Developer Documentation for Mathcad Automation Software

Thornton and Tomasetti

Piero Panariello

July 22, 2021

Contents

1 Technologies Used

1. Python is the programming language of choice
2. PySimpleGUI : used to render the graphical user interface. Documentation
3. MathcadPy : wrapper written in python used to access the Mathcad api. Documentation
4. Openpyxl : used to interface with the excel documents. Documentation
5. PyInstaller: used to freeze the python program into an executable. Documentation
6. Look in ~./dist/requirements.txt to view all the dependencies
7. Look in ~./dist/info.txt for more information on how to package the application and how to re-create the python virutal environment.

2 Mathcad API

Currently, the Mathcad API supports Mathcad Prime 3.0 and above. From my testing it works best with Mathcad Prime 7.0 (the lastest version). The API documentation can be located here. You can purchase the SDK from PTC to get more information and examples, but I would recommend against it (it's $9000).

3 Datypes and Storage of Data

3.0.1 Equipment Class: (stores all the equipment from the excel file)

1. Class variables:

   - **self.items** = list()

   List of all the equipment in the excel file, stored as individual dictionaries.

   - **self.cur_index** = 0

   The current index of the equipment that the user is viewing in the GUI

   - **self.length** = 0

   Holds the length of self.items

   - **self.fields** = list()

   A list of all the elements from the header row from the excel document

   - **self.names** = list()

   A list of all the equipment names from the excel document

   - **self.inputs** = list()

   A list of all the inputs from the header row in the excel document

2. Class methods:

   - **append(self, to_append:dict)** Takes in a dict as an argument

   Appends self.items with the new equipment dictionary, appends self.names, appends self.inputs

   - **next_index(self)** No arguments

   Increments the value of self.cur_index

   - **prev_index(self)** No arguments

   Decriments the value of self.cur_index

3.0.2 Outputs Class: (stores the values of the outputs when the user decides to preview the output variables from the Mathcad file)

1. Class variables:

   - **self.items** = list()

   Follows the format: alias, [value, unit, power] ['f_p_max_output', [408.81554560308007, 'kg', 0]], ['f_p_min_output', [76.65291480057748, 'kg', 0]], ['f_p_tot_output', [76.65291480057748, 'kg', 0]], ...

2. Class methods:

   - **append(self, to_append)** Takes tuple or list argument

   Converts to_append to array and appends self.items

   - **clear(self)** Takes no arguments

   Clears self.items

   - **display(self)->list** Takes no arguments

   Returns a list of variables and values that is easier to display in the GUI. Rounds decimals to 2 digits. Ex: ['f_p_tot_output = 408.82 kg', 'f_p_min_output = 76.65 kg', ...]

4 API Details

The MathcadPy library is used as a wrapper that allows you to access all of the mathcad api endpoints from the comfort of Python. You can read more about the Mathcad API here. The API allows you to modify and change Mathcad Prime files. Despite PTC's documenation, you cannot print documents.

get_eqpt_from_xl(filepath:str)->Equipment Takes in the filepath of the input excel file and returns the **Equipment** object. This function is executed right after the choose files window is closed.

The excel table looks similar to the one below:

eqpt_name	mounting_location	project_number	tags
Anesthesia machine	Wall, Floor	1111	Medical, ICU, something
Warming Cabinet	Floor	1111	Medical
Surgical Scrub Sink	Wall	1111	Medical
Retratable Ceiling Column	Ceiling	1111	Medical

pre_generate_report(equipment:Equipment, files, generating_multiple_reports = False) This acts as a pre-fight test. It checks if the proper template is given for the equipment and passes the equipment and a uniquely generated filename to the generate_report function.

generate_report(cur_eqpt, equipment:Equipment, file_name:str, template_file:str, files, debug = False)->bool The function connects to the Mathcad API, opens the template file specific to the mounting location, updates the input values specific to the equipment, and then saves the document. If generating multiple reports, multithreading is used to speed up the process. Currently 16 threads are being used, but feel free to increase this number if the workflow demands

more throughput. This variable is called **num_threads** in the event **generate_report_for_all**. See section 8 (Improving Performance) for more details. (Events are how PySimpleGUI handles buttons being pressed. Events are checked in the main GUI loop.)

mathcad_calculate(eqpt, files, debug = False)->dict Allows the user to preview the Mathcad calculation output. It duplicates the template file into a temp file, takes the inputs and waits for the outputs to generate. It then deletes the temp file when finished. It returns a dictionary with the output values. The debug variable changes if Mathcad will display the windows being edited or not. When **debug = False**, no window is shown, when **debug = True**, windows are shown.

5 Rendering to the GUI

Choose equipment Once the user has input the excel file they want to read from, the program extracts all information in the **get_eqpt_from_xl** function and places all the equipment names in the Choose Equipment column.

Inputs Once we get the Equipment from the **get_eqpt_from_xl** function, we can then render it to the input fields in the GUI.

Outputs If the user clicks the Preview Calculation Outputs button, the inputs from the current equipment being used is sent to the template corresponding to the correct mounting location and the output fields are gathered via the **mathcad_calculate** function. The outputs are saved in the Outputs class and the information is displayed in the GUI.

Preview Images The user has the option to include preview images that correspond to the mounting locations. The images must be included in the excel document. Use the example_sheet.xlsx as a template. Images muse be .png or .jpg or .jpeg. The images are gathered from the excel sheet using the **get_images_from_xl(self, num_images:int)** function. Images are stored as binaries. When the user views a different equipment, the image corresponding to the mounting location is loaded into the Image Preview section of the GUI. Use ./dist/img_to_b64.py to convert images from .png/.jpg to binaries. It provides a python file called output.py with the binaries stored in variables. Preview images are automatically converted to binaries in **get_images_from_xl**.

6 Saving to the Database

Saving to Database The database is a .csv file which holds some important information about generated repors. The function save_to_csv is used. When the user decides to save the report to the database, they are saving the equipment name, mounting location, tags, and the generated report's unique filename. The user can choose a specific database to save to, or it will save to the default database located in the **mathcad_automation_output** directory.

The table looks like the one below:

Date	Tags	Name	Mounting Location	File Name
2021-06-30	MEDICAL	RET... COLUMN	CEILING	test.mcdx
2021-06-30	MEDICAL	RET... COLUMN	CEILING	Retractable_....mcdx
2021-06-30	MEDICAL	SUR... SINK	WALL	Surgical_....mcdx

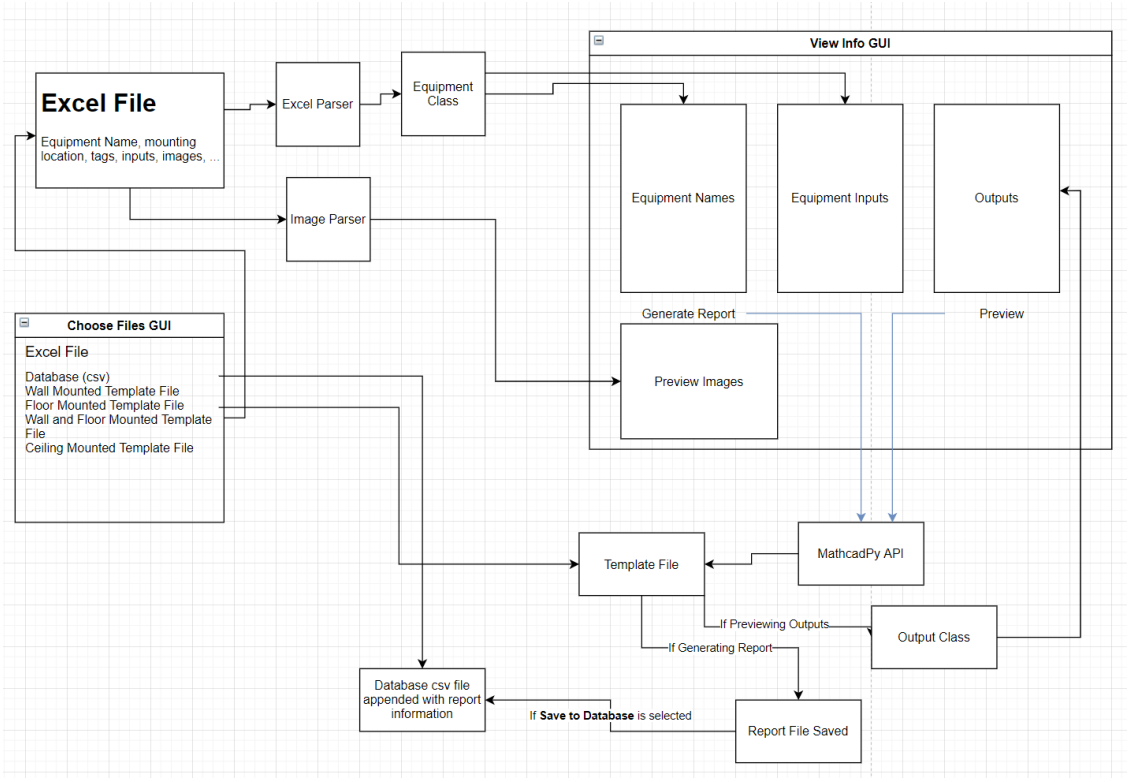

Figure 1: Flow chart of program API

7 Flowchart

8 Improving Performance

While the application is performant, there are a few ways to tune the application to improve performance. The function that is least performant is the **generate_report_for_all** event. Multiple reports need to be generated through Mathcad. To improve performance, I implemented parallel processing. This allows multiple processes to run at the same time: each report is run on a seperate process and is then joined when each process finishes. The **num_threads=16** variable allows you to adjust the number of parallel processes occuring. This number is limited to your computer's available threads. I suggest starting at **num_threads = 16** and slowly increasing the variable by powers of 2 (ex: 16, 32, 64) until you achieve the desired performance. Keep in mind that the speed in which the Mathcad API can run to generate a single report is constant. Increasing **num_threads** will only increase performance if the number of reports to be generated is greater than **num_threads**. Note that performance will not be increased if **num_threads** is greater than the number of reports to be generated.

9 Re-Creating the Virtual Environment

Typical python projects use a "virtual environment" to test and develop the application. This allows for consitancy between machines running the same program. In order to activate the virtual environment, first install **virtualenv** using pip.

```
pip install virtualenv
```

Then, create the virtual environment in the project's ./dist directory using the command below.

```
python -m venv env
```

This creates a virtual envionment called **env** in the ./dist directory.
Then, activate the virtual environment using the command below.

```
env\scripts\activate
```

Then, install all the project requirements into the virtual environment by running.

```
pip install -r requirements.txt
```

You can now make changes to main.py and test it.

10 Packaging and Distribution

I have found that PyInstaller is the best method to package python applications. It "freezes" the code in order to create an executable. Install PyInstaller using pip, and ensure that it is installed by typing

```
PyInstaller
```

in PowerShell. If it is properly installed, run the code below within the activated virtual environment to package the application. Copy the following code onto **one line** in PowerShell to run it.

```
PyInstaller --onefile --windowed
--paths= "C:\Users\Owner\Desktop\mathcad_auto\dist\main_build\MathcadPy"
 -i  "C:\Users\Owner\Desktop\mathcad_auto\dist\main_build\images\ma_logo.ico"
 --name "Mathcad_Anchorage_Automation_v1.1" main.py
```