DOKUMENTATION: STUDIENARBEIT PYTHON FÜR ALLE(S)

Philip Bartmann

Medieninformatik, 3.Semester, p.bartmann@oth-aw.de

DOKUMENTATION: STUDIENARBEIT PYTHON FÜR ALLE(S)

INHALTSVERZEICHNIS

1. STRUKTUR

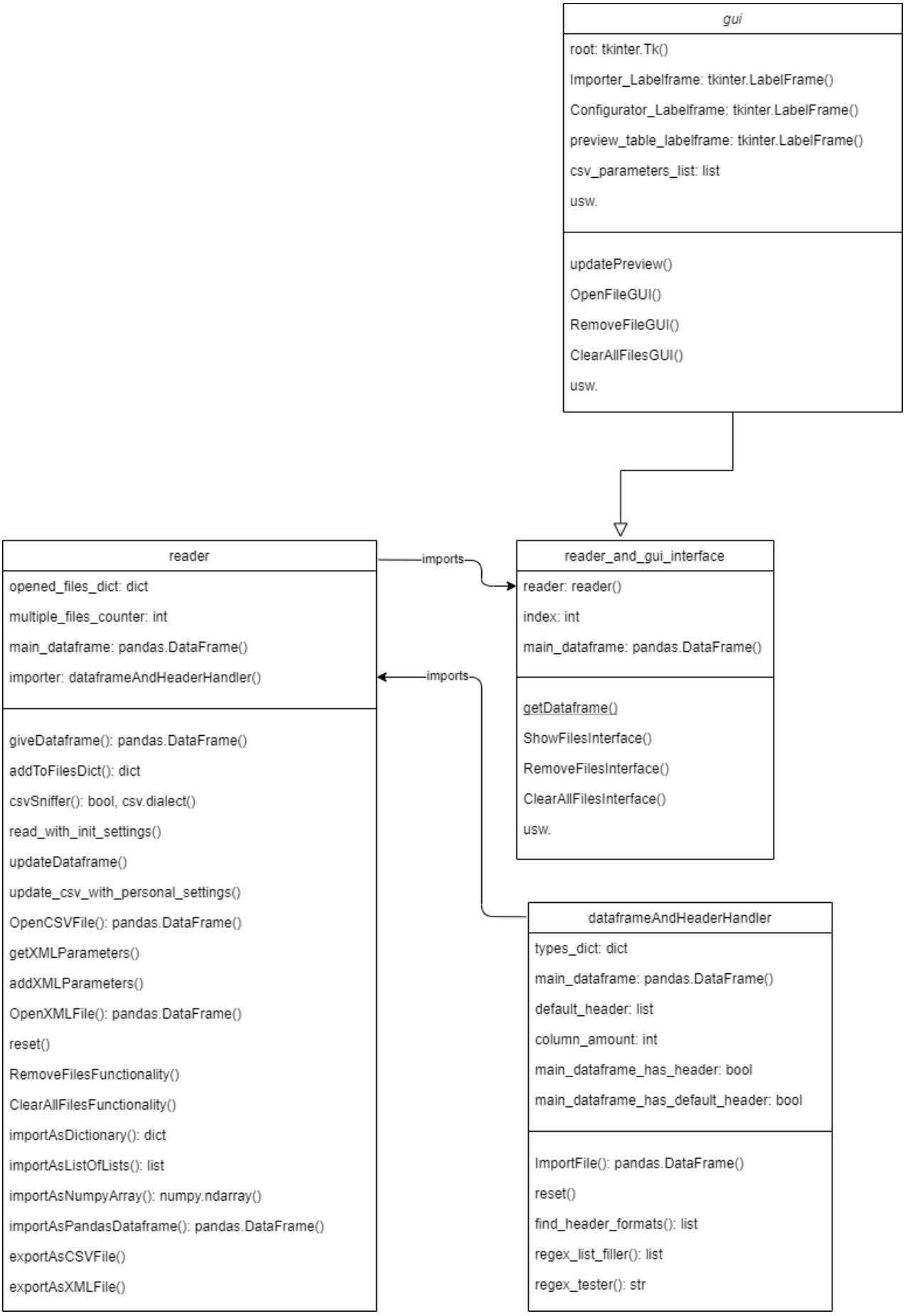

Abbildung 1: UML-Klassendiagramm der verwendeten Klassen

1.1 KLASSEN

Im Folgenden werden die Klassen, die im Programm verwendet werden, vorgestellt und beschrieben. Die Klassen sind: GUI, reader_and_gui_interface, reader und dataframeAndHeaderHandler.

1.1.1 KLASSE GUI

Die Klasse GUI ist für alle sichtbaren Widgets der Anwendung zuständig. Dazu gehören Buttons, Labelframes, Listboxen, Textboxen, Radiobuttons und Checkboxen.

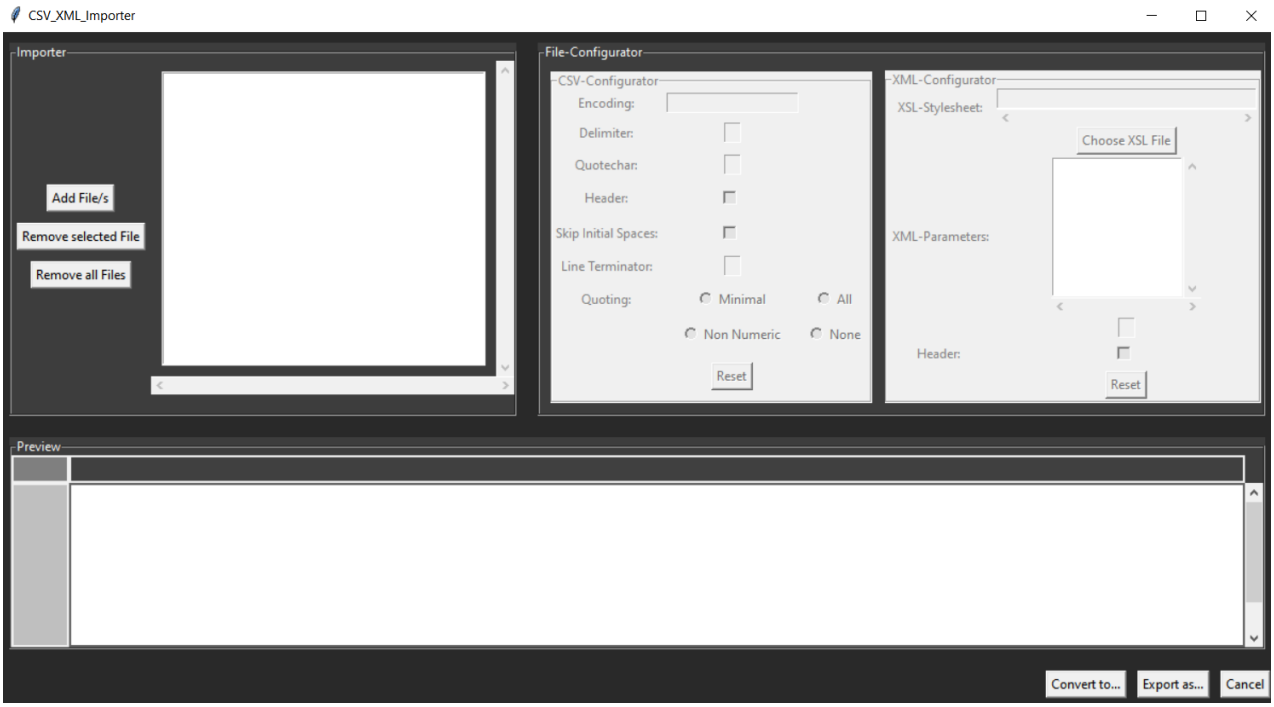

Abbildung 2: Screenshot der Anwendung ohne Eingaben

Es gibt drei Knöpfe im Labelframe, das „Importer“ genannt wurde.

„Add File/s“ öffnet bei Anklicken einen Datei Dialog, der den Benutzer eine oder mehrere CSV und/oder XML Dateien auswählen lässt. Die ausgewählten Dateien werden danach in der rechts stehenden Listbox mit ihrem absoluten Dateipfad gezeigt, wenn die ausgewählten Dateien dieselbe Spaltenanzahl haben. Damit der komplette Dateipfad der Dateien und alle ausgewählten Dateien sichtbar werden, wurden zwei Scrollbalken auf der rechten und unteren Seite der Listbox eingefügt.

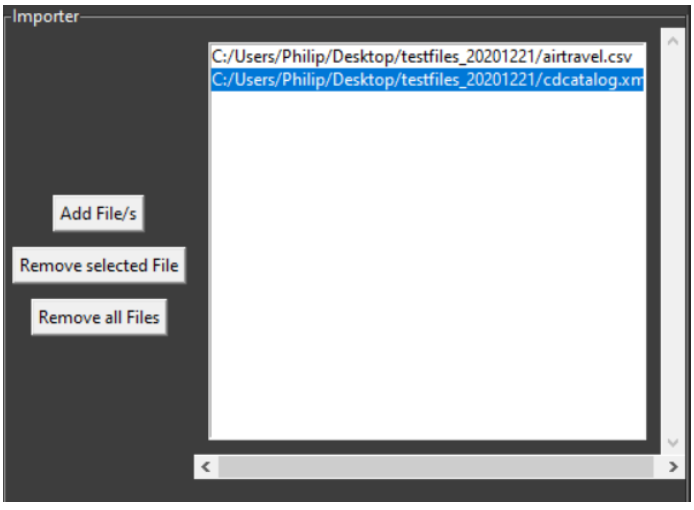

Abbildung 3: Screenshot der Listbox mit ausgewählten Dateien

„Remove selected File“ löscht den ausgewählten Dateipfad aus der Listbox. Ausgewählte Dateipfade erkennt man dadurch, dass der Dateipfad mit einem blauen Rahmen hinterlegt wurde. Dieses Auswahlverfahren wurde auch für die Konfiguration der Dateien verwendet.

„Remove all Files“ löscht alle Dateipfade aus der Listbox, um die GUI komplett zurückzusetzen.

Im Labelframe „Configurator“ sind alle Widgets untergebracht, die für die Konfiguration von Dateien der zwei möglichen Dateitypen CSV und XML benötigt werden. Die Funktionsweise der einzelnen Widgets und Labelframes wird im Kapitel „Benutzung und Bedienung“ weiter ausgeführt.

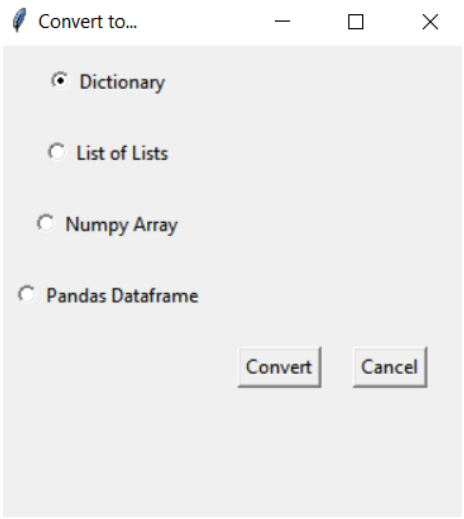

Abbildung 4: Konvertierungs Fenster

Unter diesen beiden Labelframes befindet sich eine Tabelle aus dem „pandastable“-Modul. Es zeigt eine Vorschau der ausgewählten Dateien an und aktualisiert sich automatisch, falls ein Parameter einer Datei geändert wurde oder eine neue Datei ausgewählt wurde, die sich mit den zuvor ausgewählten Dateien verknüpfen lässt. Falls dies nicht der Fall ist wird die Datei nicht eingelesen und eine Fehlermeldung angezeigt.

Zuletzt sind noch drei weitere Knöpfe in Abbildung 3 unter der Tabelle zu sehen.

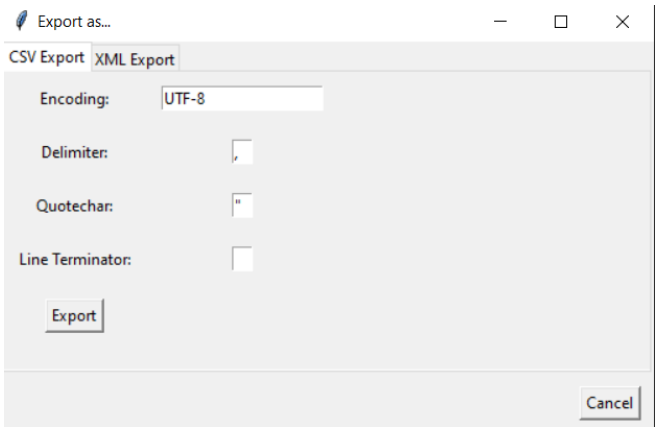

Abbildung 5: Export-Fenster für CSV Export

„Convert to…“ öffnet bei Anklicken ein weiteres Fenster, das den Nutzer zwischen vier möglichen Konvertierungen auswählen lässt. Zur Auswahl stehen: „Dictionary“, „List of Lists“, „Numpy Array“ und “Pandas Dataframe”. Dies ist in Abbildung 4 zu sehen.

Die Auswahl wird durch den „Convert“-Knopf bestätigt. Bei Drücken dieses Knopfes erscheint eine Information, falls die Dateien erfolgreich konvertiert wurden, andernfalls erscheint eine Fehlermeldung. Auch geschieht eine Ausgabe der konvertierten Dateien auf der Konsole, um dem Benutzer einen Eindruck zu geben, wie die ausgewählte Konvertierung aussieht.

Das Fenster kann durch Drücken des „Cancel“-Knopfes wieder geschlossen werden, womit der Benutzer wieder auf dem in Abbildung 1 gezeigten Fenster ankommt.

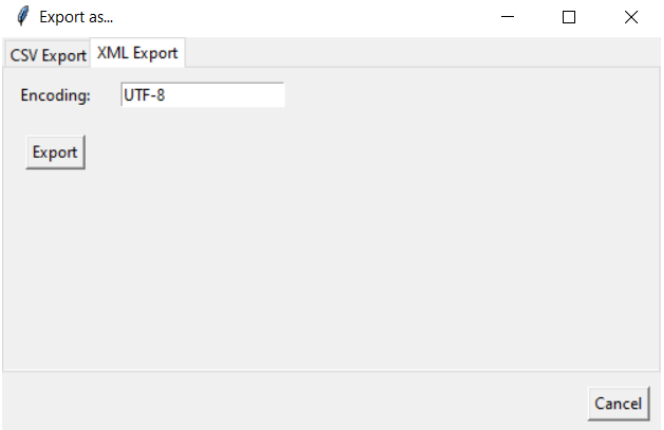

Abbildung 6: Export-Fenster bei XML Export

Der zweite der drei Knöpfe unterhalb der Tabelle öffnet ein weiteres Fenster. Dieses Fenster ist jedoch dafür zuständig, die ausgewählten Dateien in eine CSV oder XML Datei exportieren zu lassen.

Das Fenster besitzt zwei Tabs, einer für den CSV Export und einer für den XML Export. Dies ist zu sehen in Abbildung 5 bzw. 6.

Wenn die Dateien als CSV Datei exportiert werden, lassen sich Encoding, das Trennzeichen, das Zeichen für die Quotierungen und das Zeichen, welches für das Starten einer neuen Zeile benutzt werden sollen anpassen.

Bei einem Export als XML Datei lässt sich nur das Encoding anpassen.

Alle Parameter sind mit Standardwerten versehen, um die Benutzung zu vereinfachen und dem Benutzer Vorschläge zu geben, mit welchen Parameterwerten die höchste Chance besteht die Dateien erfolgreich zu exportieren.

Der Export geschieht in beiden Tabs durch Drücken des Export-Knopfes.

Bei erfolgreichem Export wird eine Information ausgegeben, dass der Export gelungen ist, andernfalls erscheint eine Fehlermeldung.

Durch Drücken des Cancel-Knopfes wird das Fenster geschlossen und der Benutzer sieht wieder das bei Abbildung 1 dargestellte Bild.

In der Klasse GUI sind noch weitere Funktionen im Code unterhalb der Definitionen der Widgets , die in den Fenstern zu sehen sind. Diese Funktionen werden ausgeführt, wenn der Benutzer mit den Widgets interagiert, zum Beispiel wird die Funktion setFileEncoding() ausgeführt, wenn der Benutzer das Encoding einer ausgewählten CSV Datei ändert. Jedoch ist die Klasse GUI einzig und allein für die Darstellung der Informationen zuständig und nicht für die Funktionalität, dass, in diesem Beispiel, das Encoding einer Datei geändert wird. Dies würde gegen das „model-view-separation-principle“ verstoßen, das postuliert, dass jegliche Implementierungslogik strikt von der Darstellung getrennt wird, um Fehlern vorzubeugen.

Um trotzdem die Implementierungslogik erreichen zu können, damit die Dateien geändert werden können wird eine Klasse benötigt, die zwischen der Implementierung und der GUI kommuniziert.

Hierfür ist die Klasse reader_and_gui_interface zuständig, von der die Klasse GUI abgeleitet ist, um die Funktionen von reader_and_gui_interface aufrufbar zu haben.

1.1.2 KLASSE READER_AND_GUI_INTERFACE

Die Klasse reader_and_gui_interface funktioniert nach dem Controller-Pattern, das heißt sie kontrolliert den Kommunikationsaustausch zwischen Darstellung und Implementierung, ohne gegen das „model-view-separation-principle“ zu verstoßen.

Hierfür besitzt die Klasse eigene Funktionen, die eine Funktion in der Implementierung aufrufen und die nötigen Informationen anfordern, die sie dann wiederum an die GUI weitergeben, damit diese sie darstellen kann.

Ein Beispiel: Der Benutzer ändert das Encoding einer CSV Datei durch die dafür bereitgestellt Textbox im GUI-Fenster. Durch Bestätigen der Eingabe mit „Enter“ wird die Funktion setFileEncoding() der GUI Klasse aufgerufen, die wiederum die Funktion setUserEncoding() für die ausgewählte CSV Datei aufruft. setUserEncoding() kommuniziert dann mit den Klassen reader und dataframeAndHeaderHandler, die die Datei verändern, hier zum Beispiel das Encoding auf den vom Benutzer eingegebenen Wert setzen. Die veränderte Datei wird von reader_and_gui_interface gelesen und an die Klasse GUI weitergegeben, die diese Datei dann verändert dem Nutzer präsentiert. Dieses Verfahren wird für alle Widgets in der GUI angewandt.

Somit wird sichergestellt, dass nicht gegen das „model-view-separation-principle“ verstoßen wird, wobei die Klasse GUI die „view“ repräsentiert und die Klassen reader und dataframeAndHeaderHandler das „model“, also die Implementierung, indem die Klasse reader_and_gui_interface als Controller-Klasse zwischen den beiden Bereichen fungiert und eine Kommunikation ermöglicht.

1.1.3 KLASSE READER

Die Klasse reader stellt mit der Klasse dataframeAndHeaderHandler die eigentliche Implementierung der Funktionalität der GUI dar, und ist somit die Hauptklasse des „model“ des „model-view-separation-principle“. Sie stellt die Funktionen bereit, um CSV und XML Dateien einzulesen und zu verändern, die dann von der Klasse reader_and_gui_interface aufgerufen werden, um den Benutzer die Dateien anpassen zu lassen.

Die Klasse legt zuerst für jede Instanz von ihr ein Dictionary an, in dem die absoluten Dateipfade der gelesenen Dateien als Schlüssel gespeichert werden. Als korrespondierende Werte werden die Parameter gesetzt, wobei dies auch wieder als Dictionary aufgebaut ist. Somit ist der Wert des Schlüssels der Dateipfade wiederrum ein Dictionary, in dem die Parameter mit ihren Werten gespeichert sind

```
{'C:/Users/Philip/Desktop/testfiles_20201221/airtravel.csv': {'hasHeader': False, 'Encoding': 'ascii', 'Delimiter': ',', 'QuoteChar': '"', 'skipInitSpace': True,
 'lineTerminator': None, 'Quoting': 0}, 'C:/Users/Philip/Desktop/testfiles_20201221/export.xml': {'sep': "',", 'q': '\'"\'', 'lf': "'\n'", 'xsl_file': 'C:/Users
/Philip/Desktop/testfiles_20201221/xml2csv.xsl', 'parameters_len': 3, 'hasHeader': True, 'init': False}}
```

Abbildung 7: Datei-Dictionary mit einer eingelesenen CSV und einer XML Datei als Ausgabe auf der Konsole

Wie in Abbildung 7 zu sehen, werden die Parameter je nach Dateityp angepasst. Bei CSV Dateien gibt es ein allgemeines Parameter-Dictionary, das als Wert gesetzt wird, bei XML Dateien hängt es von dem, vom Benutzer, ausgewählten XSL-Stylesheet ab, der die Parameter beinhaltet. Die Parameter im XSL-Stylesheet werden ausgelesen und nachträglich als Werte gesetzt.

Danach initialisiert die Klasse einen Integer-Wert, der multiple_files_counter genannt wird. Dieser Integer zeigt an ob eine Datei bereits eingelesen wurde und fügt den Wert dieser Zahl dann an den Dateipfad an, um die Dateien in der GUI und im, in Abbildung 7 beschriebenen, Dictionary unterscheiden zu können.

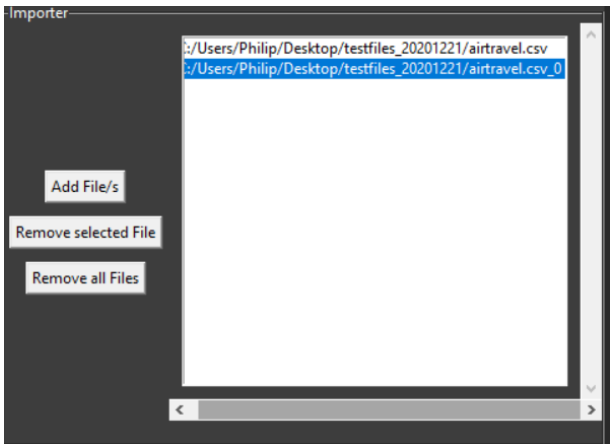

Abbildung 8: Beispiel einer doppelt eingelesenen Datei

Als nächstes wird ein Dataframe des pandas-Moduls, mit dem Namen main_dataframe angelegt. Dieses Dataframe speichert alle eingelesenen Dateien in Tabellenform ab, um die Dateien in der GUI, in der dafür bereitgestellten Tabelle, zu präsentieren. Dieses Dataframe wird bei jeder Parameteränderung aktualisiert, um dem Benutzer stets den aktuellsten Stand zu zeigen.

Das nächste Attribut der Klasse ist wieder ein Integer-Wert mit dem Namen column_count. Dieser Integer zählt, durch eine von pandas bereitgestellt Methode, die Spalten von main_dataframe und vergleicht diese Spaltenanzahl mit der Spaltenanzahl der neu eingelesenen Dateien. Falls diese nicht übereinstimmen wird die Datei verworfen und der Benutzer wird darüber in einer Meldung informiert.

Zuletzt wird eine Instanz der Klasse dataframeAndHeaderHandler angelegt, um bestimmte Funktionen auszulagern. Diese Klasse wird im Folgenden erklärt.

1.1.4 KLASSE DATAFRAMEANDHEADERHANDLER

dataframeAndHeaderHandler() ist eine Klasse die zur Implementierung und damit auch wie die Klasse reader zum „model“ gehört. Im Gegensatz zu reader() stellt sie nur Hilfsfunktionen bereit, die ausgelagert wurden, um reader() übersichtlicher zu machen.

Hier ein kurzer Überblick über die Funktionen:

ImportFile(): nimmt die Datei, die schon durch reader zu einem Dataframe konvertiert wurde, und fügt sie an main_dataframe an. main_dataframe wird auch in dieser Klasse als Attribut initialisiert, um die Bezeichnung einheitlich zu halten. Während des Prozesses des Anfügens werden die Spaltennamen von main_dataframe und der neu eingelesenen Datei verglichen. Wenn die neue Datei Spaltennamen, also eine Kopfzeile, hat wird diese als neue Kopfzeile des main_dataframe genommen, wenn nicht bleibt die Kopfzeile von main_dataframe. Falls keines der beiden Dataframes eine Kopfzeile besitzt, wird eine Standard Kopfzeile durch find_header_formats() erzeugt und übernommen.

find_header_formats(): liest die ersten fünf Zeilen des übergebenen Dataframes (meist main_dataframe) ein und versucht mithilfe von regex_list_filler() und regex_tester() eine Standard Kopfzeile zu bilden, wobei die Zeilen an regex_list_filler() übergeben werden, das eine Liste der gefundenen Typen der Einträge der Zeile zurückgibt. Die fünf übergebenen Zeilen, die nun mit Typenbezeichnungen gefüllt sind, werden miteinander verglichen, um herauszufinden, ob die Typen der Zeilen gleich sind. Falls sie gleich sind wird eine der Zeilen als Kopfzeile verwendet, falls sie nicht gleich sind werden die unterschiedlichen Einträge durch die Typenbezeichnung „String“ ersetzt und dann als Kopfzeile verwendet.

regex_list_filler(): iteriert über jeden Eintrag von der von find_header_formats() übergebenen Liste und testet für jeden Eintrag mithilfe von regex_tester() den Typ des Eintrags. Die Einträge der übergebenen Liste werden durch die herausgefundenen Typen ersetzt und die Liste wird wieder an find_header_formats() zurückgegeben.

```
self.types_dict : dict = {
                "Integer": re.compile(r"^\d+$"),
                "Float": re.compile(r"^\d+(\.|\,)\d+$"),
                "Coordinate": re.compile(r"^(N|S)?0*\d{1,3}°0*\d{1,3}(’|')0*\d{1,3}\.\d*(”|\")(?(1)|(N|S)) (E|
                "Email": re.compile(r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"),
                "Time": re.compile(r"^([0-1]\d|2[0-3]):[0-5]\d(:[0-5]\d)*$"),
                "Date": re.compile(r"^(?:(?:31(\/|-|\.)(?:0?[13578]|1[02]))\1|(?:(?:29|30)(\/|-|\.)(?:0?[13-9]
                "Datetime": re.compile(r"^(?:(?:31(\/|-|\.)(?:0?[13578]|1[02]))\1|(?:(?:29|30)(\/|-|\.)(?:0?[1
                "Web-URL": re.compile(r"^((ftp|http|https):\/\/)?(www\.)?(ftp|http|https|www\.)?([a-zA-Z0-9_-]
                "Bool": re.compile(r"^(wahr|falsch|ja|nein|yes|no|true|false|t|f|1|0)$", flags=re.IGNORECASE)
            }
```

Abbildung 3: Ausschnitt des Dictionary, welches die regulären Ausdrücke enthält

regex_tester(): liest den String, der von regex_list_filler() übergeben wird, ein und wendet verschiedene reguläre Ausdrücke auf ihn an. Falls einer der Ausdrücke ein Match ergibt, wird der Typ, der vorher in einem Dictionary definiert wurde, als Typ des Strings zurückgegeben.

1.2 ZUSAMMENARBEIT DER KLASSEN

Die in Kapitel 1.1 genannten Klassen arbeiten nach der Model-View-Controller-Architektur zusammen. Das heißt, dass das komplette Projekt in verschiedene Teile unterteilt wird, nämlich „model“, „view“ und „controller“. Zum „model“ gehören die Klassen reader() und dataframeAndHeaderHandler(), welche die Funktionalität der GUI bereitstellen und somit auch auf sich allein gestellt die komplette Funktionalität beinhalten. So lassen sich alle Operationen, die für die GUI benötigt werden, auch durch Eingabe der Befehle auf der Konsole oder durch ein eigenes Programm aufrufen ohne dabei die GUI zu benutzen. Ein

Testprogramm mit möglichen Befehlen wurde bereitgestellt unter dem Namen „testprogram.py“, das einige Befehle beinhaltet, die einen Aufruf durch die Konsole simulieren.

Somit ist die Klasse GUI allein für die leichtere Bedienung der Implementierung für den Benutzer zuständig und zählt damit zur „view“.

Der „controller“ der Model-View-Controller-Architektur ist die Klasse reader_and_gui_interface. Sie stellt Methoden bereit, um Instanzen der Klasse GUI mit Instanzen der Klasse reader kommunizieren zu lassen, und sorgt dafür, dass die Befehle, die durch den Benutzer bei Benutzen der GUI ausgelöst wurden, die Klasse reader erreichen und eine entsprechende Antwort zurückgesendet wird.

2. BENUTZUNG UND BEDIENUNG

2.1 BENUTZUNG

Die GUI kann gestartet werden, indem der Benutzer durch den Befehl „cd“ in den Ordner „src“ des Hauptordners wechselt. Hier ist einfach der Befehl „python csv_xml_importer.py“ einzugeben, vorausgesetzt der Benutzer hat eine Installation eines Python Interpreters auf dem Rechner und dieser Interpreter ist über eine PATH Variable zu erreichen.

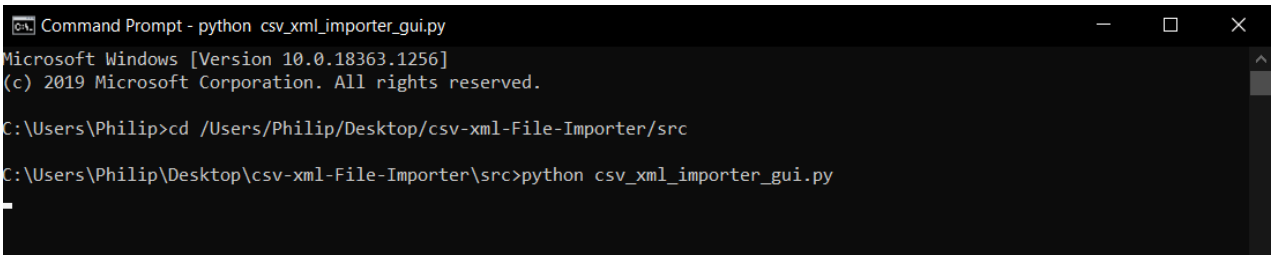

Abbildung 10: Die einzugebenden Befehle am Beispiel eines Rechners, bei dem sich der Ordner auf dem Desktop befindet

Außerdem sind noch einige Module zu installieren, falls diese nicht schon installiert wurden. Dies lässt sich mit dem Befehl „pip install name_des_moduls“ bewerkstelligen. Die zu installierenden Module lauten: pandas, pandastable, pathlib, chardet und lxml.

Nun sollte sich das Programm starten lassen und die in Abbildung 11 gezeigte GUI anzeigen. Alternativ kann die Datei csv_xml_importer_gui.py in einer Python IDE geöffnet werden und durch Anklicken eines Buttons mit der Bezeichnung „Run“ gestartet werden.

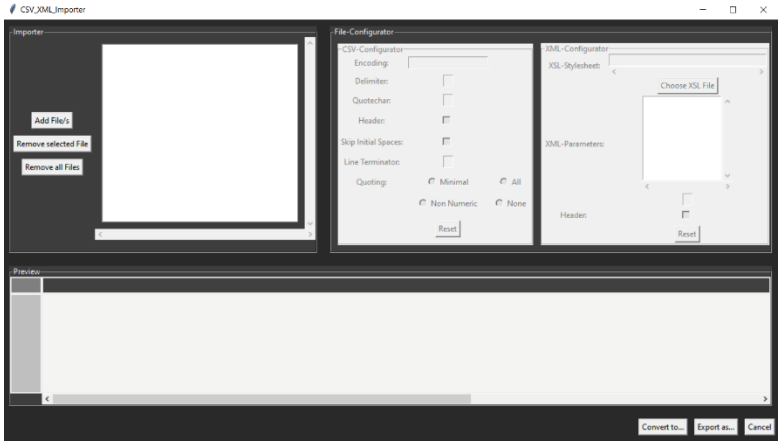

Abbildung 11: Alles außer den beschriebenen Widgets ist ausgegraut

Wie in Abbildung 11 zu sehen ist, wird der Benutzer durch die Anwendung durch die Zustände der Widgets geführt. Das heißt, dass es Widgets gibt, die ausgegraut sind und bei Anklicken nicht reagieren, und wiederrum andere Widgets, die auf Benutzereingaben reagieren und andere Bereiche freischalten, je nachdem was gewählt wurde.

Ein Beispiel: Für den Benutzer sind nach dem Start der GUI alle Widgets, außer dem Labelframe „Importer“ und den drei Knöpfen unten rechts ausgegraut und nicht responsive. Versucht der Benutzer mit anderen Widgets zu interagieren ist dies nicht möglich, da sie erst freigeschaltet werden müssen. Wenn der Benutzer jedoch in diesem Zustand versucht einen der drei unteren Knöpfe zu drücken wird entweder eine Meldung gezeigt, dass keine Dateien zum Importieren oder Exportieren vorhanden sind, wie in Abbildung 12 zu sehen ist, oder die Anwendung wird geschlossen.

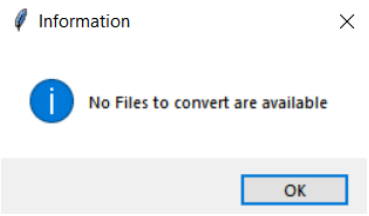

Abbildung 12: Meldung, falls der Benutzer Dateien konvertieren will, aber keine Dateien gegeben sind

Falls der Benutzer dann auf den Knopf drückt der mit „Add Files“ betitelt wurde, öffnet sich ein Datei Dialog, in dem der Benutzer eine oder mehrere CSV und XML Dateien auswählen kann, damit diese in der GUI angezeigt werden.

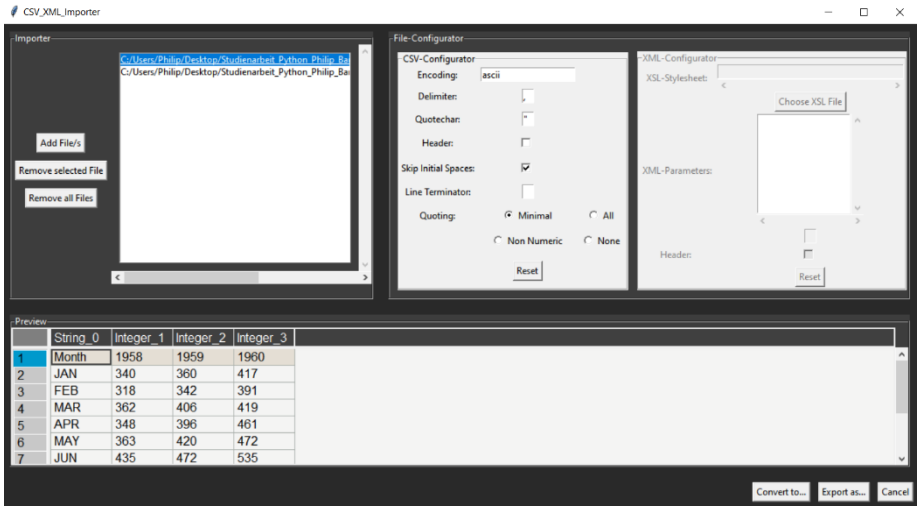

Abbildung 13: Mehrere Dateien wurden ausgewählt

Wie in Abbildung 13 zu sehen ist, wurden mehrere Dateien ausgewählt und Bereiche der GUI wurden benutzbar. Hier wurde in der Listbox, die für die Anzeige der absoluten Dateipfade der Dateien zuständig ist, die Datei „airtravel.csv“ durch Anklicken ausgewählt. Damit wird das Fenster für den CSV Konfigurator responsive und die Parameter der Dateien werden durch das Programm eingelesen und die zugehörigen Felder der GUI werden mit den Werten dieser Parameter belegt, um den Benutzer diese Werte nach Belieben ändern zu lassen. Checkboxen und Radiobuttons werden durch einfaches Anklicken geändert, wohingegen bei Textboxen, wie zum Beispiel bei „Encoding“ müssen, die Eingaben durch Drücken der Enter-Taste bestätigt werden.

In Abbildung 14 wurde nun eine XML Datei ausgewählt. Hier muss ein zusätzliches XSL-Stylesheet ausgewählt werden, um die XML Datei einlesen zu können. Dies ist möglich durch Drücken des Knopfes der „Choose XSL File“ genannt wurde. Daraufhin öffnet sich der gezeigte Datei Dialog, in welchem der Benutzer das gewünschte Stylesheet auswählt. Nach Öffnen des Stylesheet, wird der absolute Dateipfad der XSL Datei in der obigen „XSL-Stylesheet“-Textbox angezeigt und die gefundenen Parameter in diesem Stylesheet in der Listbox darunter angezeigt. Bei Auswählen eines dieser Parameter kann der Wert dessen in der, unter der Listbox liegenden, Textbox gesehen und geändert werden, wie in Abbildung 15 sichtbar.

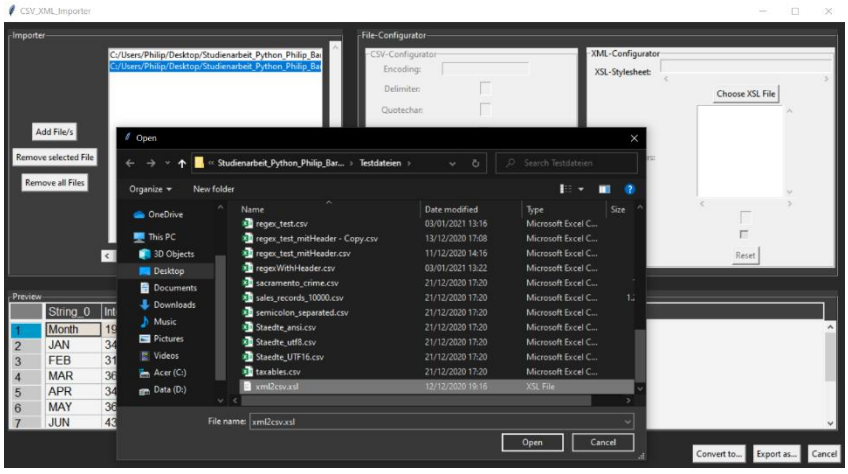

Abbildung 14: Bei Auswählen einer XML-Datei muss zuerst ein passendes XSL-Stylsheet ausgewählt werden

Für beide Dateitypen wurde noch ein Reset-Knopf eingefügt, um die Eingaben des Benutzers für diese Datei zurückzusetzen.

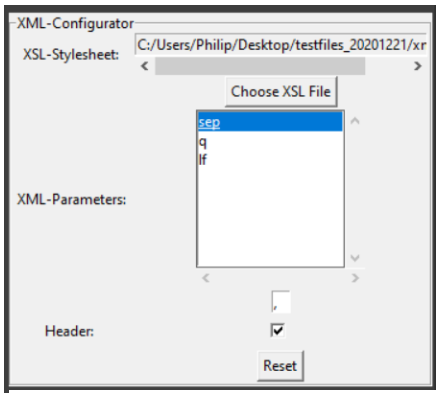

Abbildung 15: Ansicht nach Auswählen eines XSL-Stylsheets

Allgemein wurde darauf geachtet den Benutzer nicht zu sehr durch die Benutzung der GUI durch Meldungen und Informationen zu leiten, sondern durch Bereiche, die je nach Eingabe freigeschaltet werden. Auch erscheinen Meldungen und Pop-ups nur dann, wenn eine fehlerhafte Datei ausgewählt wurde, eine fehlerhafte Eingabe getätigt wurde oder bei Export und Konvertierung, dass die Dateien exportiert oder konvertiert wurden. Auch wurde versucht fehlerhafte Eingaben des Benutzers möglichst zu verbessern, damit der Benutzer nicht selbst diese Eingaben ausbessern muss. So wird zum Beispiel bei einer fehlerhaften Eingabe von dem Parameter Encoding, also zum Beispiel wurde ein Encoding angegeben, mit dem diese Datei nicht eingelesen werden kann, das Standard Encoding wieder ausgewählt, mit dem die Datei zuerst eingelesen wurde, um eine fehlerhafte Präsentation in der unteren Tabelle zu vermeiden.

2.3 TESTFÄLLE

Es wurden diverse Test Dateien verwendet, um eine möglichst fehlerfreie Benutzung der GUI zu gewährleisten. Alle Dateien, die zum Testen verwendet wurden, sind im Ordner „testfiles“ zu finden.

Im Folgenden wird ein Beispiel gegeben, um zu demonstrieren, wie Testfälle abgedeckt wurden:

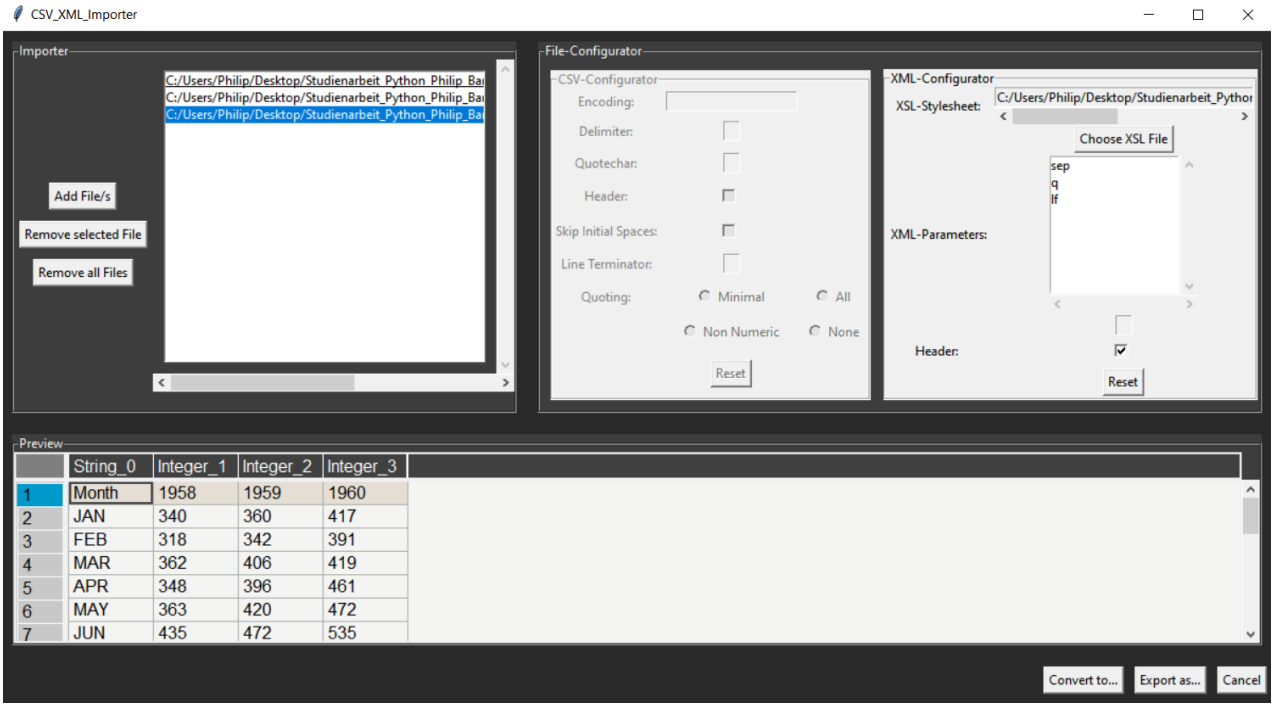

Abbildung 16: Beispiel eines Testfalls

In diesem Testfall wurde zweimal dieselbe CSV Datei eingelesen und danach eine XML Datei, die dieselbe Spaltenanzahl hat wie die vorigen CSV Dateien. Damit wurden folgende Kriterien bestätigt:

- Dieselbe CSV Datei lässt sich doppelt einlesen
- XML Dateien und CSV Dateien lassen sich zu einer Ausgabe kombinieren
- XSL-Stylesheets können angegeben werden

- Parameter von XSL-Stylesheets können ausgelesen und in der dafür vorgesehenen Listbox angezeigt werden
- Parameter von XML und CSV Dateien werden eingelesen und angezeigt
- Parameter von XML und CSV Dateien können geändert werden
- Die Tabelle im unteren Teil der GUI zeigt das erwartete Verhalten an

3. VERWENDUNG AUS EINEM ANDEREN PROGRAMM („API“)

Die Verwendung aus einem anderen Programm ist möglich, wenn sich die Datei csv_xml_importer_gui.py, reader.py und reader_support_file.py in demselben Ordner befinden, wie das Programm, das die Dateien als API verwenden soll.

Hier ein Beispiel zu Verwendung:

```
3   import csv_xml_importer_gui
4
5   gui = csv_xml_importer_gui.gui()
```

Abbildung 17: Verwendung von csv_xml_importer_gui als API

Diese Anweisungen importieren die Datei csv_xml_importer_gui.py in die aktuelle Python Datei und legen eine Instanz der Klasse GUI an, die gui genannt wird. Somit wird bei Ausführung dieses Programms die GUI gestartet.

4. QUELLEN

Anschließend werden die Quellen zu den Webseiten verlinkt, auf denen die regulären Ausdrücke getestet wurden, die zum Finden der Standard Kopfzeilen benutzt werden:

- Geo-coordinates-regex: https://regex101.com/r/Mfuc3q/1
- URL-regex: https://regex101.com/r/9BMy6h/6
- Time-regex: https://regex101.com/r/iqULdk/4
- Mail-regex: https://regex101.com/r/B5gObk/2/
- Date-regex: https://regex101.com/r/LzWH0Z/4
  Quelle: https://stackoverflow.com/questions/15491894/regex-to-validate-date-format-dd-mm-yyyy
- Datetime-regex: https://regex101.com/r/lZVgzs/3
- Int-regex: https://regex101.com/r/SayIgu/1
- Float-regex: https://regex101.com/r/0kU7GJ/1
- Bool-regex: https://regex101.com/r/6c2FRs/2