

# AutoChecker, Specification

Ricky Fan, HaoWei Chen

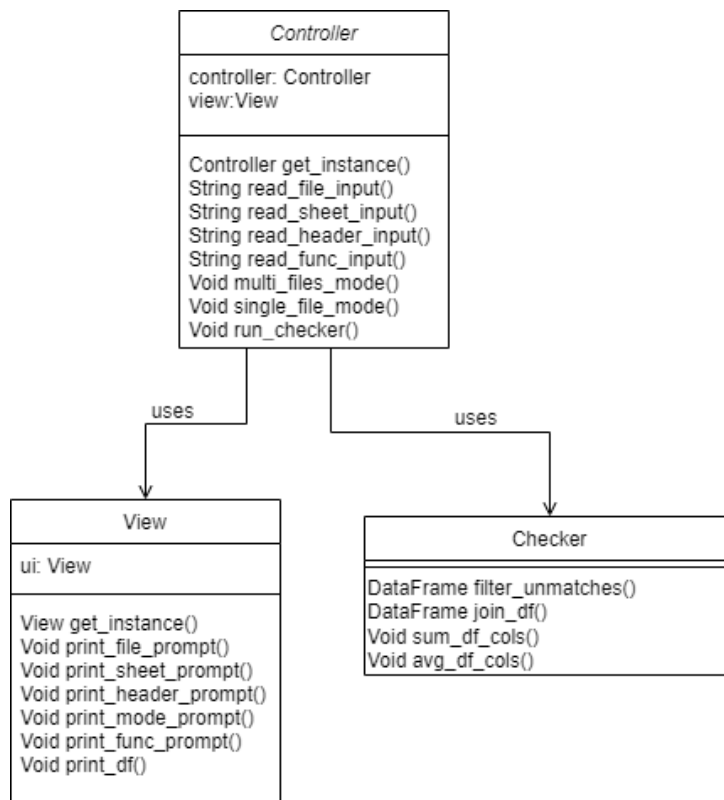
February 1, 2022

This Module Interface Specification (MIS) document contains modules, types and methods used to support the AutoChecker. The AutoChecker start by prompting the user to select a checker mode (single file vs multi-files). If the single file mode is selected, the user will be prompted to enter two file directories, their corresponding sheets and column headers. Then the checker will output a list of rows with different values. If the multi-files mode is selected, then user will be prompted to enter multiple (default 5) file directories, their corresponding sheets and column headers. Later, the user will be prompted to enter another file directory and its corresponding sheet and column header. Then, the checker will use the data from these files to output a list of rows with different values **make demo** 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 control 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_unmatches	DataFrame, String, String	DataFrame	
join_df	DataFrame, DataFrame, String	DataFrame	
sum_df_cols	DataFrame, $\mathbb{N}$ , $\mathbb{N}$ , String		
avg_df_cols	DataFrame, $\mathbb{N}$ , $\mathbb{N}$ , String		

## Semantics

### State Variables

None

### State Invariant

None

### Assumptions

None

## Access Routine Semantics

`get_unmatches(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):`

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

`sum_df_cols(df, start_idx, end_idx, col_name):`

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

`avg_df_cols(df, start_idx, end_idx, col_name):`

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

# View Module

## Module

View

## Uses

None

## Syntax

### Exported Constants

None

### Exported Types

None

### Exported Access Programs

Routine name	In	Out	Exceptions
get_instance		View	
print_file_prompt			
print_sheet_prompt			
print_header_prompt			
print_mode_prompt			
print_func_prompt			
print_df	DataFrame		

## 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 = null  $\Rightarrow$  new View())`
- output: *self*
- exception: none

`print_file_prompt()`:

- transition: `window := Displays a prompt message asking the user to enter a file directory`

`print_sheet_prompt()`:

- transition: `window := Displays a prompt message asking the user to enter a sheet name`

`print_header_prompt()`:

- transition: `window := Displays a prompt message asking the user to enter a header name`

`print_mode_prompt()`:

- transition: `window := Displays a prompt message asking the user to select a checker mode`

`print_func_prompt()`:

- transition: `window := Displays a prompt message asking the user to select a function`

`print_df(df)`:

- transition: `window := Displays the DataFrame`

**Local Function:**

`__init__`: `void`  $\rightarrow$  `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	View	Controller	
read_file_input		String	
read_sheet_input		String	
read_header_input		String	
read_func_input		String	
multi_file_mode	Map of String and String Pair of String and Map		
single_file_mode	Pair of String and Map Pair of String and Map		
run_checker			

### Semantics

#### Environment Variables

None

#### State Variables

view: View

controller: Controller



## 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( $v$ ):

- transition:  $\text{controller} := (\text{controller} = \text{null} \Rightarrow \text{new Controller } (v))$
- output: *self*
- exception: None

read\_file\_input():

- output: *input* : String, file directory entered by the User
- exception: none

read\_sheet\_input():

- output: *input* : String, sheet name entered by the User
- exception: none

read\_header\_input():

- output: *input* : String, column header entered by the User
- exception: none

read\_mode\_input():

- output: *input* : String, mode selected by the User

- exception: none

read\_func\_input():

- output: *input* : String, function selected by the User
- exception: none

single\_file\_mode(file1, file2):

- transition: operational method

```
df1 := get_df_col(file1[0], file1[1])
df2 := get_df_col(file2[0], file2[1])
dfj := Checker.join_df(df1, df2, 'prefix')
df_unmatches := Checker.get_unmatches(dfj, file1[1]['header'], file2[1]['header'])
view.print_df(df_unmatches)
```

- output: none

multi\_files\_mode(f\_map, file2):

- transition: operational method

```
f_arr := [f:String | f ∈ f_map.keys() : get_df_col(f, f_map[f]) ]
df_mult := f_arr[0]
df_mult := (Checker.join_df(df_join, f_arr[i]) DataFrame, i : ℕ | i ∈ {1..|f_arr| - 1} :
f_arr[i])
view.print_func_prompt()
input_func := read_func_input()
input_func = 'sum' ⇒ Checker.sum_df_cols(df_mult, 1, |f_arr| - 1, input_func) |
input_func = 'avg' ⇒ Checker.avg_df_cols(df_mult, 1, |f_arr| - 1, input_func)
df2 := get_df_col(file2[0], file2[1])
df_join := Checker.join_df(df_mult, df2, 'prefix')
df_unmatches := Checker.get_unmatches(df_join, input_func, file2[1]['header'])
view.print_df(df_unmatches)
```

- output: none

`run_checker()`:

- transition: operational method for running the game.  
Start by prompting the user to select the checker mode (single file vs multi files)
  - If single file mode is selected:  
`f1 := ()`  
`f2 := ()`  
`populate_pair(f1)`  
`populate_pair(f2)`  
`single_file_mode(f1, f2)`
  - If multi files mode is selected:  
`f_map := {}`  
`f1 := ()`  
`populate_pair(f1)`  
`inputs := get_inputs()`  
`f_map[inputs[0]] = {'sheet':inputs[1], 'header':inputs[2]}`  
populate `f_map` by repeating step 3 - 4 five times  
`multi_files_mode(f_map, f1)`
- output: None

### Local Function:

`__init__`: View  $\rightarrow$  Controller

`__init__(view)  $\equiv$  new Controller(view)`

`get_df_col`: String  $\times$  Map of String and String  $\rightarrow$  DataFrame

`get_df_col(file_dir, info_map)`:

`df := load_xlsx(file_dir, info_map['sheet'])`

`out := df[[prefix, info_map['header']]]`

`get_inputs`: seq of String

`get_inputs(p)`:

`view.print_file_prompt()`

`file_dir := read_file_input()`

```

view.print_sheet_input()
sheet := read_sheet_input()
view.print_header_input()
header := read_header_input()
out := [file_dir, sheet, header]

populate_pair: Pair → void
populate_pair(p):
    inputs := get_inputs()
    p[0] := inputs[0]
    p[1] := {'sheet':inputs[1], 'header':inputs[2]}

load_xlsx: String × String → DataFrame
load_xlsx(file_dir, sheet_name) ≡ pandas.read_excel(file_dir, sheet_name)

```