

Dokumentation des Clojure Projekts

Photosort 1.0

2014//THM//Clojure - Thomas Breitbach, André Wißner

Programmieren in Clojure

Prof. Dr. Burkhardt Renz

Inhaltsverzeichnis

1. Einleitung	3
1.1 Projektidee.....	3
1.2 Anforderungen an das Projekt.....	3
2. Exif - Ein Überblick	4
3. Genutzte Bibliotheken	5
3.1 Metadata-Extractor.....	5
Exif Struktur in Metadata-Extractor.....	6
3.2 Clargon.....	6
3.3 Seesaw.....	7
3.4 Clj-http.....	8
4. Exif-Data Wrapper in Clojure	9
5. Main.clj - Der Einstiegspunkt des Projektes	13
6. Projektkonfiguration	18
6.1 Leiningen und lokale JARs.....	18
7. Aufrufen des Projektes	20
7.1 Ausführung unter Windows 7/8.....	20
Möglichkeit 1 - Starten ohne Parameter.....	20
Möglichkeit 2 - Starten mit Angaben von Parameter.....	21
Möglichkeit 3 - Starten der Photosorter-easy.jar.....	21
7.2 Ausführung unter Mac OSX.....	21
Möglichkeit 1 - Starten mit Angaben von Parameter.....	21
Möglichkeit 2 - Starten der Photosorter-easy.jar.....	22
7.3 Ausführung unter Linux.....	22
Möglichkeit 1 - Starten ohne Parameter.....	22
Möglichkeit 2 - Starten der Photosorter-easy.jar.....	22
7.4 Ergebniss eines Durchlaufes unter Windows 8.....	22
8. Ausblick	24
9. Quellcode zum Projekt	25

1. Einleitung

Das folgende Kapitel geht kurz auf Projektidee sowie auf Anforderung der zu entwickelnde Software ein.

1.1 Projektidee

Die Idee dieses Projektes liegt darin, dass mit Hilfe eines Kommandozeilenprogramms digitale Fotos sortiert werden können. Folgender beispielhafter Anwendungsfall sei gegeben:

Man kommt von einem Urlaub zurück, in dem sehr viele Fotos geschossen wurden. Diese Sammlung von digitalen Fotografien stammt allerdings nicht von einer einzelnen Kamera. Somit kommt es mit sicherer Wahrscheinlichkeit vor, dass der Dateiname aller Fotografien nicht chronologisch nach Aufnahmeummer umgeschrieben wurde. Dies liegt insofern auf der Hand, als dass die Fotografien aus verschiedenen Quellen stammen. Genau das ist jedoch nötig, um die Bilder im Dateixplorer des Betriebssystems in zeitlich korrekter Reihenfolge anzeigen zu lassen.

Die Aufgabe des Command-Line Tools besteht nun darin, alle Fotografien einzulesen und nach bestimmten Kriterien umzubenennen. Beispielsweise können die Fotografien nach Aufnahmedatum und -uhrzeit umbenannt werden, was gleichzeitig dem voreingestellten Modus entsprechen soll.

1.2 Anforderungen an das Projekt

Eine Anforderung liegt darin, das Programm per Command-Line bedienen zu können. Dabei soll dies sowohl mit als auch ohne übergebene Parameter funktionieren. Sofern keine Parameter angegeben werden, übernimmt das Programm die Bearbeitung im Voreingestellten Modus

Zusätzlich soll es möglich sein, Kriterien im Sinne von Tags angeben zu können. Diese Tags bestimmen später den neuen Namen der Fotografien. Der Tag kann aus beliebig vielen Tags bestehen z.B. "Date/Time Model Make ...".

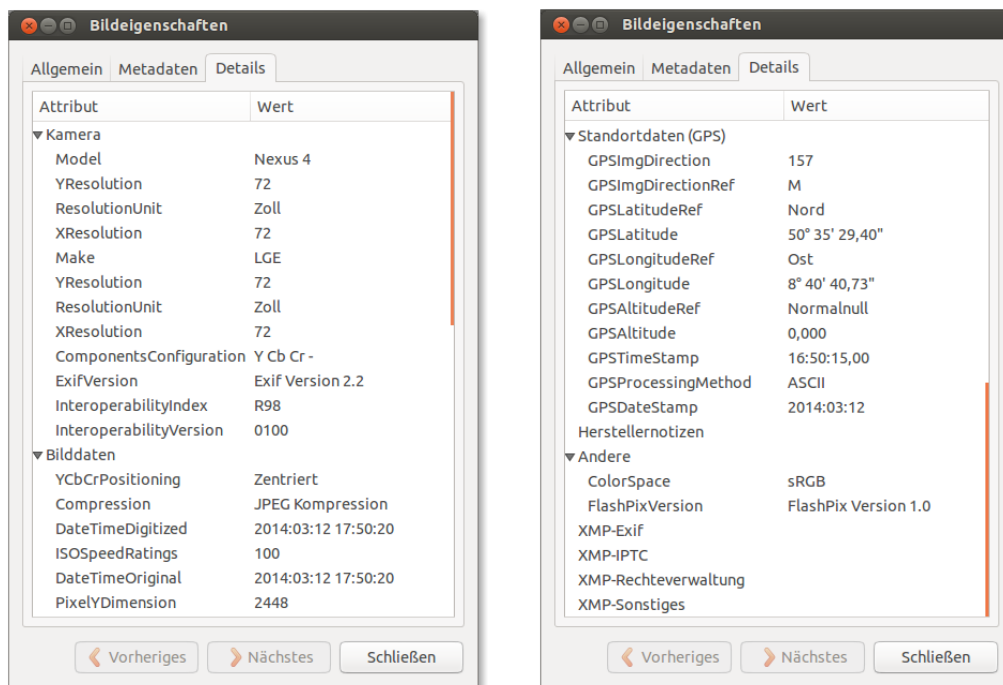
Um die spezifischen Informationen eines Fotos zu erhalten ist es notwendig, die Metadaten der jeweiligen Fotos auszulesen. Es existiert bereits eine beständige Java-Bibliothek, die später noch näher erläutert wird, mit der es möglich ist, Metadaten aus einer digitalen Fotografie zu lesen. Da diese für Java geschrieben wurde, ist eine weitere Anforderung einen Wrapper für einen Clojure zu schreiben, der den Zugriff in einem „closure-like way“ zur Verfügung stellt.

2. Exif - Ein Überblick

Die im Kapitel Anforderungen an das Projekt beschriebenen Metadaten einer digitalen Fotografie werden Exif-Daten genannt. Exif steht für *Exchangeable Image File Format* und wurde als Standard der *Japan Electronic and Information Technology Industries Association* (JEITA) formuliert und zuletzt in Verbindung mit der japanischen *CIPA* (Camera and Imaging Products Association) in der Version 2.3 herausgegeben.^{1 2}

In den Exif-Daten können demnach beispielsweise Informationen über die Aufnahmezeit und -datum des Bildes abgelegt werden. Aber auch der Bezeichner der Kameramarke und -modell bis hin zu GPS-Daten (Global Positioning System), die heutzutage gängige Smartphones und moderne Digitalkameras speichern, können in Exif-Daten repräsentiert werden.¹

Folgende Abbildung zeigt beispielhaft, welche Daten in einem Foto gespeichert sein können.



Screenshots der Bildeigenschaften unter der Linux Distribution Ubuntu

¹ http://de.wikipedia.org/wiki/Exchangeable_Image_File_Format

² http://www.jeita.or.jp/japanese/standard/book/CP-3451C_E/

3. Genutzte Bibliotheken

Dieses Kapitel geht auf verwendete Bibliotheken ein und beschreibt deren Einsatzgebiet im Projekt *Photosort*.

3.1 Metadata-Extractor

Die wichtigste Bibliothek ist der *Metadata-Extractor* von Drew Noakes³, die ursprünglich für die Programmiersprache Java entwickelt wurde. Mit dem *Metadata-Extractor* ist es möglich Exif-Daten eines Fotos auszulesen.

Wie folgender Codeausschnitt zeigt, besteht die Möglichkeit nach bestimmten Metadaten zu suchen oder eine Liste aller Metadaten auszugeben:

```
File jpegFile = new File("IMG_20140302_160056.jpg");
HashMap<String, String> hmCameraInformation = new HashMap<String, String>();
Metadata metadata = ImageMetadataReader.readMetadata(jpegFile);
for (Directory directory : metadata.getDirectories()) {
    for (Tag tag : directory.getTags()) {
        tagName = tag.getTagName();
        tagName = tagName.trim();
        tagName = tagName.replace(" ", "");
        if(tagName.equals(IMAGE_HEIGHT)){
            hmCameraInformation.put(IMAGE_HEIGHT, tag.getDescription());
        }
        if(tagName.equals(IMAGE_WIDTH)){
            hmCameraInformation.put(IMAGE_WIDTH, tag.getDescription());
        }
        if(tagName.equals(MAKE)){
            hmCameraInformation.put(MAKE, tag.getDescription());
        }
        if(tagName.equals(MODEL)){
            hmCameraInformation.put(MODEL, tag.getDescription());
        }
        if(tagName.equals(DATE)){
            hmCameraInformation.put(DATE, tag.getDescription());
        }
    }
}
..
```

Bei diesem Quellcodeausschnitt wird mit der Angabe eines Fotos versucht, die Metadaten zu lesen. Diese sind in verschiedenen Directories unterteilt, in welchen sich verschiedene Tags mit den entsprechenden Werten befinden. Das Auslesen erfolgt dabei über eine Schleife.

Hierbei sind die folgenden Aufrufe am wichtigsten:

³ <https://code.google.com/p/metadata-extractor/>

- `ImageMetadataReader.readMetadata(jpegFile)` - Liest die gesamten Metadaten aus
- `metadata.getDirectories()` - Liest aus den geladenen Metadaten, die Directories aus
- `directory.getTags()` - Liest aus einer Directory, die Tags aus
- `tag.getDescription()` - Liest den entsprechenden Wert, für einen bestimmten Tag aus

Exif Struktur in Metadata-Extractor

Die folgende Darstellung verdeutlichen noch einmal auf abstrakte Weise die verwendete Struktur der Exif-Daten in der Bibliothek.

```
com.drew.metadata.Metadata
  |__ Iterable<Directory>
      |__ com.drew.metadata.Directory
          |__ Collection<Tag>
              |__ com.drew.metadata.Tag
```

Drew Noakes, der Entwickler der Metadata-Extractor Bibliothek, beschreibt die Klasse *Metadata* in seinem Java-Doc mit folgenden Worten:

„A top-level object that holds the metadata values extracted from an image. Metadata objects may contain zero or more objects. Each directory may contain zero or more tags with corresponding values.“⁴

3.2 Clargon

Die Clojure-Erweiterung *Clargon*⁵, die mittlerweile auch unter `Clojure.tools.cli/cli` in der Clojure-Umgebung gefunden werden kann, ist eine Erweiterung, mit der die Command-Line inklusive der Optionen nutzen werden kann. Da diese Bibliothek alle Möglichkeiten zur Nutzung einer Command-Line umsetzt, erfüllt sie die zuvor definierte Anforderung das Programm in der selbigen nutzen zu können.

Im nachfolgenden Code wird anhand eines Codefragments gezeigt, wie eine Main-Funktion in Clojure definiert wird und diese für die Kommandozeile nutzbar gemacht wird.

```
(defn -main [& args]
  (ccl/-with-command-line args
    "DESCRIPTION"
    [
      [help      "Shows exactly this menu"]
      [in        "This specifies the input directory to the pictures"]
      [out       "This specifies the output directory for a new folder"]
      [tag       "To sort and rename the pictures by given tag/s"]
      [newFolder "Create a subfolder in the output directory"]
      extras]
    ...))
```

⁴<https://code.google.com/p/metadata-extractor/source/browse/Source/com/drew/metadata/Metadata.java>

⁵ <https://github.com/gar3thjon3s/clargon>

Die Funktion kann über die Konsole beliebig viele Optionen aufnehmen. Im unteren Teil, nach der Beschreibung der Main-Funktion, wird nur nach definierten Eingaben geschaut. Die nachfolgende Grafik zeigt beispielhaft, wie diese Funktion über die Kommandozeile aufgerufen wird. Hierbei wird das Hilfemenü angezeigt. Die Anzahl der möglichen Optionen, ist bei einer Command-Line nicht begrenzt. Jedoch ist es sinnvoll, nur notwendigen Optionen anzubieten.



```
C:\Users\Wissner\Desktop\Finished 29.03.14>java -jar RunnableJAR/Photosorter-normal.jar --help

////////////////////////////////////
//
//                               Fotosort 1.0
//
//          by Thomas Breitbach
//          Andre Wissner
//
//       https://github.com/andrewissner/sorter1.0
//
////////////////////////////////////

Available useful Tags:

Date/Time - Get the date and time from the metadata
Model     - Get the model and the make of the camera
Make      - Get the camera make
Width     - Get the image width
Height    - Get the image height

Options
--help <arg>      Shows exactly this menu
--in <arg>        This specifies the input directory to the pictures
--out <arg>       This specifies the output directory for a new folder
--tag <arg>       To sort and rename the pictures by given tag/s
--newFolder <arg> Create a subfolder in the output directory

C:\Users\Wissner\Desktop\Finished 29.03.14>pause
Drücken Sie eine beliebige Taste . . .
```

Aufrufen des Projekts in der Kommandozeile unter Windows

3.3 Seesaw

Diese Bibliothek wird zwar eingebunden und getestet, allerdings in dieser Version des Projektes nicht weiter verwendet. *Seesaw*⁶ ist eine Bibliothek, die es ermöglicht grafische Oberflächen in Clojure zu erstellen. *Seesaw* selbst verwendet die von Java stammende Bibliothek *Swing* und *AWT*.

3.4 Clj-http

*Clj-http*⁷ ist ein Clojure-Wrapper für die *Apache HttpComponents* Bibliothek. Sie wird verwendet, um Bilder über das HTTP-Protokoll zu laden und in weiteren Schritten zu verarbeiten.

⁶ <https://github.com/daveray/seesaw>

⁷ <https://github.com/dakrone/clj-http>

4. Exif-Data Wrapper in Clojure

Wie bereits kurz beschrieben, ist das Kernstück dieses Projektes der *Metadata-Extractor*. Da diese Erweiterung ursprünglich für Java entwickelt wurde, wird im Folgenden die Vorgehensweise beschrieben, um diese Erweiterung in Clojure komfortabel einsetzen und nutzen zu können (s. [Exif.clj](#)).

Zunächst wurde die im Kapitel Metadata-Extractor beschriebene Vorgehensweise, Metadaten aus einem Foto auszulesen, mittels drei Funktion in Clojure umgesetzt.

```
(defn exif-for-file
  "Takes an image file (as a java.io.InputStream or java.io.File) and extracts exif
  information into a map"
  [file]
  (let [metadata (ImageMetadataReader/readMetadata file)
        exif-dirs (filter #(re-find exif-directory-regex (.getName %)) (.getDirectories
        metadata))
        tags (map #(.getTags %) exif-dirs)]
    (into {} (map extract-from-tag tags))))
```

Die Funktion *exif-for-file*, im späteren Projektverlauf um einen weiteren Parameter *dir* erweitert, nimmt vorerst als einzigen Parameter eine Datei vom Typ *java.io.File* oder *java.io.FileInputStream* entgegen und liest daraus mit Hilfe der statischen Methode *readMetadata* der Klasse *ImageMetadataReader* die Exif-Daten des Bildes aus. Das zurückgegebene Objekt wird an die Variable *metadata* gebunden und ist mittels *let* im entsprechenden Kontext verfügbar.

Als nächstes werden aus diesem *Metadata*-Objekt durch die Methode *getDirectories* alle im Bild verfügbaren Metadata-Ordner ausgelesen und als iterierbare Collection zurückgegeben. *Directories* bündeln, wie in Kapitel Exif Struktur in Metadata-Extractor beschrieben, Exif-Tags eines bestimmten Typs. Die Funktion *filter* hat die Eigenschaft, alle Elemente einer Collection, in diesem konkreten Fall alle Exif-Directories, auf ein Prädikat anwenden zu können und nur solche zurückzugeben, die vom Prädikat zurückgegeben werden. Das Prädikat ist in diesem konkreten Fall eine anonyme Funktion (*#(...)*), die mittels *re-find* (steht für regex find) solche Directories zurück gibt, die in *exif-directory-regex* definiert wurden.

```
(def exif-directory-regex
  (re-pattern (str "(?i)(" (join "|"
    ["GPS" "Exif" "JPEG" "JFIF"
     "Agfa" "Canon" "Casio" "Epson"
     "Fujifilm" "Kodak" "Kyocera"
     "Leica" "Minolta" "Nikon" "Olympus"
     "Panasonic" "Pentax" "Sanyo"
     "Sigma/Foveon" "Sony"]) ")"))))
```

Dies ist eine Möglichkeit, um sich auf ein paar wenige, wichtige Directories beschränken zu können. Damit beinhaltet zu diesem Zeitpunkt *exif-dirs* eine Collection aller Directories, die in dem regulären Ausdruck definiert wurden.

Im nächsten Schritt werden alle Tags der an *exif-dirs* gebundenen Directories ausgelesen. Hierbei kommt die Funktion *map* zum Einsatz, die alle Elemente einer Collection auf eine Funktion anwendet und als lazy sequence zurück gibt. Folgendes Beispiel aus der Clojure Onlinedokumentation veranschaulicht das Verhalten:

```
user=> (map inc [1 2 3 4 5])
(2 3 4 5 6)
```

Jedes Element der Collection wird um genau eins erhöht.

Der Einsatz von *map* an dieser Stelle ist nötig, da, wie bereits angesprochen, *exif-dirs* eine Collection aller Directories ist die wiederum als Collection ihre zugehörigen Tags halten. Nun sind alle Tags der in *exif-directory-regex* gelisteten Directories an *tags* gebunden. Allerdings muss auch hier die Struktur im Hinterkopf gehalten werden. Die Java-Funktion *getTags* der Bibliothek gibt eine Collection von Tags zurück. Durch *map* besteht *tags* nun aus einer Collection von Collections aus Tags.

Anschließend wird die Funktion *extract-from-tag* auf jede Collection in *tags* angewendet und der Tag-Name und Tag-Wert ausgelesen und in eine *HashMap* verpackt, um diese als Schlüssel-Wert-Paare repräsentieren zu können. Da es unnötig ist, für jedes Schlüssel-Wert-Paar eine einzelne *HashMap* zu verwenden, wird mittels *into* alle Paar in eine neue leere Map gesetzt und anschließend zurückgegeben.

```
(defn- extract-from-tag
  [tag]
  (into {} (map #(hash-map (.getTagName %) (.getDescription %)) tag)))
```

Das Gleiche geschieht in der letzten Zeile der Funktion *exif-for-file*:

```
...
(into {} (map extract-from-tag tags))
...
```

Damit hat man alle Tags aller Directories in einer Map und kann somit schnell und unkompliziert darauf zugreifen (bspw. zum filtern).

Mit der Funktion *exif-for-file* ist es zu diesem Zeitpunkt möglich Exif-Daten eines Bildes, das als Objekt der Klasse *java.io.File* oder *java.io.FileInputStream* übergeben wird, auszulesen. Allerdings sollen neben Objekten vom Typ *File* auch andere als Argument übergeben werden können. Zusätzlich soll der Wrapper zukünftig leicht zu erweitern sein. Aus diesem Grund wurde das Protokoll *exif* definiert, das die Funktion *exif-data* mit zwei Deklarationen ([x] oder [x tag-or-dir]) bereitstellt.

```
(defprotocol exif
  (exif-data
    [x]
    [x tag-or-dir]))
```

Diese Funktion kann entweder einen Parameter, nur die einzulesende Datei, oder zusätzlich einen zweiten Parameter entgegen nehmen. Der Zweite Parameter ist als Filter zu sehen der angibt, welche Exif-Tags zurückgegeben werden sollen. Implementiert wird das Protokoll mit Hilfe des *extend-protocol* Macros gefolgt vom zuvor definierten Bezeichner *exif*.

```
(extend-protocol exif
  ...
)
```

Nun kann der Datentyp des ersten Parameters definiert und die spezifische Implementierung für diesen angegeben werden. Sofern ein zweites Argument angegeben wurde, wird dieses auf seine Klassenherkunft geprüft und mit diesem Wissen die korrekte Funktion aufgerufen. Die Bezeichner der Funktionen sind selbsterklärend. Demzufolge wird die Logik vorliegenden Dokument nicht explizit aufgeführt. Bei Interesse kann sie jedoch im öffentlichen Git-Repository auf GitHub.com eingesehen werden.

Funktionen neben der bereits genannten *exif-for-file*:

Bezeichner	Verhalten
exif-tag-for-file	Liefert einen speziellen Exif-Tag, sofern vorhanden
exif-tags-for-file	Gibt die in der übergebenen Collection angegebenen Exif-Tags zurück, sofern vorhanden.
exif-for-filename	Ermöglicht das Auslesen von Exif-Daten eines Bildes, dessen Pfad als String spezifiziert wurde.
exif-tag-for-filename	Pendant zu <i>exif-tags-for-file</i> . Erwartet allerdings einen String
exif-tags-for-filename	Pendant zu <i>exif-tags-for-file</i> . Erwartet allerdings einen String.
exif-for-url	Ermöglicht das Auslesen von Exif-Daten eines Bildes, das über eine URL im Internet erreichbar ist.
exif-tag-for-url	Pendant zu <i>exif-tag-for-file</i> für Objekte vom Typ java.net.URL
exif-tags-for-url	Pendant zu <i>exif-tags-for-file</i> für Objekte vom Typ java.net.URL

Intern rufen diese Funktionen immer die Funktion *exif-for-file* auf. Vorher findet im Großen und Ganzen Typumwandlung statt.

Folgender Codeausschnitt zeigt die Implementierung des Protokolls für die drei Klassen *File*, *String* und *URL*:

```
(extend-protocol exif
  File
```

```

(exif-data
  ([f tag-or-dir]
    (if (instance? String tag-or-dir)
      (exif-tag-for-file f tag-or-dir)
      (if (instance? Directory tag-or-dir)
        (exif-for-file f tag-or-dir)
        (exif-tags-for-file f tag-or-dir))))
    ([f]
      (exif-for-file f)))

String
(exif-data
  ([s tag-or-dir]
    (if (url? s)
      ;s is treated as url
      (if (instance? String tag-or-dir)
        (exif-tag-for-url s tag-or-dir)
        (if (instance? Directory tag-or-dir)
          (exif-for-url s tag-or-dir)
          (exif-tags-for-url s tag-or-dir)))
      ;s isn't an url -> check fs
      (if (instance? String tag-or-dir)
        (exif-tag-for-filename s tag-or-dir)
        (if (instance? Directory tag-or-dir)
          (exif-for-filename s tag-or-dir)
          (exif-tags-for-filename s tag-or-dir)))))
    ([s]
      (if (url? s)
        (exif-for-url (URL. s))
        (exif-for-filename s))))

URL
(exif-data
  ([url]
    (exif-for-url url))
  ([url tag-or-dir]
    (if (instance? String tag-or-dir)
      (exif-tag-for-url url tag-or-dir)
      (if (instance? Directory tag-or-dir)
        (exif-for-url url tag-or-dir)
        (exif-tags-for-url url tag-or-dir)))))

```

Die Implementierung der Funktionen *exif-for-...* ist zwar nicht zwingend notwendig, man hätte die Logik auch direkt im Protokoll implementieren können, verleiht dem Wrapper jedoch eine weit aus bessere Lesbar- und Wartbarkeit.

5. Main.clj - Der Einstiegspunkt des Projektes

Im Kapitel 4 wurde bereits detailliert erläutert, wie der Exif-Wrapper funktioniert. Im Folgenden wird auf den Aufbau der [Main.clj](#) eingegangen. In diesem Namespace wird einerseits der Exif-Wrapper eingebunden und zum Lesen der Metadaten verwendet, andererseits befindet sich hier die Logik zum Erzeugen der neuen Dateinamen/Dateien.

Die wichtigste Funktion im Namespace *Main* ist *main*. Die *Main*-Funktion wird als Startpunkt verwendet, sobald das Projekt aus der Konsole aufgerufen wird.

```
(defn -main [& args]
  (ccl/with-command-line args
    "
// ////////////////////////////////////////
//                                     //
//                               Photosort 1.0                               //
//                                     //
//                               by Thomas Breitbach                       //
//                               Andre Wissner                             //
//                                     //
//                               https://github.com/andrewissner/sorter1.0   //
//                                     //
// ////////////////////////////////////////

Available useful Tags:

Date/Time - Get the date and time from the metadata
Model    - Get the model and the make of the camera
Make     - Get the camera make
Width    - Get the image width
Height   - Get the image height
"

    [
      [help    "Shows exactly this menu"]
      [in      "This specifies the input directory for the pictures"]
      [out     "This specifies the output directory for a new folder"]
      [tag     "To sort and rename the pictures by given tag/s"]
      [newFolder "Create a subfolder in the output directory"]
      extras]

    (if
      (not (clojure.string/blank? newFolder))
      (def theNewFolder newFolder)
      (def theNewFolder ""))

    (if
      (not (clojure.string/blank? in))
      (def theInput in)
      (def theInput "./"))
```

```

    )

    (if
      (not (clojure.string/blank? out))
      (def theOutput out)
      (def theOutput "./")
    )

    (if
      (not (clojure.string/blank? tag))
      (def theTag (split-whitespace tag))
      (def theTag ["Date/Time"])
    )

    (println "\n\nThe script is running with these configuration:\n\nTAG: " theTag "\nIN: " theInput
      "\nOUT: " theOutput "\n\n")
    (println "Run this configuration? (J/N)")
    (let [input (clojure.string/lower-case(read-line))]
      (if (= input "j")
        (copy-image-with-format theInput theOutput theTag theNewFolder)
        (println "Nothing to do here!")
      )
    )
  )
)
)

```

Die Methode enthält nach dem Methoden-Kopf, eine Beschreibung der Methode. Diese kann mittels `--help` aufgerufen werden. Nach der Definition der Beschreibung, erfolgt die Definition der möglichen Eingabeparameter.

```

[
  [help      "Shows exactly this menu"]
  [in        "This specifies the input directory for the pictures"]
  [out       "This specifies the output directory for a new folder"]
  [tag       "To sort and rename the pictures by given tag/s"]
  [newFolder "Create a subfolder in the output directory"]
  extras]

```

Für das Projekt sind in erster Linie nur die obigen Parameter wichtig. Aufgerufen werden die Parameter in der Konsole mit `--help`, `--in "ort/zu/den/fotos"` `--out "ausgabe/ort"`, `--tag "Date/Time"` und für den Fall, dass die Fotos in einem neuen Ordner kopiert werden sollen, kann man mit `--newFolder "Name"` einen neuen Ordner anlegen und die Fotografien werden in diesen Ordner gespeichert. Nach dieser Definition erfolgt eine Abfrage der möglichen Parameter. Wie in Kapitel 1.2 Anforderungen an das Projekt angesprochen, soll es möglich sein das Programm ohne Parameter aufzurufen. Aus diesem Grund fiel die Entscheidung auf eine Standarteinstellung, die folgende Konfiguration beinhaltet:

- tag: Aufnahmedatum und die Aufnahmezeit

- in: ./
- out: ./

```
(if
  (not (clojure.string/blank? tag))
  (def theTag (split-whitespace tag))
  (def theTag ["Date/Time"]))
```

Am Ende befindet sich noch eine Abfrage, die den Benutzer fragt, ob die Konfiguration verwendet werden soll. Für die einfache Handhabung gibt es zusätzlich das Projekt als *Photosorter-easy.jar*, die das Sortieren mit den Standardeinstellungen (Date/Time) und ohne Ausgabe auf der Kommandozeile vornimmt.

Die Kernmethode im Namespace *Main*, ist die *copy-image-with-format* Methode. Sie enthält die Logik, die für das Ändern der Fotografien notwendig ist.

```
(defn- copy-image-with-format
  "Function to read the exif data and write new pictures with new name"
  [theIn theOut tag nFolder]

  ;(def tagList (split-whitespace tag)); only you come from repl
  ;(println tag)
  (if (not (clojure.string/blank? nFolder))
    (if (.mkdir (new File (str theOut (check-path-seperator theOut) nFolder)))
      (println "New folder created: " nFolder)
      (println "Folder already exists!"))
    )
  )

  (def theString "")
  (def res (list-images (str theIn)))

  (doseq [x res]
    (doseq [t tag]
      (if (= t "Date/Time")
        (def theString
          (str theString
            (let [date (exif-data (str theIn (check-line-seperator theIn) x) "Date/Time Original")]
              (if (not (nil? date))
                (create-new-date
                  (split-the-date date)))
              )))

          (if (= t "Model")
            (def theString
              (str theString
                (let [model (exif-data (str theIn (check-line-seperator theIn) x) t)]
                  (if (not (nil? model))
                    (create-new-model
                      (split-whitespace model))
```

```

))))

(if (= t "Make")
  (def theString
    (str theString
      (let [make (exif-data (str theIn (check-line-seperator theIn) x) t)]
        (if (not (nil? make))
          (create-new-model
            (split-whitespace make))
          )))))

(if (= t "Width")
  (def theString
    (str theString
      (let [width (exif-data (str theIn (check-line-seperator theIn) x) "Image Width")]
        (if (not (nil? width))
          (create-new-model
            (split-whitespace width))
          )))))

(if (= t "Height")
  (def theString
    (str theString
      (let [height (exif-data (str theIn (check-line-seperator theIn) x) "Image Height")]
        (if (not (nil? height))
          (create-new-model
            (split-whitespace height))
          )))))

(def theString
  (str theString
    (exif-data (str theIn (check-line-seperator theIn) x) t)
  ))
)))));EOF DOSEQ
(if (not (clojure.string/blank? nFolder))
  (def newOut (str theOut (check-path-seperator theOut) nFolder (check-line-seperator
theOut) theString "_"))
  (def newOut (str theOut (check-path-seperator theOut) theString "_"))
  )

(copy-file
  (str theIn (check-path-seperator theIn) x)
  (str newOut x)
  )
(def theString "")
)
(println "Job done!")
)

```


In dieser Methode werden die nötigen Methoden aufgerufen, die zum Auslesen von Ordnern und der Metadaten nötig sind. Auch Hilfsmethoden, wie z.B. das Umschreiben des Datums, werden hier aufgerufen. Die aufgerufene Methode *check-path-seperator* wird verwendet, um die Eingaben der Pfade zu überprüfen. In dem unteren Codeausschnitt wird diese Methode verwendet, um einen neuen Ordner zu erstellen.

```
(if (not (clojure.string/blank? nFolder))
  (if (.mkdir (new File (str theOut (check-path-seperator theOut) nFolder)))
    (println "New folder created: " nFolder)
    (println "Folder already exists!")
  ))
```

Um die Bilder nun aus einem gegebenen Ordner auszulesen, reicht die folgende Zeile. Der Methode wird der Eingabeort übergeben, der in der *Main*-Funktion definiert bzw. vom Benutzer eingegeben worden ist.

```
(def res (list-images (str theIn)))
```

Die Abarbeitung der beliebig vielen Fotografien werden in dieser Funktion in Sequenzen abgearbeitet. Nun erfolgt der erste wichtige Teil der Funktion, das Aufrufen der Metadaten eines Fotos. *Exif-data* wird der Pfad eines Fotos übergeben und den Tag, der als Wert zurückgegeben werden soll. Als Rückgabewert für den Tag *Date/Time*, wird beispielsweise das Aufnahmedatum und die Aufnahmezeit zurückgegeben.

```
(exif-data (str theIn (check-path-seperator theIn) x) t)
```

Im nachfolgenden Codeausschnitt wird exemplarisch gezeigt, wie die Auswertung der eingegebenen Tags abgearbeitet wird. Bei der Auswertung des Datums wird hier noch eine Funktion verwendet, die das Datum und die Zeit in die Bestandteile Tag, Monat, Jahr, Stunde, Minute und Sekunde zerlegen. Aus der Zerlegung wird im Anschluss ein neues Datum mit eigener Formatierung erzeugt. Das Ganze wird anschließend in einem *String* gespeichert. Dieser String ist Bestandteil für den neuen Dateinamen des Bildes.

```
(if (= t "Date/Time")
  (def theString
    (str theString
      (create-new-date
        (split-the-date (exif-data (str theIn (check-path-seperator theIn) x) t))))))
```

Diese Art der Auswertung wird für alle vorher definierten Tags angewandt. Am Ende der Funktion erfolgt das eigentliche Kopieren der Fotografien mit neuem Namen.

```
(copy-file
  (str theIn (check-path-seperator theIn) x)
  (str newOut x)
)
```

Die Methode *copy-file* erwartet das Originalfoto und den neuen Ort mit dem neuen Namen der Fotografie. Wie das Programm genau genutzt werden kann, wird näher in Kapitel 7 Ausführen des Programms beschrieben.

6. Projektkonfiguration

Um das Projekt weiter entwickeln zu können ist es wichtig, die nachfolgende Konfiguration durchzuführen. Die Konfiguration bezieht sich in erster Linie auf das Einrichten einer lokalen JAR-Datei in *Maven* und Clojure.

6.1 Leiningen und lokale JARs

Es gibt verschiedene Wege, wie man lokale JAR-Dateien in die *Projekt.clj* (Konfigurationsdatei von Leiningen) in Clojure einbinden kann. Eine Möglichkeit wird mittels

```
:resource-paths ["resources/Datei.jar" "resources/Datei2.jar"])
```

gegeben. Da diese Möglichkeit für das *Photosorter* Projekt nicht funktionierte, wurde der etwas aufwendigeren Weg über ein lokales *Maven* Repository gewählt.

Um lokale JAR-Dateien unter Clojure mittels *Maven* zu verwenden, sind die folgenden Schritte unter Windows nötig:

1. Installieren der Maven Anwendung. Dabei befolgt man am Besten schon die Installationsanweisung des Herstellers (Englisch)⁸:
 - a. Unzip the distribution archive, i.e. apache-maven-3.2.1-bin.zip to the directory you wish to install Maven 3.2.1. These instructions assume you chose C:\Program Files\Apache Software Foundation. The subdirectory apache-maven-3.2.1 will be created from the archive.
 - b. Add the M2_HOME environment variable by opening up the system properties (WinKey + Pause), selecting the "Advanced" tab, and the "Environment Variables" button, then adding the M2_HOME variable in the user variables with the value C:\Program Files\Apache Software Foundation\apache-maven-3.2.1. Be sure to omit any quotation marks around the path even if it contains spaces. Note: For Maven 2.0.9, also be sure that the M2_HOME doesn't have a '\' as last character.
 - c. In the same dialog, add the M2 environment variable in the user variables with the value %M2_HOME%\bin.
 - d. Optional: In the same dialog, add the MAVEN_OPTS environment variable in the user variables to specify JVM properties, e.g. the value -Xms256m -Xmx512m. This environment variable can be used to supply extra options to Maven.
 - e. In the same dialog, update/create the Path environment variable in the user variables and prepend the value %M2% to add Maven available in the command line.

⁸ <http://maven.apache.org/download.cgi#Installation>

- f. In the same dialog, make sure that `JAVA_HOME` exists in your user variables or in the system variables and it is set to the location of your JDK, e.g. `C:\Program Files\Java\jdk1.7.0_51` and that `%JAVA_HOME%\bin` is in your Path environment variable.
 - g. Open a new command prompt (Winkey + R then type `cmd`) and run `mvn --version` to verify that it is correctly installed.
2. Zur Überprüfung, ob *Maven* nun wirklich installiert worden ist, geben Sie folgenden Befehl in die Konsole ein:

```
$ mvn --version
```

3. Das Hinzufügen der JAR-Datei in das lokale Repository, erfolgt über die Eingabe der Befehle:

```
$ git clone https://code.google.com/p/metadata-extractor/  
$ cd metadata-extractor  
$ mvn install
```

Hierbei wird zuerst das Repository in das aktuelle Verzeichnis geklont. Anschließend wechselt man in das durch den Klon-Vorgang angelegte Verzeichnis und führt darin den Install-Befehl aus.

4. Nun muss noch der Konfigurationsdatei von Leiningen (*Project.clj*) folgendes unter `:dependencies` hinzugefügt werden:

```
:dependencies [ .. [com.drewnoakes/metadata-extractor "2.7.0-SNAPSHOT"] ..]
```

5. Um sicherzustellen, dass Clojure die JAR-Datei findet, sollte man mit der Konsole in den Projekt Ordner navigieren und *lein deps* starten

7. Ausführen des Programms

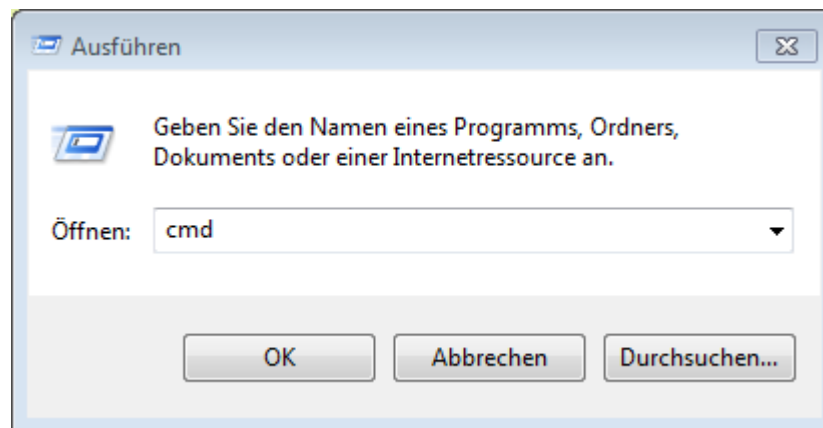
Folgende Parameter können beim Starten der *Photosorter-normal.jar* angegeben werden:

--help	Ruft das Hilfemenü auf
--in ""	Gibt an, von welchem Ort die Bilder eingelesen werden sollen
--out ""	Gibt den Ausgabeort der Bilder an
--tag ""	Legt fest, welche Tags verwendet werden sollen.
" Date/Time Model Make Width Height "	
--newFolder	Gibt einen neuen Ordner an, in dem die Bilder gespeichert werden

7.1 Ausführung unter Windows 7/8

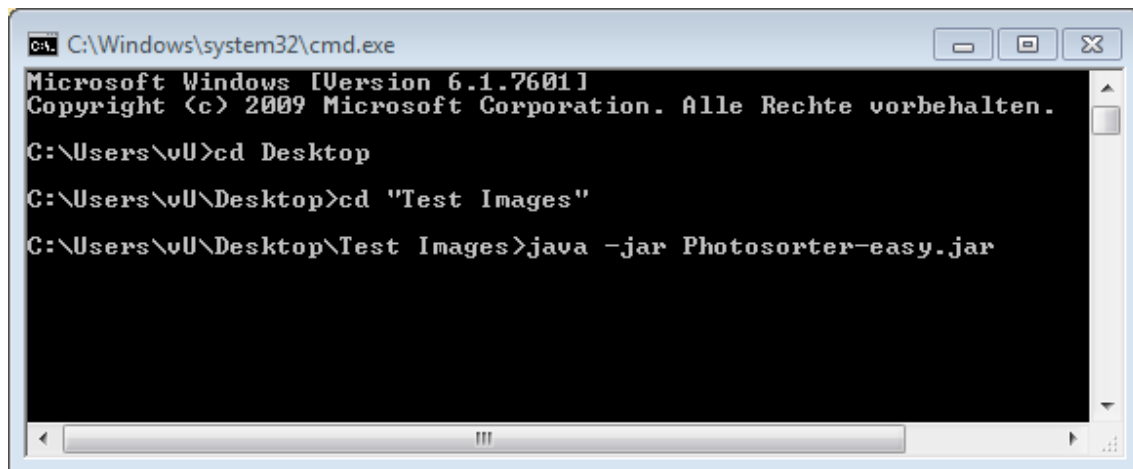
Möglichkeit 1 - Starten ohne Parameter

1. Um die Command-Line(CMD) zu starten, drücken Sie die *Windows-Taste* + *R-Taste* um die Ausführung aufzurufen und schreiben *CMD* in das Eingabefeld.



Ausführungsfenster unter Windows

2. Kopieren Sie die JAR-Projekt Datei, in dem Ordner, wo die Fotografien liegen.
3. Navigieren Sie nun zu dem Ordner, wo die zu sortierenden Fotografien liegen.
4. Mit dem Befehl `java -jar Photosorter-normal.jar`, starten Sie die das Sortieren. Ohne Angaben von weiteren Parameter, werden die Fotografien nach dem Aufnahmedatum und der Aufnahmezeit kopiert.



Aufrufen des Projektes ohne Parameter

Achten Sie dabei darauf, dass die JAR-Datei in dem selben Ordner liegt, wo auch die Fotografien liegen.

Möglichkeit 2 - Starten mit Angaben von Parameter

1. Folgen Sie den Angaben 1 bis 2 aus Möglichkeit eins.
2. Die JAR-Datei muss nun nicht in dem selben Ordner liegen wie die Fotografien.
3. Navigieren Sie mit der Command-Line zu dem Ordner, wo die Photosorter-normal.jar liegt und starten Sie die JAR-Datei mit dem folgenden Befehl die.

```
$ java -jar Photosorter-normal.jar --in "C:\\pfad\\zu\\den\\bilder" --out "C:\\pfad\\zu\\einem\\vorhandenen\\ordner" --tag "Date/Time"
```

Möglichkeit 3 - Starten der Photosorter-easy.jar

1. Die beiliegende *Photosorter-easy.jar* mit einem Doppelklick starten. Diese JAR-Datei kopiert und sortiert die Bilder nach dem Aufnahmedatum und der Aufnahmenzeit. Diese ist die einfachste Möglichkeit, die Fotografien umzubenennen und sie zu sortieren.
2. Das war es. Nun sollten die Bilder mit dem Datum im Namen erstellt worden sein.

7.2 Ausführung unter Mac OSX

Möglichkeit 1 - Starten mit Angaben von Parameter

1. Um unter Mac OSX das Terminal zu starten, drücken Sie *Cmd + Leertaste* und geben Sie im Suchfeld *Terminal* ein. Eine weitere Möglichkeit ist es, im Dienstprogrammordner das Terminal zu starten. Dieser Ordner befindet sich im Programmordner.
2. Befolgen Sie als nächstes die Schritte 2–4 aus Kapitel 7.1 Möglichkeit 1 - Starten ohne Parameter oder aus Kapitel 7.1 Möglichkeit 2 - Starten mit Angaben von Parameter die Schritte 2 und 3.

Möglichkeit 2 - Starten der Photosorter-easy.jar

1. Die beiliegende *Photosorter-easy.jar* mit einem Doppelklick starten. Diese JAR-Datei kopiert und sortiert die Bilder nach dem Aufnahmedatum und der Aufnahmezeit. Diese ist die einfachste Möglichkeit, die Fotografien umzubenennen und sie zu sortieren.
2. Das war es. Nun sollten die Bilder mit dem Datum im Namen erstellt worden sein.

7.3 Ausführung unter Linux

Möglichkeit 1 - Starten ohne Parameter

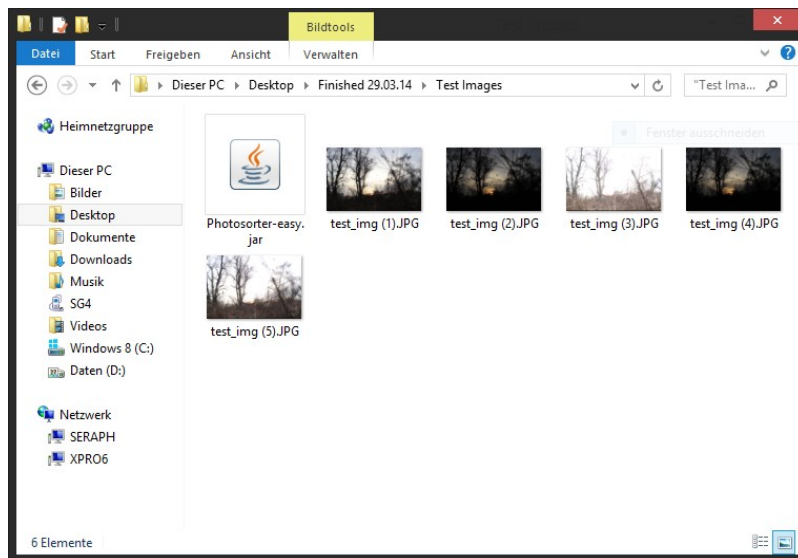
1. Klicken Sie mit der linken Maustaste auf das Ubuntu Symbol in der linken oberen Menüleiste und tippen Sie „Terminal“ in das Textfeld. Alternativ lässt sich ein Terminal auch mit Tastenkombination **Strg + Alt + T** öffnen.
2. Befolgen Sie als nächstes die Schritte 2–4 aus Kapitel 7.1 Möglichkeit 1 - Starten ohne Parameter oder aus Kapitel 7.1 Möglichkeit 2 - Starten mit Angaben von Parameter die Schnitte 2 und 3.

Möglichkeit 2 - Starten der Photosorter-easy.jar

1. Die beiliegende *Photosorter-easy.jar* mit Rechtsklick, „Öffnen mit“ → „OpenJDK Java / Runtime“ starten. Diese JAR-Datei kopiert und sortiert die Bilder nach dem Aufnahmedatum und der Aufnahmezeit. Diese ist die einfachste Möglichkeit, die Fotografien umzubenennen und sie zu sortieren.
2. Das war es. Nun sollten die Bilder mit dem Datum im Namen erstellt worden sein.

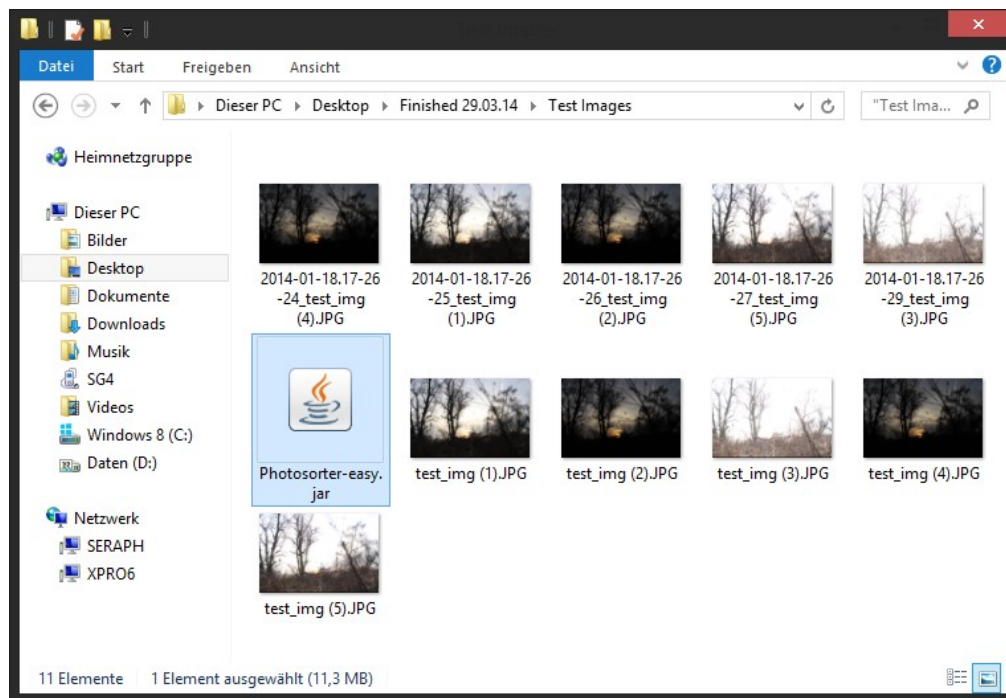
7.4 Ergebniss eines Durchlaufes unter Windows 8

Im Folgenden, wird kurz gezeigt, wie das Ergebnis aussehen sollte, wenn man die *Möglichkeit 2 - Starten der Photosorter-easy.jar* des jeweiligen Betriebssystems befolgt. Das gezeigte Ergebnis wird unter Windows 8 gezeigt. Die *Photosorter-easy.jar* wurde in einen Ordner mit einer Serie von Fotografien kopiert. Nach dem Kopieren, wird die *Photosorter-easy.jar* ausgeführt.



Ordner mit Fotografien und der Photosorter-easy.jar

Nach dem Ausführen der Datei, werden die vorhandenen Fotografien kopiert und mit dem entsprechenden Datum umbenannt. Am Besten ist es jedoch, wenn man die JAR-Datei mit Parameter aufruft. Man sollte einen neuen Ordner angeben, wo die neuen Fotografien abgespeichert werden.



Nach der Ausführung der JAR-Datei

8. Ausblick

Der wichtigste Teil, bei einer Fotosortierung durch das Auslesen der Metadaten, ist der Exif-Data Wrapper aus dem Kapitel Exif-Data Wrapper in Clojure. Das Aktuelle Projekt, was unter github.com öffentlich zur Verfügung steht, ist momentan nur über die Konsole oder durch direktes anklicken der JAR-Datei ausführbar aufrufbar. Die *Photosorter-normal.jar*, ist momentan für erfahrene Terminal/Konsolen Anwender gedacht. Für die weiter Entwicklung ist es notwendig, dass es für Anwender die wenig bis keine Erfahrung mit dem Terminal oder mit der Konsole haben, eine grafische Oberfläche entwickelt wird. Das Projekt wurde mit der Clojure-Erweiterung Seesaw schon darauf vorbereitet.

Das Erstellen der Fotografien, erfolgt momentan nur durch das Kopieren der Fotografien mit neuen Namen. Da das Überschreiben und Löschen von Fotografien derzeit nicht möglich ist, ist es daher sinnvoll, dies weiter zu Entwickeln.

9. Quellcode zum Projekt

Projektcode - <https://github.com/andrewissner/sorter1.0>