0-9]"
363.
364.                pattern_7 = "[a-zA-Z0-9]\s(CONTAINS)\s[a-zA-Z0-9]"
365.
366.                pattern_8 = "[a-zA-Z0-9]\s(DOES\sNOT\sCONTAIN)\s[a-zA-Z0-9]"
367.
368.                self.filterFound = True
369.
370.                self.fileHitName_info = ""
371.                self.fileHitComment_info = ""
372.
373.                global imageFound
374.
375.
376.                for m in range(filterCounter):
377.                    self.wordToSearch = self.local_settings.getSetting("word_search_" +
    str(m+1))
378.                    self.log(Level.INFO, "word_search_" + str(m+1) + ": " +
    self.wordToSearch)
379.
380.
381.                interestingFileHitArtifact =
    file.newArtifact(BlackboardArtifact.ARTIFACT_TYPE.TSK_INTERESTING_FILE_HIT)
382.
383.                for m in range(filterCounter):
384.
385.                    self.wordToSearch = self.local_settings.getSetting("word_search_" +
    str(m+1))
386.
387.                    self.fileHitName_info += self.wordToSearch + " | \n"
388.
389.
390.                    if self.filterFound is True:
391.
392.                        # 5th LEVEL: "=="
393.                        if (re.search(pattern_5, self.wordToSearch)):
394.                            validation = 0
395.
396.                            # output[0] = tag of the metadata
397.                            # output[1] = corresponding value of the metadata
398.                            output = re.split("\s==\s", self.wordToSearch)
399.
400.                            for m in range(0, len(metadata)):
401.
402.                                metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
```

```
403.
404.                                 metadataTagPairs = re.split(":",
    imageMetadataTagList[m])
405.
406.                                 # There is usually an error with the first metadata
    because does not cotain the ':',
407.                                 # and for that reason there is only 1 name in the list
    and not 2
408.                                 try:
409.                                     metadataTag = metadataTagPairs[1]
410.                                 except:
411.                                     metadataTag = metadataTagPairs[0]
412.
413.                                 # If the tag is the same as introduced and it is the
    same name as the metadata value introduced, the filter is found
414.                                 if metadataTag.lower() == output[0].lower() and
    metadataValue.lower() == output[1].lower():
415.                                     validation += 1
416.                                     metadataEntireTag = imageMetadataTagList[m]
417.                                     break
418.
419.                             # If it is true, the "==" condition has been accomplished
420.                             if validation == 1:
421.
422.                                 metadataTagValue = metadataEntireTag + "=>" +
    metadataValue
423.
424.                                 self.fileHitComment_info += "\n" + metadataTagValue + "
    | "
425.
426.                             else:
427.                                 self.filterFound = False
428.                                 self.log(Level.INFO, '"' + self.wordToSearch + '"
    filter not found in the metadata of ' + '"' + file.getName() + '"')
429.
430.                         # 6th LEVEL: "!="
431.                         elif (re.search(pattern_6, self.wordToSearch)):
432.                             validation = 0
433.
434.                             # output[0] = tag of the metadata
435.                             # output[1] = corresponding value of the metadata
436.                             output = re.split("\s!=\s", self.wordToSearch)
437.
438.                             for m in range(0, len(metadata)):
439.
440.                                 metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
441.
442.                                 metadataTagPairs = re.split(":",
    imageMetadataTagList[m])
443.
444.                                 # There is usually an error with the first metadata
    because does not cotain the ':',
445.                                 # and for that reason there is only 1 name in the list
    and not 2
446.                                 try:
447.                                     metadataTag = metadataTagPairs[1]
448.                                 except:
449.                                     metadataTag = metadataTagPairs[0]
450.
451.                                 # If the tag is the same as introduced and it is NOT
    the same name as the metadata value introduced, the filter is found
452.                                 if metadataTag.lower() == output[0].lower() and
    metadataValue.lower() != output[1].lower():
453.                                     validation += 1
454.                                     metadataEntireTag = imageMetadataTagList[m]
455.                                     break
456.
```

```
457.                             # If it is true, the "!=" condition has been accomplished
458.                             if validation == 1:
459.
460.                                 metadataTagValue = metadataEntireTag + "=>" +
    metadataValue
461.
462.                                 self.fileHitComment_info += "\n" + metadataTagValue + "
    | "
463.
464.                             else:
465.                                 self.filterFound = False
466.                                 self.log(Level.INFO, '"' + self.wordToSearch + '"
    filter not found in the metadata of ' + '"' + file.getName() + '"')
467.
468.                         # 7th LEVEL: "CONTAINS"
469.                         elif (re.search(pattern_7, self.wordToSearch)):
470.                             validation = 0
471.
472.                             # output[0] = tag of the metadata
473.                             # output[1] = corresponding value of the metadata
474.                             output = re.split("\sCONTAINS\s", self.wordToSearch)
475.
476.                             for m in range(0, len(metadata)):
477.
478.                                 metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
479.
480.                                 metadataTagPairs = re.split(":",
    imageMetadataTagList[m])
481.
482.                                 # There is usually an error with the first metadata
    because does not cotain the ':',
483.                                 # and for that reason there is only 1 name in the list
    and not 2
484.                                 try:
485.                                     metadataTag = metadataTagPairs[1]
486.                                 except:
487.                                     metadataTag = metadataTagPairs[0]
488.
489.                                 # If the tag is the same as introduced and it CONTAINS
    the metadata value introduced, the filter is found
490.                                 # Here is the difference between '==' and 'CONTAINS' =>
    output[1] in metadataValue
491.                                 if (metadataTag.lower() == output[0].lower()) and
    (output[1].lower() in metadataValue.lower()):
492.                                     validation += 1
493.                                     metadataEntireTag = imageMetadataTagList[m]
494.                                     break
495.
496.                             # If it is true, the "CONTAINS" condition has been
    accomplished
497.                             if validation == 1:
498.
499.                                 metadataTagValue = metadataEntireTag + "=>" +
    metadataValue
500.
501.                                 self.fileHitComment_info += "\n" + metadataTagValue + "
    | "
502.
503.                             else:
504.                                 self.filterFound = False
505.                                 self.log(Level.INFO, '"' + self.wordToSearch + '"
    filter not found in the metadata of ' + '"' + file.getName() + '"')
506.
507.                         # 8th LEVEL: "DOES NOT CONTAIN"
508.                         elif (re.search(pattern_8, self.wordToSearch)):
509.                             validation = 0
510.
```

```
511.                            # output[0] = tag of the metadata
512.                            # output[1] = corresponding value of the metadata
513.                            output = re.split("\sDOES\sNOT\sCONTAIN\s",
    self.wordToSearch)
514.
515.                            for m in range(0, len(metadata)):
516.
517.                                metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
518.
519.                                metadataTagPairs = re.split(":",
    imageMetadataTagList[m])
520.
521.                                # There is usually an error with the first metadata
    because does not cotain the ':',
522.                                # and for that reason there is only 1 name in the list
    and not 2
523.                                try:
524.                                    metadataTag = metadataTagPairs[1]
525.                                except:
526.                                    metadataTag = metadataTagPairs[0]
527.
528.                                # If the tag is the same as introduced and it DOES NOT
    CONTAIN the metadata value introduced, the filter is found
529.                                # Here is the difference between '!=' and 'DOES NOT
    CONTAIN' => output[1] not in metadataValue
530.                                if (metadataTag.lower() == output[0].lower()) and
    (output[1].lower() not in metadataValue.lower()):
531.                                    validation += 1
532.                                    metadataEntireTag = imageMetadataTagList[m]
533.                                    break
534.
535.                            # If it is true, the "DOES NOT CONTAIN" condition has been
    accomplished
536.                            if validation == 1:
537.
538.                                metadataTagValue = metadataEntireTag + "=>" +
    metadataValue
539.
540.                                self.fileHitComment_info += "\n" + metadataTagValue + "
    | "
541.
542.                            else:
543.                                self.filterFound = False
544.                                self.log(Level.INFO, '"' + self.wordToSearch + '"
    filter not found in the metadata of ' + '"' + file.getName() + '"')
545.
546.                        # 2nd LEVEL: "AND".
547.                        #   - We can include as many "AND" as we want: Canon AND iPhone
    AND Jordi AND ...
548.                        elif (re.search(pattern_2, self.wordToSearch)):
549.                            metadataTagValueList = []
550.                            validation = 0
551.                            output = re.split("\sAND\s", self.wordToSearch) # Output is
    a list containing the words to search
552.
553.                            for x in output:
554.
555.                                for m in range(0, len(metadata)):
556.
557.                                    # Check the value of the metadata: if it is not a
    string, convert it to string for a correct evaluation
558.                                    # metadataValue is the str of each metadata value
559.                                    metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
560.
561.                                    if x.lower() in metadataValue.lower():
562.                                        validation += 1
```

```
563.                                     break
564.
565.                             # If it is true, the "AND" condition has been accomplished
566.                             if validation == len(output):
567.
568.                                 # Adding the comments, that is the tags where the
    metadata values have been found
569.                                 for x in output:
570.                                     for m in range(0, len(metadata)):
571.
572.                                         metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
573.
574.                                         if x.lower() in metadataValue.lower():
575.                                             metadataTagValue = imageMetadataTagList[m]
    + "=>" + metadataValue
576.
    metadataTagValueList.append(metadataTagValue)
577.
578.                                 self.fileHitComment_info += "\n" + ' |
    \n'.join(metadataTagValueList)
579.
580.                             else:
581.                                 self.filterFound = False
582.                                 self.log(Level.INFO, '"' + self.wordToSearch + '"
    filter not found in the metadata of ' + '"' + file.getName() + '"')
583.
584.                         # 3rd LEVEL: "OR".
585.                         #   - We can include as many "OR" as we want: Canon OR iPhone
    OR Jordi OR ...
586.                         elif (re.search(pattern_3, self.wordToSearch)):
587.                             metadataTagValueList = []
588.                             validation = 0
589.                             output = re.split("\sOR\s", self.wordToSearch) # Output is
    a list containing the words to search
590.
591.                             for x in output:
592.
593.                                 for m in range(0, len(metadata)):
594.
595.                                     # Check the value of the metadata: if it is not a
    string, convert it to string for a correct evaluation
596.                                     # metadataValue is the str of each metadata value
597.                                     metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
598.
599.                                     if x.lower() in metadataValue.lower():
600.                                         validation += 1
601.
602.                             # If it is true, the "OR" condition has been accomplished
603.                             if validation > 0:
604.
605.                                 # Adding the comments, that is the tags where the
    metadata values have been found
606.                                 for x in output:
607.                                     for m in range(0, len(metadata)):
608.
609.                                         metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
610.
611.                                         if x.lower() in metadataValue.lower():
612.                                             metadataTagValue = imageMetadataTagList[m]
    + "=>" + metadataValue
613.
    metadataTagValueList.append(metadataTagValue)
614.
615.                                 self.fileHitComment_info += "\n" + ' |
    \n'.join(metadataTagValueList)
```

```
616.
617.                             else:
618.                                 self.filterFound = False
619.                                 self.log(Level.INFO, '"' + self.wordToSearch + '"
    filter not found in the metadata of ' + '"' + file.getName() + '"')
620.
621.                         # 4th LEVEL: "NOT".
622.                         #   - We can only include one "not"
623.                         elif (re.search(pattern_4, self.wordToSearch)):
624.                             validation = 0
625.                             output = re.sub("NOT\s", "", self.wordToSearch) # Output is
    a list containing the words to search
626.
627.                             for m in range(0, len(metadata)):
628.
629.                                 # Check the value of the metadata: if it is not a
    string, convert it to string for a correct evaluation
630.                                 # metadataValue is the str of each metadata value
631.                                 metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
632.
633.                                 if output.lower() in metadataValue.lower():
634.                                     validation += 1
635.
636.                             # If it is true, the "NOT" condition has been accomplished
637.                             if validation == 0:
638.
639.                                 self.fileHitComment_info += "\n" + "The metadata of the
    image does NOT contain the word searched: " + output + " | "
640.
641.                             else:
642.                                 self.filterFound = False
643.                                 self.log(Level.INFO, '"' + self.wordToSearch + '"
    filter not found in the metadata of ' + '"' + file.getName() + '"')
644.
645.                         # 1st LEVEL: One word.
646.                         #   - We can only include one word to search
647.                         elif (re.search(pattern_1, self.wordToSearch)):
648.                             validation = 0
649.                             metadataTagValueList = []
650.                             output = self.wordToSearch
651.
652.                             for m in range(0, len(metadata)):
653.
654.                                 # Check the value of the metadata: if it is not a
    string, convert it to string for a correct evaluation
655.                                 # metadataValue is the str of each metadata value
656.                                 metadataValue =
    ImageAnalyzerLib.metadataToString(imageMetadataValueList[m])
657.
658.                                 if output.lower() in metadataValue.lower():
659.                                     metadataTagValue = imageMetadataTagList[m] + "=>" +
    metadataValue
660.                                     metadataTagValueList.append(metadataTagValue)
661.                                     validation += 1
662.
663.                             # If it is true, the 1st level condition has been
    accomplished
664.                             # Adding the comments, that is the tags where the metadata
    values have been found
665.                             if validation > 0:
666.
667.                                 self.fileHitComment_info += "\n" + ' |
    \n'.join(metadataTagValueList)
668.
669.                             else:
670.                                 self.filterFound = False
```

```
671.                                self.log(Level.INFO, '"' + self.wordToSearch + '"
    filter not found in the metadata of ' + '"' + file.getName() + '"')
672.
673.                        elif (self.wordToSearch == ""):
674.
675.                            self.filterFound = False
676.                            self.log(Level.INFO, "You have not entered any filter")
677.
678.                        else:
679.
680.                            self.filterFound = False
681.                            ImageAnalyzerLib.postInformationForTheUser(self,
    ImageMetadataFileIngestModuleWithUIFactory.moduleName,
682.                                                                       Level.WARNING,
    IngestMessage.MessageType.ERROR, "Incorrect filter's input!")
683.
684.                # If we find an image that contains the filters, we add the Interesting
    File Hit Attribute
685.                if self.filterFound:
686.
687.                    # Dictionary (key:value) with the attribute name (key) and its type
    (value)
688.                    # { AttributeName: AttributeType }
689.                    attributesDict = {
690.                        self.fileHitName_info:
    BlackboardAttribute.ATTRIBUTE_TYPE.TSK_SET_NAME,
691.                        self.fileHitComment_info:
    BlackboardAttribute.ATTRIBUTE_TYPE.TSK_COMMENT
692.                        }
693.
694.
    ImageAnalyzerLib.addInterestingFileHitAttributes(interestingFileHitArtifact,
695.
    ImageMetadataFileIngestModuleWithUIFactory.moduleName,
696.                                                                        attributesDict)
697.
698.                    imageFound = True
699.
700.
701.                self.log(Level.INFO, "Image Found: " + str(imageFound))
702.
703.                #================== end of IMAGE METADATA FILTERING ==================#
704.
705.
706.            return IngestModule.ProcessResult.OK
707.
708.
709.
710.        # Where any shutdown code is run and resources are freed.
711.        def shutDown(self):
712.            # As a final part, we'll send a message to the ingest inbox with the number
    of files found (in this thread)
713.            message = IngestMessage.createMessage(
714.                IngestMessage.MessageType.INFO,
    ImageMetadataFileIngestModuleWithUIFactory.moduleName,
715.                    str(self.filesAnalyzed) + " files analyzed")
716.            ingestServices = IngestServices.getInstance().postMessage(message)
717.
718.            global imageFound
719.
720.            # If we have not found any image that contain the filters introduced, we
    send a message to the user
721.            if (not imageFound) and (self.fileHitName_info != " | \n"):
722.                self.log(Level.WARNING, 'Filter "'  + self.fileHitName_info + '" NOT
    FOUND !')
723.                ImageAnalyzerLib.postInformationForTheUser(self,
    ImageMetadataFileIngestModuleWithUIFactory.moduleName,
```

```
724.                                                    Level.WARNING,
    IngestMessage.MessageType.WARNING, 'Filter "'  + self.fileHitName_info + '" NOT FOUND
    !')
725.
726.
727.
728.
729.
730.  #============================================================================#
731.  #                          -- PANEL GUI RELATED CLASSES --                   #
732.  #============================================================================#
733.
734.  #========================= start of GENERIC GUI CLASS ========================#
735.
736.  # UI that is shown to user for each ingest job so they can configure the job.
737.  class
    ImageMetadataFileIngestModuleWithUISettingsPanel(IngestModuleIngestJobSettingsPanel):
738.      # Note, we can't use a self.settings instance variable.
739.      # Rather, self.local_settings is used.
740.      # https://wiki.python.org/jython/UserGuide#javabean-properties
741.      # Jython Introspector generates a property - 'settings' on the basis
742.      # of getSettings() defined in this class. Since only getter function
743.      # is present, it creates a read-only 'settings' property. This auto-
744.      # generated read-only property overshadows the instance-variable -
745.      # 'settings'
746.
747.      # We get passed in a previous version of the settings so that we can
748.      # prepopulate the UI
749.      def __init__(self, settings):
750.          self.local_settings = settings
751.          self.initComponents()
752.          #self.customizeComponents()
753.
754.
755.      def initComponents(self):
756.
757.          self.setLayout(None)
758.
759.          self.card = CardLayout()
760.          self.mainPanel = JPanel(self.card)
761.          self.mainPanel.setBounds(0,0,350,265)
762.
763.          metadataAnalyzerGUI = JMetadataAnalyzerGUI(self.local_settings, self)
764.          metadataFilterGUI = JMetadataFilterGUI(self.local_settings, self)
765.
766.          metadataAnalyzerGUIPanel = metadataAnalyzerGUI.getPanel()
767.          metadataFilterGUIPanel = metadataFilterGUI.getPanel()
768.
769.          self.mainPanel.add(metadataAnalyzerGUIPanel, "1")
770.          self.mainPanel.add(metadataFilterGUIPanel, "2")
771.
772.          self.add(self.mainPanel)
773.
774.          button1 = JButton("Analyzer Menu", actionPerformed = self.testEvent1)
775.          button1.setHorizontalAlignment(SwingConstants.CENTER)
776.          button1.setBounds(15, 275, 125, 22)
777.          self.add(button1)
778.
779.          button2 = JButton("Filter Menu", actionPerformed = self.testEvent2)
780.          button2.setHorizontalAlignment(SwingConstants.CENTER)
781.          button2.setBounds(160, 275, 125, 22)
782.          self.add(button2)
783.
784.          button3 = JButton("Load Module", actionPerformed =
    metadataFilterGUI.wordCheckBoxEvent)
785.          button3.setHorizontalAlignment(SwingConstants.CENTER)
786.          button3.setBounds(0, 300, 300, 22)
787.          self.add(button3)
```

```
788.
789.
790.      def customizeComponents(self):
791.          pass
792.
793.
794.      def testEvent1(self, event):
795.          self.card.show(self.mainPanel, "1")
796.
797.
798.      def testEvent2(self, event):
799.          self.card.show(self.mainPanel, "2")
800.
801.      # Return the settings used
802.      def getSettings(self):
803.          return self.local_settings
804.
805.  #========================== end of GENERIC GUI CLASS ==========================#
806.
807.
808.  #=================== start of METADATA ANALYZER GUI CLASS ====================#
809.
810.  class JMetadataAnalyzerGUI:
811.
812.      def __init__(self, settings, frame):
813.
814.          self.local_settings = settings
815.          self.frame = frame
816.          self.initComponents()
817.          self.customizeComponents()
818.
819.
820.      def initComponents(self):
821.
822.          self.panel = JPanel()
823.          self.panel.setLayout(None)
824.          #self.panel.setBackground(Color.gray)
825.
826.          title1 = JLabel("IMAGE METADATA ANALYZER")
827.          title1.setHorizontalAlignment(SwingConstants.LEFT)
828.          title1.setFont(Font("Tahoma", Font.BOLD, 14))
829.          title1.setBounds(5, 10, 300, 20)
830.          self.panel.add(title1)
831.
832.          label1 = JLabel("Select the type of Metadata you want to analyze:")
833.          label1.setHorizontalAlignment(SwingConstants.LEFT)
834.          label1.setFont(Font("Default", Font.PLAIN, 11))
835.          label1.setBounds(5, 40, 300, 20)
836.          self.panel.add(label1)
837.
838.          # EXIF checkbox
839.          self.exif_checkbox = JCheckBox("EXIF Metadata",
    actionPerformed=self.exifCheckBoxEvent)
840.          self.exif_checkbox.setBounds(10, 60, 110, 20)
841.          self.panel.add(self.exif_checkbox)
842.
843.          # IPTC checkbox
844.          self.iptc_checkbox = JCheckBox("IPTC Metadata",
    actionPerformed=self.iptcCheckBoxEvent)
845.          self.iptc_checkbox.setBounds(130, 60, 110, 20)
846.          self.panel.add(self.iptc_checkbox)
847.
848.          # XMP checkbox
849.          self.xmp_checkbox = JCheckBox("XMP Metadata",
    actionPerformed=self.xmpCheckBoxEvent)
850.          self.xmp_checkbox.setBounds(10, 80, 110, 20)
851.          self.panel.add(self.xmp_checkbox)
852.
```

```
853.          # Other checkbox
854.          self.other_checkbox = JCheckBox("Other",
    actionPerformed=self.otherCheckBoxEvent)
855.          self.other_checkbox.setBounds(130, 80, 110, 20)
856.          self.panel.add(self.other_checkbox)
857.
858.
859.          label2 = JLabel("Select the type of image file you want to analyze:")
860.          label2.setHorizontalAlignment(SwingConstants.LEFT)
861.          label2.setFont(Font("Default", Font.PLAIN, 11))
862.          label2.setBounds(5, 110, 300, 20)
863.          self.panel.add(label2)
864.
865.          # JPG checkbox
866.          self.jpg_checkbox = JCheckBox("JPG (JPEG)",
    actionPerformed=self.jpgCheckBoxEvent)
867.          self.jpg_checkbox.setBounds(10, 130, 80, 20)
868.          self.panel.add(self.jpg_checkbox)
869.
870.          # PNG checkbox
871.          self.png_checkbox = JCheckBox("PNG", actionPerformed=self.pngCheckBoxEvent)
872.          self.png_checkbox.setBounds(105, 130, 80, 20)
873.          self.panel.add(self.png_checkbox)
874.
875.          # TIFF checkbox
876.          self.tiff_checkbox = JCheckBox("TIFF",
    actionPerformed=self.tiffCheckBoxEvent)
877.          self.tiff_checkbox.setBounds(200, 130, 80, 20)
878.          self.panel.add(self.tiff_checkbox)
879.
880.          # GIF checkbox
881.          self.gif_checkbox = JCheckBox("GIF", actionPerformed=self.gifCheckBoxEvent)
882.          self.gif_checkbox.setBounds(10, 150, 80, 20)
883.          self.panel.add(self.gif_checkbox)
884.
885.          # HEIC checkbox
886.          self.heic_checkbox = JCheckBox("HEIC",
    actionPerformed=self.heicCheckBoxEvent)
887.          self.heic_checkbox.setBounds(105, 150, 80, 20)
888.          self.panel.add(self.heic_checkbox)
889.
890.          self.frame.add(self.panel)
891.
892.
893.      def customizeComponents(self):
894.
895.          # Mantain the GUI EXIF box selected if selected before
896.          self.exif_checkbox.setSelected(self.local_settings.getSetting("exif") ==
    "true")
897.
898.          # Mantain the GUI IPTC box selected if selected before
899.          self.iptc_checkbox.setSelected(self.local_settings.getSetting("iptc") ==
    "true")
900.
901.          # Mantain the GUI XMP box selected if selected before
902.          self.xmp_checkbox.setSelected(self.local_settings.getSetting("xmp") ==
    "true")
903.
904.          # Mantain the GUI Other box selected if selected before
905.          self.other_checkbox.setSelected(self.local_settings.getSetting("other") ==
    "true")
906.
907.          # Mantain the GUI JPG box selected if selected before
908.          self.jpg_checkbox.setSelected(self.local_settings.getSetting("jpg") ==
    "true")
909.
910.          # Mantain the GUI PNG box selected if selected before
```

```
911.            self.png_checkbox.setSelected(self.local_settings.getSetting("png") ==
    "true")
912.
913.            # Mantain the GUI TIFF box selected if selected before
914.            self.tiff_checkbox.setSelected(self.local_settings.getSetting("tiff") ==
    "true")
915.
916.            # Mantain the GUI GIF box selected if selected before
917.            self.gif_checkbox.setSelected(self.local_settings.getSetting("gif") ==
    "true")
918.
919.            # Mantain the GUI HEIC box selected if selected before
920.            self.heic_checkbox.setSelected(self.local_settings.getSetting("heic") ==
    "true")
921.
922.
923.        def exifCheckBoxEvent(self, event):
924.            if self.exif_checkbox.isSelected():
925.                self.local_settings.setSetting("exif", "true")
926.            else:
927.                self.local_settings.setSetting("exif", "false")
928.
929.
930.        def iptcCheckBoxEvent(self, event):
931.            if self.iptc_checkbox.isSelected():
932.                self.local_settings.setSetting("iptc", "true")
933.            else:
934.                self.local_settings.setSetting("iptc", "false")
935.
936.        def xmpCheckBoxEvent(self, event):
937.            if self.xmp_checkbox.isSelected():
938.                self.local_settings.setSetting("xmp", "true")
939.            else:
940.                self.local_settings.setSetting("xmp", "false")
941.
942.        def otherCheckBoxEvent(self, event):
943.            if self.other_checkbox.isSelected():
944.                self.local_settings.setSetting("other", "true")
945.            else:
946.                self.local_settings.setSetting("other", "false")
947.
948.        def jpgCheckBoxEvent(self, event):
949.            if self.jpg_checkbox.isSelected():
950.                self.local_settings.setSetting("jpg", "true")
951.            else:
952.                self.local_settings.setSetting("jpg", "false")
953.
954.        def pngCheckBoxEvent(self, event):
955.            if self.png_checkbox.isSelected():
956.                self.local_settings.setSetting("png", "true")
957.            else:
958.                self.local_settings.setSetting("png", "false")
959.
960.        def tiffCheckBoxEvent(self, event):
961.            if self.tiff_checkbox.isSelected():
962.                self.local_settings.setSetting("tiff", "true")
963.            else:
964.                self.local_settings.setSetting("tiff", "false")
965.
966.        def gifCheckBoxEvent(self, event):
967.            if self.gif_checkbox.isSelected():
968.                self.local_settings.setSetting("gif", "true")
969.            else:
970.                self.local_settings.setSetting("gif", "false")
971.
972.        def heicCheckBoxEvent(self, event):
973.            if self.heic_checkbox.isSelected():
974.                self.local_settings.setSetting("heic", "true")
```

```
975.          else:
976.              self.local_settings.setSetting("heic", "false")
977.
978.      def getPanel(self):
979.          return self.panel
980.
981.  #==================== end of METADATA ANALYZER GUI CLASS ====================#
982.
983.  #==================== start of METADATA FILTER GUI CLASS ====================#
984.
985.  class JMetadataFilterGUI:
986.
987.      word_search_list = []
988.
989.      def __init__(self, settings, frame):
990.
991.          self.local_settings = settings
992.
993.          self.frame = frame
994.          self.initComponents()
995.          self.customizeComponents()
996.
997.
998.
999.      def initComponents(self):
1000.
1001.         global filterCounter
1002.
1003.         self.panel = JPanel()
1004.         self.panel.setLayout(None)
1005.         #self.panel.setBackground(Color.yellow)
1006.         self.panel.setAlignmentX(Component.LEFT_ALIGNMENT)
1007.
1008.         filterCounter += 1
1009.         self.lastPosition = 120
1010.
1011.         title2 = JLabel("IMAGE METADATA FILTER")
1012.         title2.setHorizontalAlignment(SwingConstants.LEFT)
1013.         title2.setFont(Font("Tahoma", Font.BOLD, 14))
1014.         title2.setBounds(5, 10, 250, 20)
1015.         self.panel.add(title2)
1016.
1017.         button1 = JButton("+", actionPerformed=self.addEvent)
1018.         button1.setHorizontalAlignment(SwingConstants.CENTER)
1019.         button1.setBounds(230, 10, 43, 22)
1020.         self.panel.add(button1)
1021.
1022.         self.advertisment = JLabel("You have added " + str(filterCounter) + "
    filters")
1023.         self.advertisment.setForeground(Color.green)
1024.         self.advertisment.setHorizontalAlignment(SwingConstants.CENTER)
1025.         self.advertisment.setFont(Font("Default", Font.ITALIC, 10))
1026.         self.advertisment.setBounds(5, 33, 250, 20)
1027.         self.panel.add(self.advertisment)
1028.
1029.         self.filter_checkbox = JCheckBox("Use only the Image Metadata Filter",
    actionPerformed=self.filterChekBoxEvent)
1030.         self.filter_checkbox.setBounds(5, 55, 300, 20)
1031.         self.panel.add(self.filter_checkbox)
1032.
1033.
1034.         label3 = JLabel("Write the filters you want to introduce and search it:")
1035.         label3.setHorizontalAlignment(SwingConstants.LEFT)
1036.         label3.setFont(Font("Default", Font.PLAIN, 12))
1037.         label3.setBounds(5, 80, 300, 20)
1038.         self.panel.add(label3)
1039.
1040.
```

```
1041.         label3_1 = JLabel("For example: Name1 OR Name2")
1042.         label3_1.setHorizontalAlignment(SwingConstants.LEFT)
1043.         label3_1.setFont(Font("Default", Font.ITALIC, 8))
1044.         label3_1.setBounds(5, 95, 300, 20)
1045.         self.panel.add(label3_1)
1046.
1047.
1048.         self.lbl_1 = JLabel("Filter 1: ")
1049.         self.lbl_1.setBounds(10, 120, 50, 20)
1050.         self.lbl_1.setFont(Font("Default", Font.ITALIC, 11))
1051.         self.panel.add(self.lbl_1)
1052.
1053.         self.word_search = JTextField()
1054.         self.word_search.setBounds(65, 120, 200, 23)
1055.         self.panel.add(self.word_search)
1056.
1057.         # Creation of a list to access to all JTextFields of each filter
1058.         # the position of the list corresponds to the number of the filter - 1
1059.         self.word_search_list = [self.word_search]
1060.
1061.
1062.         self.panel.setPreferredSize(Dimension(300,650))
1063.         self.scroll = JScrollPane(self.panel,
    JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED, JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED)
1064.         self.scroll.setPreferredSize(Dimension(300,650))
1065.
1066.         #frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)
1067.         self.frame.add(self.scroll)
1068.         self.frame.setVisible(True)
1069.
1070.
1071.     def customizeComponents(self):
1072.         # Mantain the GUI "Use only the Image Metadata Filter" box selected if
    selected before
1073.         #self.filter_checkbox.setSelected(self.local_settings.getSetting("filter")
    == "true")
1074.         pass
1075.
1076.     def addEvent(self, event):
1077.         self.addNewFilter()
1078.
1079.     def wordCheckBoxEvent(self, event):
1080.         for m in range(len(self.word_search_list)):
1081.             self.local_settings.setSetting("word_search_" + str(m + 1),
    self.word_search_list[m].getText())
1082.
1083.     def filterChekBoxEvent(self, event):
1084.         if self.filter_checkbox.isSelected():
1085.             self.local_settings.setSetting("filter", "true")
1086.         else:
1087.             self.local_settings.setSetting("filter", "false")
1088.
1089.     def addNewFilter(self):
1090.
1091.         global filterCounter
1092.
1093.         filterCounter += 1
1094.         self.lastPosition += 25
1095.
1096.         if (filterCounter <= 20):
1097.             self.lbl_2 = JLabel("Filter " + str(filterCounter) + ":")
1098.             self.lbl_2.setBounds(10, self.lastPosition, 50, 20)
1099.             self.lbl_2.setFont(Font("Default", Font.ITALIC, 11))
1100.             self.panel.add(self.lbl_2)
1101.
1102.             self.word_search_2 = JTextField()
1103.             self.word_search_2.setBounds(65, self.lastPosition, 200, 23)
1104.             self.panel.add(self.word_search_2)
```

```
1105.
1106.             self.word_search_list.append(self.word_search_2)
1107.
1108.             self.advertisment.text = "You have added " + str(filterCounter) + "
    filters"
1109.
1110.         else:
1111.             self.advertisment.setForeground(Color.red)
1112.             self.advertisment.text = "You have added the maximum number of
    filters!"
1113.
1114.         # Refresh the panel, with this we don't have problems when adding new
    filters
1115.         # We click the button to add a new filter and the filter appears
    immediately
1116.         self.panel.revalidate();
1117.         self.panel.repaint();
1118.
1119.
1120.     def getPanel(self):
1121.         return self.scroll
1122.
1123.
1124. #====================== end of METADATA FILTER GUI CLASS ======================#
```

Appendix XI: ImageAnalyzerLib.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 30 20:10:07 2021

@author: JORDI
"""


from java.util.logging import Level

from org.sleuthkit.autopsy.ingest import IngestMessage
from org.sleuthkit.autopsy.ingest import IngestServices
from org.sleuthkit.datamodel import BlackboardAttribute
from org.sleuthkit.datamodel import BlackboardArtifact
from org.sleuthkit.autopsy.ingest import ModuleDataEvent


def startUpLogs(self):

    # Determine if user configured exif in UI
        if self.local_settings.getSetting("exif") == "true":
            self.log(Level.INFO, "exif is set")
        else:
            self.log(Level.INFO, "exif is not set")

        # Determine if user configured iptc in UI
        if self.local_settings.getSetting("iptc") == "true":
            self.log(Level.INFO, "iptc is set")
        else:
            self.log(Level.INFO, "iptc is not set")

        # Determine if user configured xmp in UI
        if self.local_settings.getSetting("xmp") == "true":
            self.log(Level.INFO, "xmp is set")
        else:
            self.log(Level.INFO, "xmp is not set")

        # Determine if user configured other in UI
        if self.local_settings.getSetting("other") == "true":
            self.log(Level.INFO, "other is set")
        else:
            self.log(Level.INFO, "other is not set")

        # Determine if user configured jpg in UI
        if self.local_settings.getSetting("jpg") == "true":
            self.log(Level.INFO, "jpg is set")
        else:
            self.log(Level.INFO, "jpg is not set")

        # Determine if user configured png in UI
        if self.local_settings.getSetting("png") == "true":
            self.log(Level.INFO, "png is set")
        else:
            self.log(Level.INFO, "png is not set")

        # Determine if user configured tiff in UI
        if self.local_settings.getSetting("tiff") == "true":
            self.log(Level.INFO, "tiff is set")
        else:
            self.log(Level.INFO, "tiff is not set")

        # Determine if user configured gif in UI
```

```
65.         if self.local_settings.getSetting("gif") == "true":
66.             self.log(Level.INFO, "gif is set")
67.         else:
68.             self.log(Level.INFO, "gif is not set")
69.
70.         # Determine if user configured heic in UI
71.         if self.local_settings.getSetting("heic") == "true":
72.             self.log(Level.INFO, "heic is set")
73.         else:
74.             self.log(Level.INFO, "heic is not set")
75.
76.         if self.local_settings.getSetting("filter") == "true":
77.             self.log(Level.INFO, "Use only the Image Metadata Filter is set")
78.         else:
79.             self.log(Level.INFO, "Use only the Image Metadata Filter is not set")
80.
81.
82.
83. # Write logs and post them on the User Interface to inform the user of any
    inconvenience or interesting information
84. def postInformationForTheUser(self, moduleName, levelType, messageType, sentence):
85.
86.     self.log(levelType, sentence)
87.
88.     message = IngestMessage.createMessage(messageType, moduleName, sentence)
89.     ingestServices = IngestServices.getInstance().postMessage(message)
90.
91.
92.
93. # Check the value of the metadata: if it is not a string, convert it to string for a
    correct printing
94. # metadataValue -> corresponds to each value/attribute of the metadata analyzed by
    the exiftool library
95. # Returns the string of each metadata attribute
96. def metadataToString(metadataValue):
97.
98.     if type(metadataValue) is not str:
99.         metadataValueStr = str(metadataValue)
100.      else:
101.          metadataValueStr = metadataValue
102.
103.      return metadataValueStr
104.
105.
106.
107.  # Add the new Attribute with the value of the metadata analyzed
108.  # metadataAttributeName -> corresponds to each name/title of the metadata analyzed
    by the exiftool library
109.  # metadataAttribute -> is the string of the value/attribute of each metadata
110.  def addNewMetadataAttribute(blackboard, artifact, artId, moduleName,
    metadataAttributeName, metadataAttribute):
111.
112.      attId = blackboard.getOrAddAttributeType("TSK_" + metadataAttributeName,
113.
    BlackboardAttribute.TSK_BLACKBOARD_ATTRIBUTE_VALUE_TYPE.STRING,
    metadataAttributeName)
114.      attribute = BlackboardAttribute(attId, moduleName, metadataAttribute)
115.
116.      # Adding the Attribute to the Artifact
117.      artifact.addAttribute(attribute)
118.      # Error Test
119.      #artifact.addAttribute(None)
120.
121.      blackboard.indexArtifact(artifact)
122.
123.      # Fires an event to notify the UI and others that there is a new artifact
124.      # So that the UI updates and refreshes with the new artifacts when the module
    is executed
```

```
125.      IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(moduleName,
    artId, None))
126.
127.
128.
129.  # Add a new Attribute in the File Interesting Hit Artifact
130.  # artifact -> File Interesting Hit Artifact
131.  # moduleName -> name of the module
132.  # attributeName -> name of the attribute we want to visualize on the blackboard
133.  def addInterestingFileHitAttribute(artifact, moduleName, attributeName):
134.
135.      attribute =
    BlackboardAttribute(BlackboardAttribute.ATTRIBUTE_TYPE.TSK_SET_NAME.getTypeID(),
    moduleName, attributeName)
136.
137.      artifact.addAttribute(attribute)
138.
139.      IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(moduleName,
    BlackboardArtifact.ARTIFACT_TYPE.TSK_INTERESTING_FILE_HIT, None))
140.
141.
142.
143.  # Add a new Attributes in the File Interesting Hit Artifact
144.  # artifact -> File Interesting Hit Artifact
145.  # moduleName -> name of the module
146.  # attributeName -> name of the attribute we want to visualize on the blackboard
147.  # comment -> string, comment you want to add
148.  # attributesDict -> dictrionary with the name of the attributes and its type (ex:
    {'iPhone' : BlackboardAttribute.ATTRIBUTE_TYPE.TSK_SET_NAME})
149.  def addInterestingFileHitAttributes(artifact, moduleName, attributesDict):
150.      attributesList = []
151.
152.      for m in range(0, len(attributesDict)):
153.          attribute =
    BlackboardAttribute(list(attributesDict.values())[m].getTypeID(), moduleName,
    list(attributesDict)[m])
154.          attributesList.append(attribute)
155.
156.
157.      artifact.addAttributes(attributesList)
158.
159.      IngestServices.getInstance().fireModuleDataEvent(ModuleDataEvent(moduleName,
160.
    BlackboardArtifact.ARTIFACT_TYPE.TSK_INTERESTING_FILE_HIT, None))
```