Mathcad Automation Software

Developer Documentation – Written By: Piero Panariello

Key

Any word/phrase surrounded by asterisks (*) are literals. That is, they are to be taken exactly as the combination of characters.

Any word/phrase in `mono font` is code.

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. Source Code

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.

File Descriptions

All files are in the *./dist* directory.

1. *info.txt*
    a. This file has information about how to re-create the development environment and how to package and distribute the application. Consult the section *Packaging and Distribution* for more information.
2. *img_to_b64.py*
    a. To allow images to be displayed within the application, we cannot use traditional .jpg or .png files. We must convert them to a binary format. This allows images to be stored as strings (base64 format). This file converts .png and .jpg files into a python file containing string variables. These variables hold the information of those photos in base46 format. Run the script using *python img_to_b64.py*.
3. *images.py*
    a. This file holds the images we will be using in the application as python variables in base64 format.
4. *main.py*
    a. The application source code. Ensure the developer environment is activated and run *python main.py* to start the application.
5. *requirements.txt*

    a. Holds the name and versions of all the python packages that the program uses. You will use this file when creating the developer environment. Read more in the Re-Creating the Virtual Environment section.
6. *./main_build/MathcadPy*
    a. This is the python wrapper for Mathcad. It is directly imported into the application. Feel free to edit this file if the connection to the API is not working for some reason.
7. *./main_build/dependencies*

All the dependencies that the program uses are located here.

1. *data.py*
    a. Holds the Equipment and Output Classes. Modify this file to change the classes
2. *filestream.py*
    a. Handles any operation that deals with files on the hard disk. For example, reading from the excel input file, writing to the csv, and saving to the excel input file.
3. *gui.py*
    a. All the GUI (graphical user interface) elements that are not the main window (main.py) are declared here. This includes popup windows and the choose file window you see at startup.
4. *helpers.py*
    a. Holds all helper functions the program needs. Ex: *gen_random_string(length:int)->str*
5. *reports.py*
    a. Anything having to do with generating reports or Mathcad calculations takes place in this file.
6. *validation.py*
    a. Validates inputs given by the user. Ensures proper inputs are given, and will throw error if not.
7. *verbose.py*
    a. Handles converting strings like *w_p_input* into a more readable format. Handles converting inputs and outputs into a more verbose format.

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 latest version). The API documentation can be located here. You can purchase the SDK from Mathcad to get more information and examples, but I would recommend against it (it's $9000).

Datatypes and Storage of Data

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

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

self.outputs = list()

A list of all the outputs per equipment generated when the user clicked *Preview Calculation Outputs*.

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

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

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]],
...
```

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. Makes outputs verbose so they are easier to read. Rounds decimals to 2 digits.

```
Ex: [ 'f_p_tot_output = 408.82 kg', 'f_p_min_output = 76.65 kg', ...]
```

API Details

The MathcadPy library is used as a wrapper that allows you to access all 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 documentation, you cannot print documents. The following functions either interact with the Mathcad API or pull or save information to files.

1. *get_eqpt_from_xl(filepath:str)->Equipment*
   - Located in ./dist/main_build/dependencies/filestream.py

   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 like the one below:

eqpt_name	mounting_location	project_number	eqpt_number	tags	eqpt_tags
Anesthesia machine	Wall, Floor	1111	1234	Medical	Foo, Bar
Warming Cabinet	Floor	1111	1234	Medical	Foo
Surgical Scrub Sink	Wall	1111	1234	Medical	Bar

Retractable Ceiling Column	Ceiling	1111	1234	Medical	Foo, Bar

2. *pre_generate_report(equipment:Equipment, files, generating_multiple_reports = False)*
    - Located in ./dist/main_build/dependencies/reports.py

    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.

3. *generate_report(cur_eqpt, equipment:Equipment, file_name:str, template_file:str, files, debug = False)->bool*
    - Located in *./dist/main_build/dependencies/reports.py*

    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.

4. *mathcad_calculate(eqpt, files, debug = False)->dict*
    - Located in *./dist/main_build/dependencies/reports.py*

    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. *def save_eqpt_to_xl(equipment: Equipment, filepath:str)->bool*
    - Located in *./dist/main_build/dependencies/filestream.py*

    Saves the inputs the user has changed in the application back to the excel file. Works similarly *get_eqpt_from_xl* but in reverse. Cannot save to the excel file if it is open.

6. *class DebugLogger()*
    - Located in *./dist/main_build/dependencies/filestream.py*

    Saves user actions using the *self.log(to_log:str)->bool* function to a .debug file in the top-level directory of the application. This .debug file can be shared with the developer using the *Share* button in the GUI. The .debug file is capped at 50kb to improve performance. Once it reaches 50kb, half of the information in the file is discarded.

Rendering to the GUI

1. *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.

2. *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.

3. *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, the category of the output is decided *verbose.py/is_asd_output* and *verbose.py/is_lfrd_output*, converted to verbose names, and the information is displayed in the GUI.

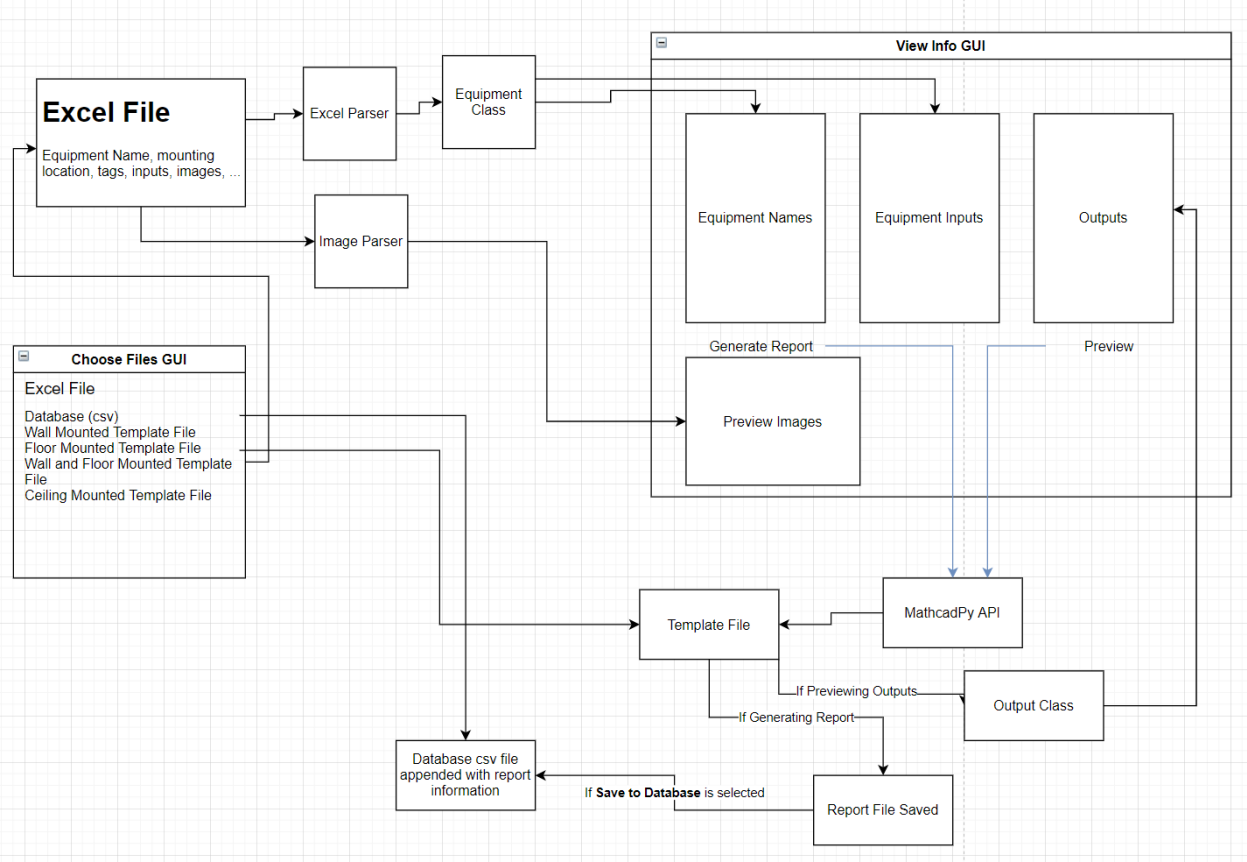
4. *Preview Images*

   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*.

Saving to the 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.

Flowchart



Re-Creating the Developer Environment

Typical python projects use a "virtual environment" to test and develop the application. This allows for consistency between machines running the same program. To activate the virtual environment, first install *virtualenv* using pip. Open PowerShell and type the following commands (make sure you have python3 installed first):

Install virtualenv

```
pip install virtualenv
```

Create the virtual environment in the project's ./dist directory

```
python -m venv env
```

Activate the virtual environment

```
env\scripts\activate
```

Install all the requirements into the virtual environment

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

You can now make changes to the application in *main.py* and run the application with those new changes. If you are creating updates, please create a new git branch and create a pull request when finished.

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 and 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
```

This will create a new executable in the *./dist/dist* directory.