

AutoChecker, Specification

Ricky Fan, HaoWei Chen

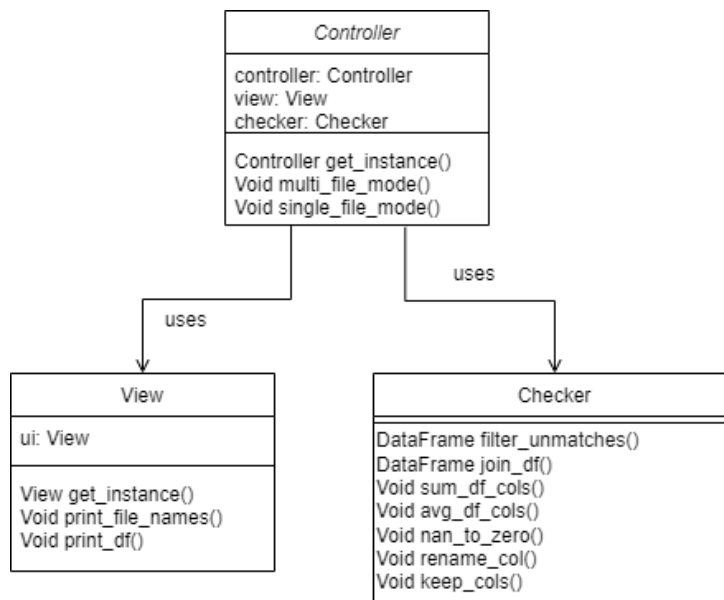
February 14, 2022

This Module Interface Specification (MIS) document contains modules, types and methods used to support the AutoChecker. The AutoChecker reads, compares and displays data from different Excel files. By specifying the Excel file paths and grouping the file information in `utils.py`, the AutoChecker reads and compares data from these files, and displays all rows with unmatching values. The AutoChecker can be launched by typing `python Demon.py` in terminal.

1 Overview of the design

This design applies Module View Specification (MVC) design pattern and Singleton design pattern. The MVC components are *Checker* (model module), *View* (view module), and *Controller* (controller module). Singleton pattern is specified and implemented for *View* and *Controller*

An UML diagram is provided below for visualizing the structure of this software architecture



The MVC design pattern are specified and implemented in the following way: the abstract object *Checker* compare, aggregate and transform the data gather from external (excel) files. A view module *View* displays prompt messages and rows with different values. The controller *Controller* is responsible for handling input actions and the work flow of the automation

For *View* and *Controller*, use the *get_instance()* method to obtain the abstract object.

Checker Module (Abstract Object)

Module

Checker

Uses

pandas

Syntax

Exported Constants

None

Exported Types

Checker = ?

Exported Access Programs

| Routine name | In | Out | Exceptions |
|--------------|--------------------------------------|-----------|------------|
| get_unmatch | DataFrame, String, String | DataFrame | |
| join_df | DataFrame, DataFrame, String, String | DataFrame | |
| diff_df | DataFrame, DataFrame, String | DataFrame | |
| sum_df_col | DataFrame | | |
| avg_df_cols | DataFrame | | |
| get_rows_sum | DataFrame, String | N | |
| find_val_idx | DataFrame, String, String | Seq of N | |
| nan_to_zero | DataFrame | | |
| rename_col | DataFrame, Map of String and String | | |
| keep_cols | DataFrame, Seq of String | | |

Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

`get_unmatch(df, col1, col2):`

- output: filter out every row in the DataFrame that has different values in the two specified columns
- exception: none

`join_df(df1, df2, key, how):`

- output: join (default inner) two DataFrames on the specified key
- exception: none

`diff_df(df1, df2, key):`

- output: create another DataFrame that contains values in df1 but not in df2 and vice versa
- exception: none

`sum_df_cols(df):`

- transition: add an additional column the DataFrame that stores the sum of every row from the specified starting index to the ending index
- exception: none

`avg_df_cols(df):`

- transition: add an additional column to the DataFrame that stores the average of every row from the specified starting index to the ending index
- exception: none

`get_rows_sum(df, header):`

- output: the sum of all rows under the specified header
- exception: none

`find_val_idx(df, header, value):`

- output: a list of indices in the specified column that contains the input value
- exception: none

`nan_to_zero(df):`

- transition: fill out all the nan cells with 0
- exception: none

`rename_col(df, name_map):`

- transition: rename the column headers in the DataFrame using the name map
- exception: none

`keep_cols(df, cols):`

- transition: drop all the columns in the DataFrame and keep the ones specified from the input
- exception: none

View Module

Module

View

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

| Routine name | In | Out | Exceptions |
|------------------|-------------------|-----|------------|
| print_file_names | Seq of String | | |
| print_df | DataFrame | | |
| out_file | DataFrame, String | | |

Semantics

Environment Variables

window: A portion of computer screen to display the messages (i.e. the terminal)

State Variables

ui: View

State Invariant

None

Assumptions

- The View constructor is called for each object instance before any other access routine is called for that object.
- The constructor can only be called once.

Access Routine Semantics

get_instance():

- transition: $ui := (ui = \text{null} \Rightarrow \text{new View}())$
- output: *self*
- exception: none

print_file_names(f_names):

- transition: window := Displays a sequence of file names

print_df(df):

- transition: window := Displays the DataFrame

print_df(df, file_name):

- transition: writes the input DataFrame to the specified file

Local Function:

`__init__`: $\text{void} \rightarrow \text{View}$

`__init__()` \equiv `new View()`

Controller Module

Controller Module

Uses

Checker, View, pandas

Syntax

Exported Types

None

Exported Constants

None

Exported Access Programs

| Routine name | In | Out | Exceptions |
|-----------------|---------------------------------------------------------------|------------|------------|
| get_instance | Checker, View | Controller | |
| single_col_mode | Pair of String and Map, Pair of String and Map, Boolean | | |
| multi_col_mode | Map of String and Map, Pair of String and Map | | |

Semantics

Environment Variables

None

State Variables

view: View

controller: Controller

checker: Checker

State Invariant

None

Assumptions

- The Controller constructor is called for each object instance before any other access routine is called for that object.
- The constructor can only be called once.
- Assume that the view instances are already initialized before calling Controller constructor

Access Routine Semantics

`get_instance(c, v):`

- transition: `controller := (controller = null \Rightarrow new Controller (c, v))`
- output: *self*
- exception: None

`single_col_mode(file1, file2, check_idx):`

- transition: operational method

```
f1_pfx := file1[1]["prefix"]  
f2_pfx := file2[1]["prefix"]  
f1_hd := file1[1]["header"]  
f2_hd := file2[1]["header"]  
df1 := get_df_col(file1[0],file1[1])  
checker.rename_col(df1, f1_pfx, "prefix")  
df2 := get_df_col(file2[0], file2[1])  
checker.rename_col(df2, f2_pfx, "prefix")  
dfj := checker.join_df(df1, df2, "prefix")  
checker.nan_to_zero(dfj)  
file_name = file1[0] + file2[0]
```

```

check_idx  $\Rightarrow$  process_df(df1, df2, dfj, file_name) | !check_idx  $\Rightarrow$  df_umatches :=
checker.get_unmatch(dfj, f1_hd, f2_hd)  $\wedge$  view.out_file(df_umatches, file_name)
dfd := checker.diff_df(df1, df2, "prefix");
view.out_file(dfd, file_name)

```

- output: none

multi_col_mode(f_map, file2):

- transition: operational method

```

use the data form f_map to generate a DataFrame
aggregate the DataFrame from f_map by summing or averaging the columns
drop the extra columns from the f_map DataFrame
use the data form file2 to generate a DataFrame
compare the two DataFrames and display all the unmatching rows

```

- output: none

Local Function:

init: View \rightarrow Controller

init(checker, view) \equiv new Controller(checker, view)

process_df: DataFrame \times DataFrame \times DataFrame \times String \rightarrow void

process_df(dfj, df1, df2, file_name) \equiv (r : Tuple of \mathbb{N} and Series | $r \in$ dfj.iterrows() |
process_util($r[1]$, df1, df2, file_name))

process_util: Series \times DataFrame \times DataFrame \times String \rightarrow void

process_util(row, df1, df2, f_name):

```
idx1 := checker.find_val_idx(df1, 'prefix', row['prefix'])
```

```
idx2 := checker.find_val_idx(df2, 'prefix', row['prefix'])
```

```
idx1  $\neq$  idx2  $\Rightarrow$  view.out_file(row, f_name, idx1, idx2) | idx1 = idx2  $\Rightarrow$  (row[1]  $\neq$  row[2]
 $\Rightarrow$  view.out_file(row, f_name))
```