Manual

In this document we outline a procedure of how you can adapt our algorithm to your setting. It is important that you have someone in your team who can work with python and with regular expressions.

1 Install the fitz package.

We assume that you have a version of python (version 3.6 or higher) on your computer, along with the standard libraries that are required for running the script (re, os, and csv). A package that you probably need to install is fitz, which can be installed by running the command pip install PyMuPDF.

2 Specify the *inputfolder* and the *outputcsvfile* variables.

The *inputfolder* variable specifies the folder which contains the PDF files that will be used for information extraction. The *outputcsvfile* variable is the file path of the new csv file that will be output of the process and hence will contain the extracted information.

```
# Specify the folder that contains the pdf-files
inputfolder = r"C:/Users/username/Documents/pdffolder/"
# Specify the filepath of the output csv file
outputcsvfile = r"C:/Users/username/Documents/output.csv"
```

Figure 1: line number 11-14

The csv output file will contain multiple columns and rows. Each row will represent a line item for each of the tables. The first column will contain the line item names; each subsequent column will output the information that is extracted for each PDF file.

3 Specify the tables to extract from the PDF using regular expressions.

The script uses regular expressions to extract tables from the PDF files. As there is no uniform way in which all firms display their tables in their PDF files, there needs to be customization of our script in order to be able to extract the tables. However, one important condition for the script to work is that in your setting the PDF files display tables in a similar manner. That is, across the PDF files, tables must start and end in a similar way, otherwise we cannot specify a way to extract the tables in a uniform way.

In order to extract the tables, you need to extract a substring from the total text string by using the *find_tables* function. The first parameter of this function is the text string where the tables need to be extracted from. This is often the *text_concat* variable, which is a long text string that contains the content of the PDF file without newline characters. Sometimes this parameter can be another extracted table if we for example want to split the balance sheet in an asset table and a liability table, as in our script.

The second parameter of the *find_tables* function is the regular expression with a capture group that is needed to extract the table from the text_concat variable. Again, when constructing these regular expressions, please note that there are no newline characters in the *text_concat* variable.

```
# Extract the tables using regular expressions and capture groups from the text_concat string variable
text_concat_balance = find_tables(text_concat, r"(Statement\s+of\s+financial\s+position\s+(.*?)Statemen
text_concat_balance_debit = find_tables(text_concat_balance, r"(Assets(.*?)Liabilities)")
text_concat_balance_credit = find_tables(text_concat_balance, r"(Liabilities(.+))")
text_concat_pl = find_tables(text_concat, r"(Statement\s+of\s+comprehensive\s+income(.*?)Statement\s+of
text_concat_cashflow = find_tables(text_concat, r"(Statement\s+of\s+Cash\s+Flows(.+))")
```

Figure 2: line number 141-146

4 For each table, specify the line items to extract using regular expressions.

As we now have extracted our tables into concatenated text strings, we now can search within these text strings for different line items. For each table you need to create a list that contains the line items that you want to extract. More specifically, each list contains tuples. The first element of the tuple is a string value that contains the line item name. This line item name will be displayed in the first column of the output csv file. The second element of the tuple is the regular expression that is used to find the line item in the table text string.

```
profitloss_lineitems = [("P&L: Recognized net gains/(losses) on financial instruments at fair value through profit or loss",
                         ("P&L: Net interest income", r"Net\s+interest\s+income"),
                         ("P&L: Withholding tax", r"Withholding\s+tax"),
                         ("P&L: Total investment result", r"Total\s+investment\s+result"),
                         ("P&L: Subscription and redemption fee income", r"Subscription\s+and\s+redemption\s+fee\s+income"),
                         ("P&L: Other income", r"Other\s+income"),
                         ("P&L: Total other results", r"Total\s+other\s+results"),
                         ("P&L: Investment management fee", r"Investment\s+management\s+fee"),
                         ("P&L: Management fee (external)", r"Management\s+fee\s+\(external\)"),
                         ("P&L: Pricing expenses", r"Pricing\s+expenses"),
                         ("P&L: Regulatory fee", r"Regulatory\s+fee"),
                         ("P&L: Audit fee", r"Audit\s+fee"),
                         ("P&L: Depositary fee", r"Depositary\s+fee"),
                         ("P&L: Custody fee", r"Custody\s+fee"),
                         ("P&L: Other charges", r"Other\s+charges"),
                         ("P&L: Total charges", r"Total\s+charges"),
                         ("P&L: Net result attributable to holders of participations", r"Net\s+result\s+attributable\s+to\s+ho
                         ("P&L: Participation Class A", r"Participation\s+Class\s+A"),
                         ("P&L: Participation Class B", r"Participation\s+Class\s+B"),
                         ("P&L: Participation Class I", r"Participation\s+Class\s+I"),
                         ("P&L: Participation Class J", r"Participation\s+Class\s+J"),
                         ("P&L: Net result attributable to holders of participations", r"Net\s+result\s+attributable\s+to\s+ho
                         ]
```

Figure 3: line number 65-87

5 Check and potentially adapt the *pattern_integers* variable.

In order to extract amounts for each line item from the table, we need to have a regular expression pattern with capture groups that will be concatenated with the regular expression for each of the line items. Our script uses the *pattern_integers* variable to do this. For your setting, it is important to carefully examine and potentially adapt this variable. In our setting, we want to extract the first amount after the line item (which can contain digits, commas and/or the minus sign), which is not an amount that is surrounded by square brackets (which in our case indicates a reference to the notes of the financial statements).

```
pattern_integers = r"\s+(?:\[[\d]+\]\s+)?([\d\,\-]+)\s+[\d\,\-]+"
```

Figure 4: line number 37

Our script allows you to have multiple pattern variables, as not all the tables in the PDF have the same format (for example, some tables might include percentage signs) and you might want to extract multiple amounts as well.

6 Check and potentially adapt the *string_to_integer_or_float* function.

The script extracts the amounts with the *pattern_integers* variable. However, the resulting values are still string values. As end-users often want the amounts in a number format, we need to transform the string value into an integer or a float value. This is what our *string_to_integer_or_float* function does. For your setting, it is important to carefully examine and potentially adapt this variable, as your tables may contain symbols that are different from the symbols in our setting (dots instead of commas, percentage signs, etc.).

```
def string_to_integer_or_float(string):
    """ This function casts a string to an integer or a float """
    if string == "-":
        return 0
    if "," in string:
        return int(string.replace(",", ""))
    else:
        return int(string)
```

Figure 5: line numbers 17-24

Although in this function the output will always be an integer, it is of course possible to specify that, for example, if an amount contains a percentage sign, the output should be of type float.

7 Customize the *fundname* variable

The *fundname* variable will be the column name in the output csv file. In our setting, we extract the fund name from the PDF file name. However, you could also use the complete filename as your *fundname* variable.

```
# we use a regular expression to extract the fund name from the file name
fundname = re.findall(r"((.*?)fund)", filename)[0][0]
```

Figure 6: line number 124-125

8 Specify the elements in the *items* list

As a final step, you need to customize the *items* list by including several of the python objects that you created. The script will loop through all the tuples (for each table you need to create one tuple) and use the elements of the tuple to extract the amounts from the tables.

```
items = [(balance_sheet_debit, balance_debit_lineitems, text_concat_balance_debit, pattern_integers, 0),
         (balance_sheet_credit, balance_credit_lineitems, text_concat_balance_credit, pattern_integers, 0),
         (pl_account, profitloss_lineitems, text_concat_pl, pattern_integers, 0),
         (cashflow_statement, cashflow_lineitems, text_concat_cashflow, pattern_integers, 0)
         ]
```

Figure 7: line number 163-167

The first element of each tuple is an empty dictionary for each table, that will contain key-value pairs where the key is the line item, and the value is the extracted amount. You need to create these empty dictionaries in the script for each of the tables that you want to extract information from.

```
balance_sheet_debit = {}
balance_sheet_credit = {}
pl_account = {}
cashflow_statement = {}
```

Figure 8: line number 151-154

The second element is the line items list that you created in step 4. The third element is the text string that you created in step 3. The fourth element is the *pattern_integers* variable that you created in step 5. Note again that it is possible to have multiple of these variables in your script, as not all tables will have the same format. The fifth element in the tuple specifies the position of the amount to be extracted. If your pattern in step 5 allows for multiple amounts to be captured, you can specify which amount you want to be captured here. If you only want the most recent amount, this element should be a zero, as in our script.