MSOrganiser

Developer Documentation

Version 0.0.2

2020

# Version Control Table

|  |  |
| --- | --- |
| **Title** | MSOrganiser User Documentation |
| **Created By** | Jeremy John Selva |
| **Date Created** | 26 February 2020 |
| **Maintained By** | Jeremy John Selva |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version Number** | **Modified By** | **Date Modified** | **Modifications Made** | **Status** |
| 0.0.2 | Jeremy John | 7 Oct 2021 | Added   * 8.3 Code Coverage | Draft |
| 0.0.1 | Jeremy John | 26 Feb 2020 | Initial commit and review | Draft |

# Introduction

## Purpose

This document highlights the MSOrganiser code structure as well as the procedures required to convert the codes into an executable file.

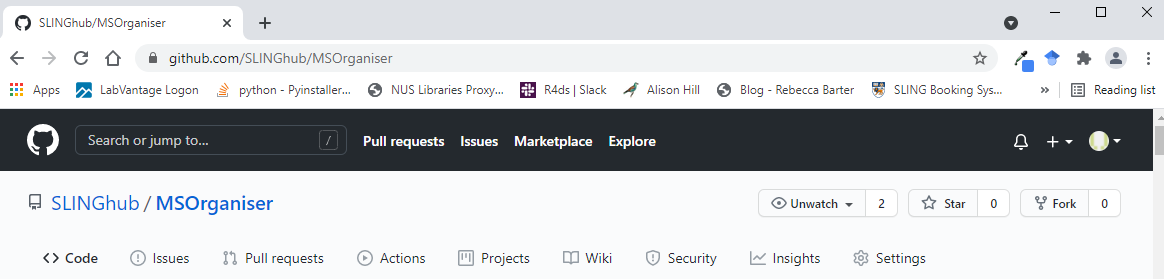
## Scope

This guide will assist developers to understand the organisation of the code structure so that changes in the feature of the software can be done efficiently. It also provide instructions on how to install the software dependencies and the steps needed to convert it into a working executable file. This document assumes you are using VS 2019 to install the python packages and create your virtual environment.

# Code Repository

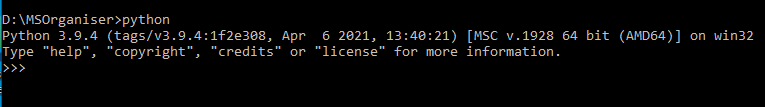
The codes can be found in <https://github.com/SLINGhub/MSOrganiser>

Click on the source tab to see



# Python Setup

MSOrganiser is run using Python 3.9.4



# Python Packages Dependencies

The python packages (with version number) required are as follows:

|  |  |
| --- | --- |
| **Python Package Name** | **Version Number** |
| alabaster | 0.7.12 |
| altgraph | 0.17.2 |
| Babel | 2.9.1 |
| cairocffi | 1.2.0 |
| CairoSVG | 2.5.2 |
| certifi | 2021.5.30 |
| cffi | 1.14.6 |
| chardet | 4.0.0 |
| colorama | 0.4.4 |
| cssselect2 | 0.4.1 |
| defusedxml | 0.7.1 |
| docutils | 0.17.1 |
| et-xmlfile | 1.1.0 |
| future | 0.18.2 |
| Gooey | 1.0.8.1 (**changes required**) |
| html5lib | 1.1 |
| idna | 3.2 |
| imagesize | 1.2.0 |
| importlib-metedata | 4.8.1 |
| install | 1.3.4 |
| jdcal | 1.4.1 |
| Jinja2 | 3.0.1 |
| macholib | 1.15.2 |
| MarkupSafe | 2.0.1 |
| numpy | 1.21.2 |
| olefile | 0.46 |
| openpyxl | 3.0.9 (**changes required**) |
| packaging | 21 |
| pandas | 1.3.3 |
| pdfrw | 0.4 |
| pefile | 2021.9.13 |
| Pillow | 8.3.2 |
| pip | 21.2.4 |
| pockets | 0.9.1 |
| psutil | 5.8.0 |
| pycparser | 2.20 |
| Pygments | 2.10.0 |
| pygtrie | 2.4.2 |
| Pyinstaller-hooks-contrib | 2021.3 |
| PyInstaller | 4.5.1 (**changes required**) |
| pyparsing | 2.4.7 |
| Pyphen | 0.11.0 |
| pypiwin32 | 223 |
| Pypubsub | 4.0.3 |
| PyQt5-Qt5 | 5.15.2 |
| PyQt5-sip | 12.9.0 |
| PyQt5 | 5.15.4 |
| python-dateutil | 2.8.2 |
| pytz | 2021.1 |
| pywin32-ctypes | 0.2.0 |
| pywin32 | 301 |
| requests | 2.26.1 |
| setuptools | 58.1.0 |
| sip | 6.1.1 |
| six | 1.16.0 |
| snowballstemmer | 2.1.0 |
| sphinx-rtd-theme | 1.0.0 |
| Sphinx | 4.2.0 |
| sphinxcontrib-applehelp | 1.0.2 |
| sphinxcontrib-devhelp | 1.0.2 |
| sphinxcontrib-htmlhelp | 1.0.3 |
| sphinxcontrib-jsmath | 1.0.1 |
| sphinxcontrib-qthelp | 1.0.3 |
| sphinxcontrib-serializinghtml | 1.1.4 |
| sphinxcontrib-websupport | 1.2.4 |
| tabulate | 0.8.9 |
| tinycss2 | 1.1.0 (changes required on build spec) |
| toml | 0.10.2 |
| typing-extensions | 3.10.1.2 |
| urllib3 | 1.26.7 |
| WeasyPrint | 52.5 (**changes required)** |
| webencodings | 0.5.1 |
| wheel | 0.37.0 |
| wxPython | 4.1.1 |
| XlsxWriter | 3.0.1 |
| zipp | 3.5.1 |

# GTK3 Version to get cairo

Cairo is required by the python package cairocffi, cariosvg and weasyprint. See “Weasyprint’s cairo dependencies”

More information about cairo can be found in <https://www.cairographics.org/>

For Windows, we install GTK3 instead as it contains the cairo software

The version used is



Downloaded from <https://github.com/tschoonj/GTK-for-Windows-Runtime-Environment-Installer>

<https://github.com/Kozea/WeasyPrint/issues/1292>

In the future, Weasyprint (version 53 onwards) may not depend on Cairo.

# Difficult python package installation

There are some python package that can be quite difficult to install due to its numerous dependencies.

## Weasyprint

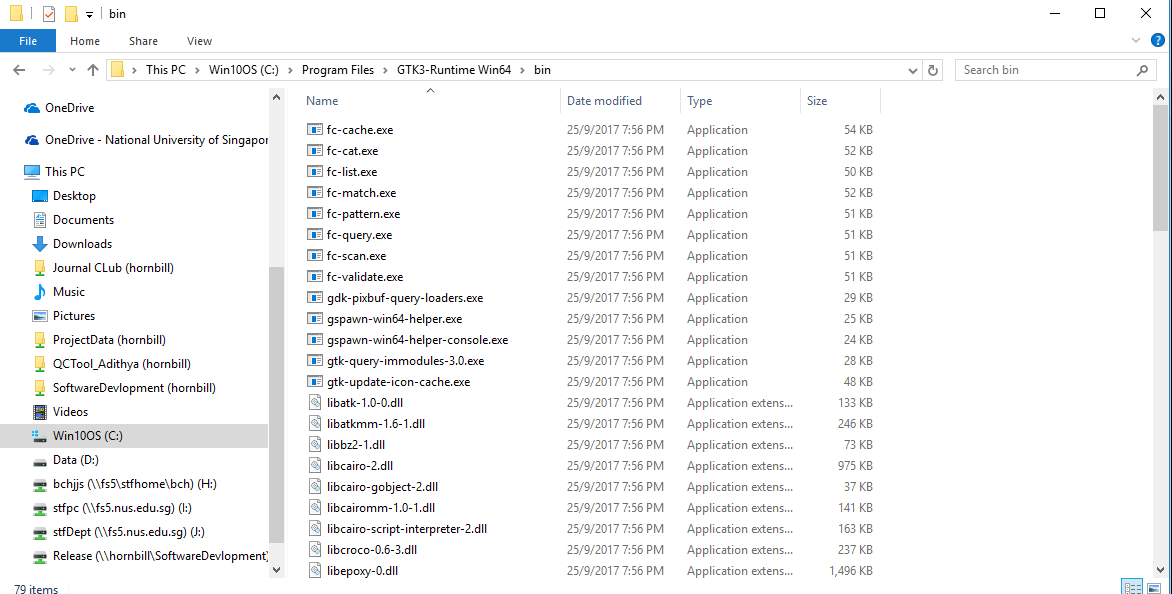
A clear instruction can be given in <http://weasyprint.readthedocs.io/en/stable/install.html>

Before installing the Weasyprint package, you must do the following after installing python and created your virtual environment

### Installing the right GTK

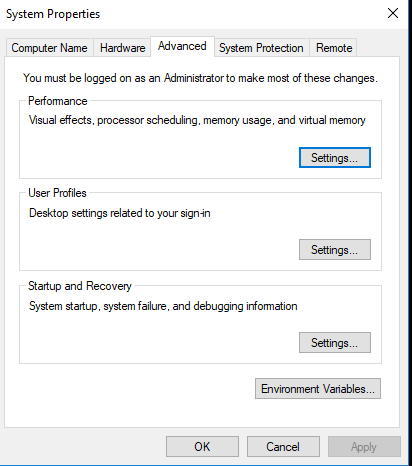
Install the win64 bit version of GTK, given in documentation <https://github.com/tschoonj/GTK-for-Windows-Runtime-Environment-Installer>

After installation, the content should



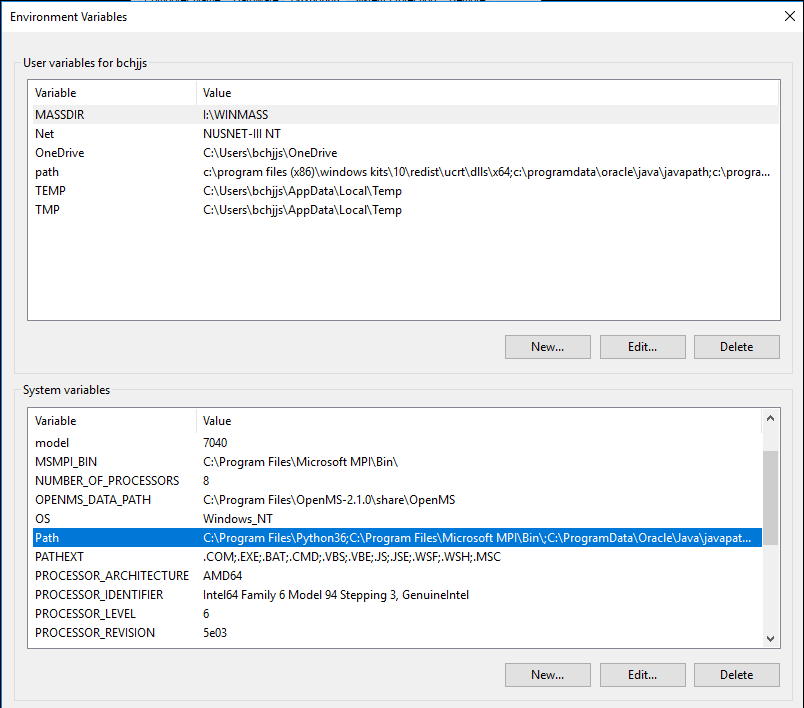
Set the environment variable "PATH" to include the bin folder

On search, go to view advanced system settings to open this window

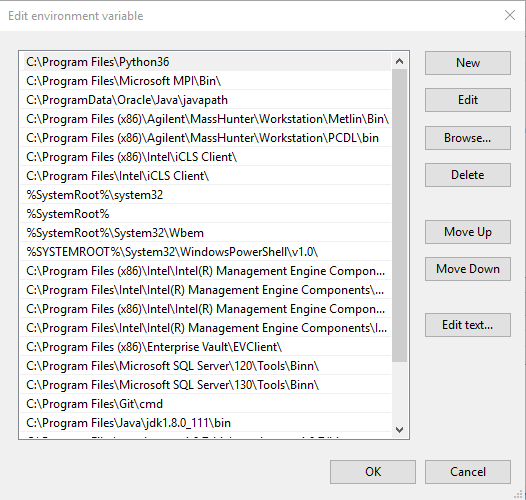


Click on Environment Variables

Go to System Variables -> Path -> Click Edit



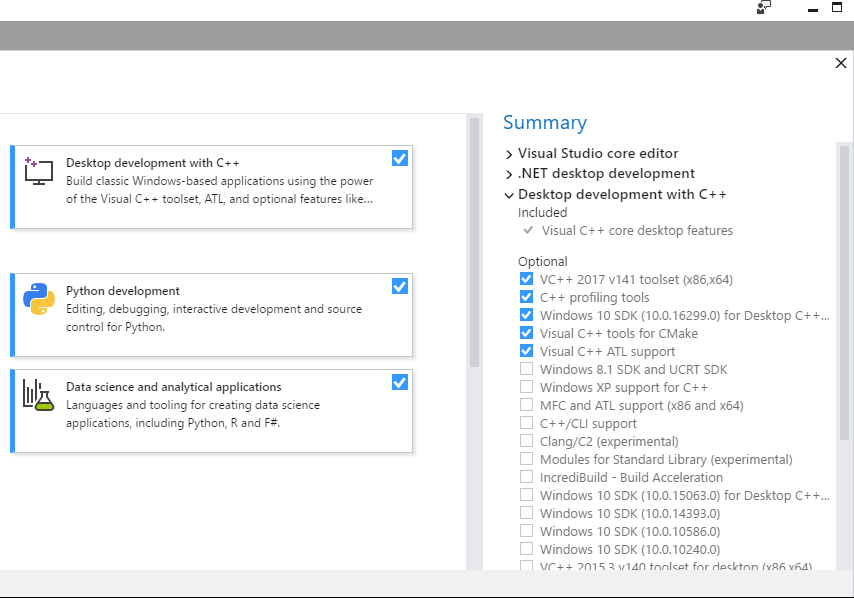
This window should appear



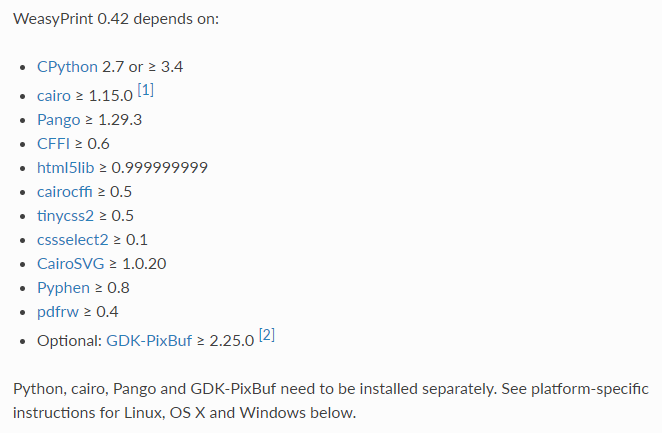
Click on New to add the file path of GTK bin file, it should be C:\Program Files\GTK3-Runtime Win64\bin

You may need to reboot your computer for GTK file pathway to be fixed in the PATH environmental variable.

### Installing a C++ compiler in VS2017 (not applicable to VS2019)



You may wish to install its python package dependencies first before installing Weasyprint



With these steps, you should be able to install weasyprint using pip without errors.



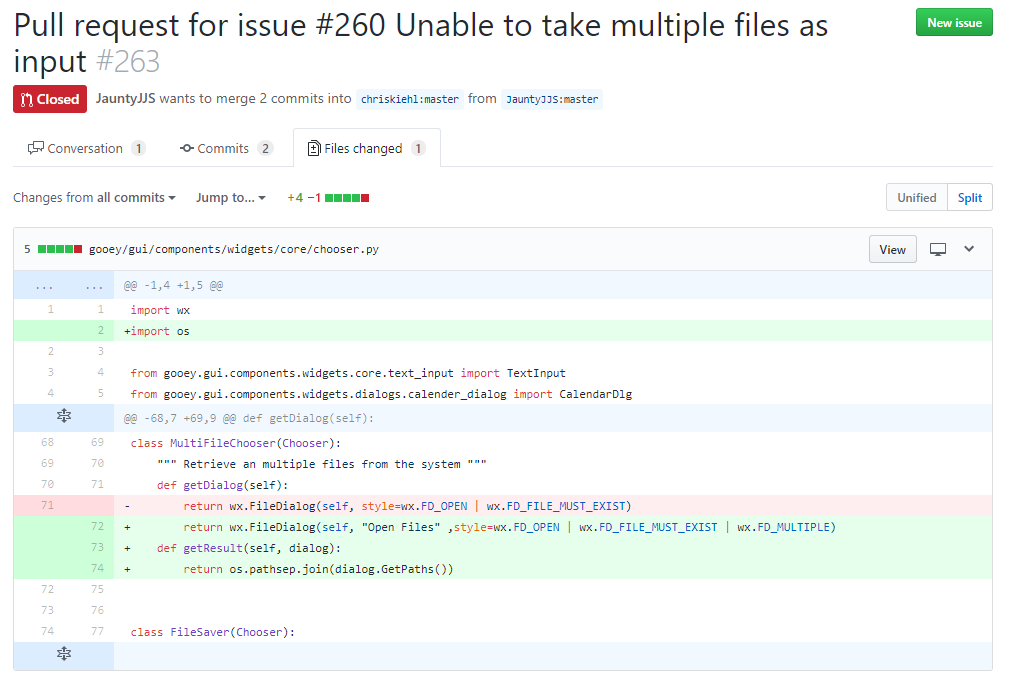
# Changes of codes in the python package after installation.

Unfortunately, even when the packages are installed. Not all of them are free from bugs. Below are the list of correction that needs to be done on the source code of the package for the application MSOrganiser to run successfully.

## Gooey (Fixed at later versions)

Unfortunately, Gooey at version 1.0.0 is unable to accept multiple files as input due to a bug in the code. This is required for MSOrganiser to accept multiple files as input.

To fix this bug, refer to <https://github.com/chriskiehl/Gooey/pull/263/files>



## Pyinstaller (Fixed at later version)

Recently, pyinstaller is unable to transfer all the relevant modules from a recent version of pandas.

See these websites for more info.

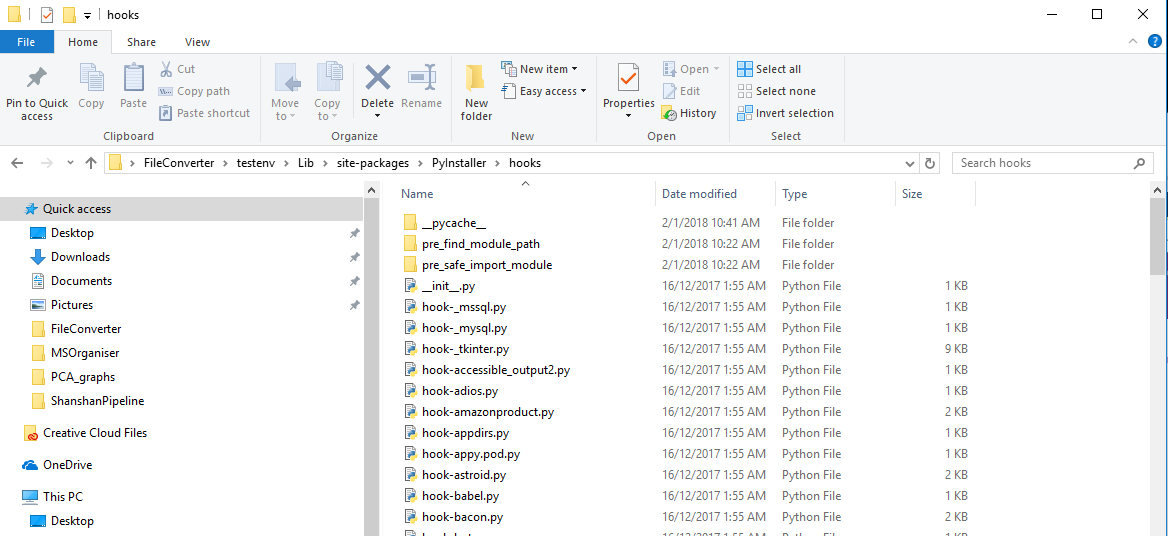
<https://stackoverflow.com/questions/47318119/no-module-named-pandas-libs-tslibs-timedeltas-pyinstaller-pandas>

<https://github.com/pyinstaller/pyinstaller/issues/2999>

<https://stackoverflow.com/questions/37815371/pyinstaller-failed-to-execute-script-pyi-rth-pkgres-and-missing-packages>

One way to resolve the issue is to install the development version of Pyinstaller as indicated in <https://stackoverflow.com/questions/37815371/pyinstaller-failed-to-execute-script-pyi-rth-pkgres-and-missing-packages>

Another way to resolve this issue is to go to the folder of python or your virtual environment, locate the pyinstaller folder ..\hooks



Create a file called hook-pandas.py with contents

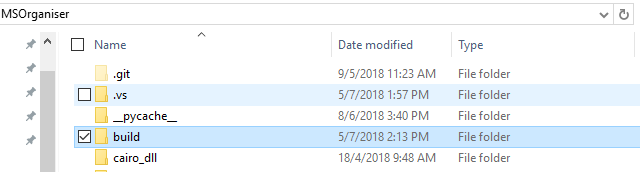
hiddenimports = ['pandas.\_libs.tslibs.timedeltas',

'pandas.\_libs.tslibs.np\_datetime',

'pandas.\_libs.tslibs.nattype',

'pandas.\_libs.skiplist']

Delete the build folder if it exists to ensure pyinstaller rebuild the exe file from scratch

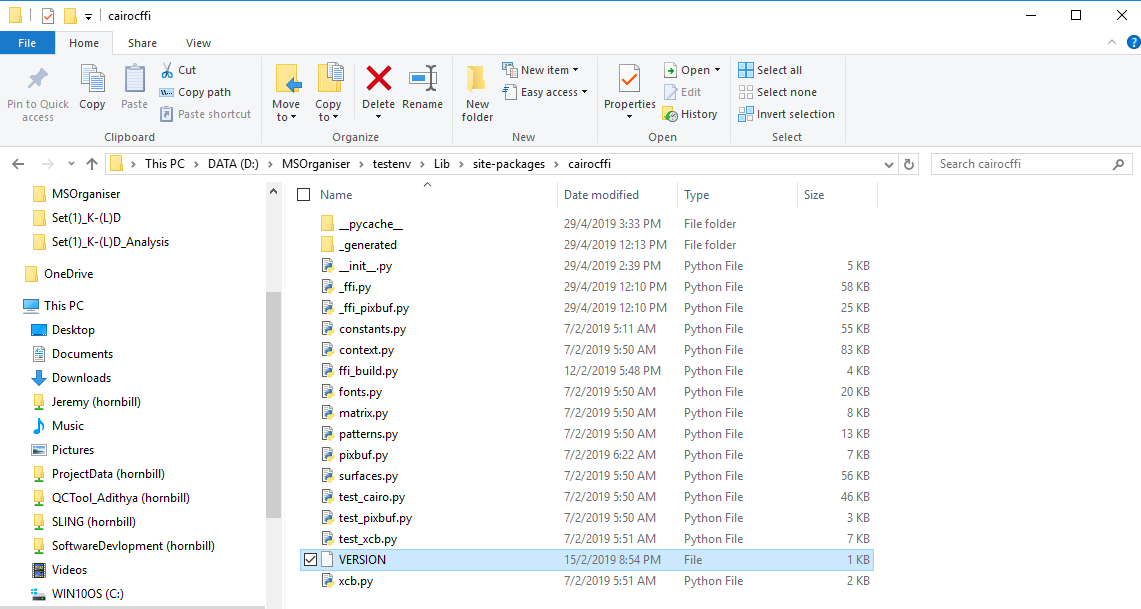


Rerun pyinstaller on your python script to convert it into an exe file. This fix should work as long as pyinstaller in not upgrade or reinstalled.

## Packages that have VERSION files that Pyinstaller does not copy

So far this applies for the cairosvg package only.

Currently, newly installed versions of these python packages involve the VERSION file.



The \_\_init\_\_.py in the installed python package has the following code which makes use of the VERSION file



The problem is that Pyinstaller does not copy the VERSION file for some packages when it is creating the exe file. The code above however, assumes the VERSION file in in the same directory as the exe file. You will need to do the following:

You will need to import these in the \_\_init\_\_.py of your problematic package

import os

import sys

Next, do the following but change the highlight text to the name of your package.

if hasattr(sys, 'frozen'):

if hasattr(sys, '\_MEIPASS'):

# Frozen with PyInstaller

# See https://github.com/Kozea/WeasyPrint/pull/540

ROOT = Path(sys.\_MEIPASS,'weasyprint')

else:

# Frozen with something else (py2exe, etc.)

# See https://github.com/Kozea/WeasyPrint/pull/269

ROOT = os.path.dirname(sys.executable)

else:

ROOT = Path(os.path.dirname(\_\_file\_\_))

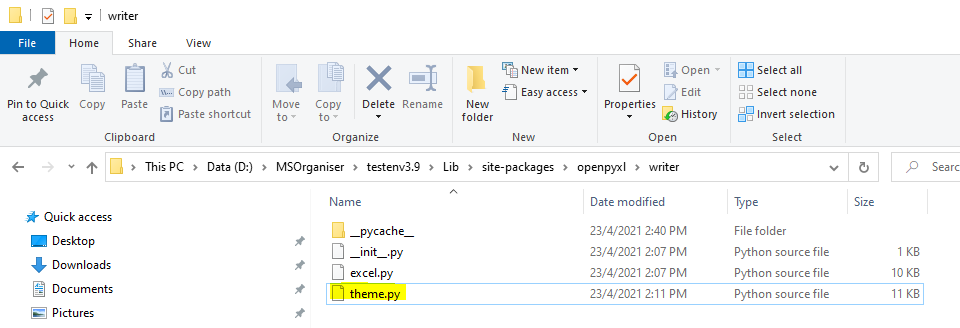
VERSION = \_\_version\_\_ = (ROOT / 'VERSION').read\_text().strip()

#VERSION = \_\_version\_\_ = (Path(\_\_file\_\_).parent / 'VERSION').read\_text()

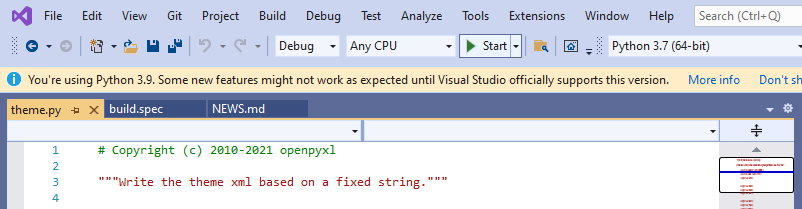
## Openpyxl change output excel to Consolas

By default, the excel file default cell font is Arial. Unfortunately, openpyxl do not have a function to change all cells in the sheet to Consolas in a very quick way. To change to Consolas, we have to modify the script in the virtual environment.

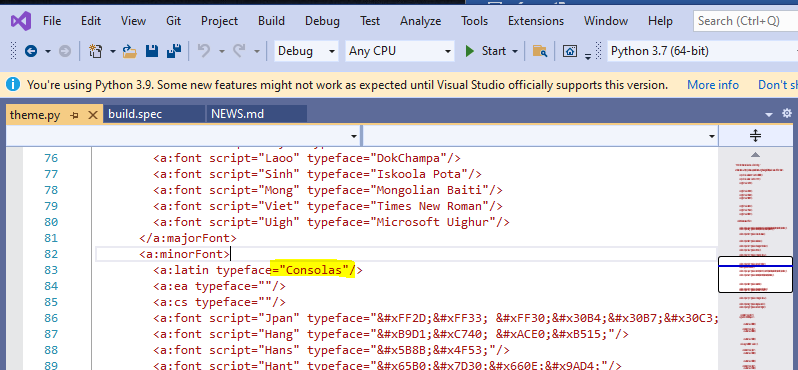
This is how it is done. In the openpyxl package folder in the virtual environment, go to the folder writer and open the file theme.py



You will see this

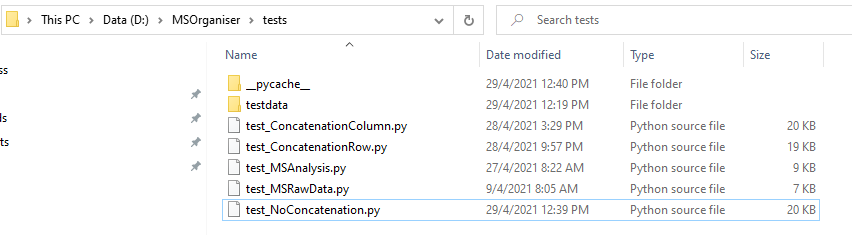


Go to the line that says “minorfont” and change the latin typeface to Consolas



# Unit Testing

Unit Test are available in the tests folder



## Run all test

To run all these test in command line, cd to the tests folder and type “python –m unittest”



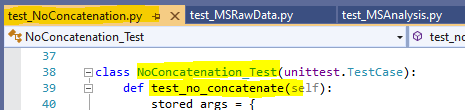


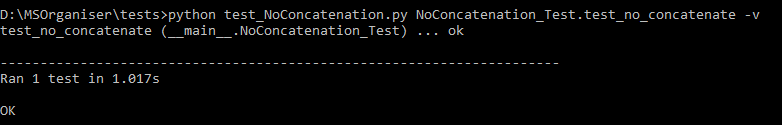
## Run specific test

To run all test in a test file via command line, cd to the tests folder and type “python [name of test unit file]”



To run a test in a test file via command line, cd to the tests folder and type “python [name of test unit file] [Name of class the rest belongs in].[test function name]”

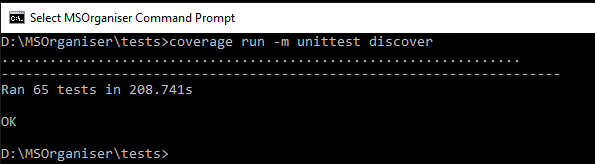




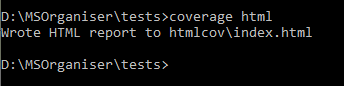
## Code Coverage

The python Coverage is used <https://coverage.readthedocs.io/> to show which part of your code are being used in your test units and which are not.

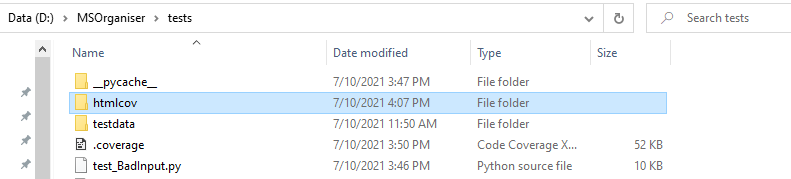
After installing Coverage and ensuring all the unit test pass, run the following command



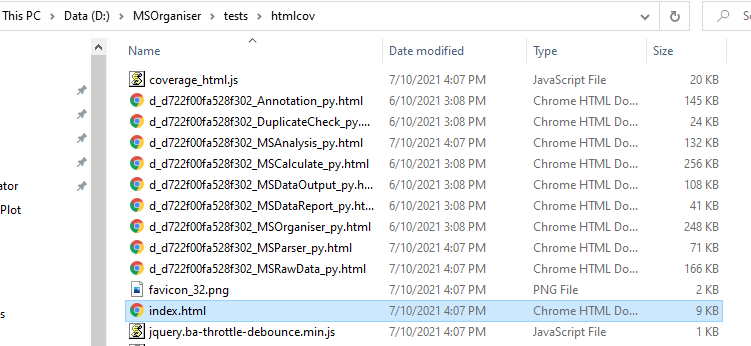
Next run

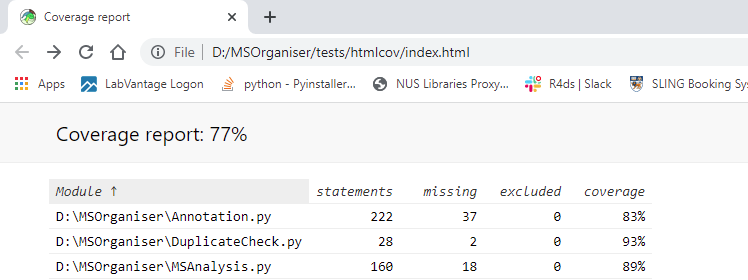


The html report can be found in the test folder



Double click on the index.html file and you will see the following





# Requirements before using Pyinstaller

Ensure that these steps are done before converting the python code into a standalone executable file.

## Folders that needs to be present and specified in the build.spec file

### VERSION file dependencies

So far this applies for the cairocffi and cairosvg package only.

These packages require the VERSION file stated in 7.3

These files however are not copied by Pyinstaller by default.

To tell Pyinstaller that these files are needed. Write in the build.spec file, the following to import these packages successfully.

import tinycss2

def \_\_is\_frozen():

return getattr(sys, 'frozen', False)

#This is to fixed the error of not able to find cairo dll files

#C:\Users\bchjjs\AppData\Local\Temp

def \_get\_report\_dir(dir\_name):

if \_\_is\_frozen():

# MEIPASS explanation:

# https://pythonhosted.org/PyInstaller/#run-time-operation

basedir = getattr(sys, '\_MEIPASS', None)

if not basedir:

basedir = os.path.dirname(sys.executable)

resource\_dir = os.path.join(basedir, dir\_name)

if not os.path.isdir(resource\_dir):

raise IOError(

("Cannot locate MSreport resources. It seems that the program was frozen, "

"but resource files were not copied into directory of the executable "

"file. Please copy `msreport` folders into `{}{}` directory.".format(resource\_dir, os.sep)))

return resource\_dir

else:

resource\_dir = os.path.dirname('\_\_file\_\_')

return os.path.join(resource\_dir, dir\_name)

os.environ['PATH'] = \_get\_report\_dir('cairo\_dll') + os.pathsep + os.environ['PATH']

#To remove the @font-face not available in Windows warning

with warnings.catch\_warnings():

warnings.filterwarnings("ignore", category=UserWarning)

import cairocffi

import cairosvg

Next, get the root path

cairocffi\_root = os.path.dirname(cairocffi.\_\_file\_\_)

cairosvg\_root = os.path.dirname(cairosvg.\_\_file\_\_)

tinycss2\_root = os.path.dirname(tinycss2.\_\_file\_\_)

To specify that the VERSION file must be copied

a = Analysis(['MSOrganiser.py'],

pathex=['.\\testenv\\Scripts'],

hiddenimports=[],

hookspath=None,

runtime\_hooks=None,

binaries=[],

datas=[ (os.path.join(cairocffi\_root, 'VERSION'), 'cairocffi'),

(os.path.join(cairosvg\_root, 'VERSION'), 'cairosvg'),

(os.path.join(tinycss2\_root, 'VERSION'), 'tinycss2')

],

)

exe = EXE(pyz,

a.scripts,

a.binaries,

a.zipfiles,

a.datas,

... (other necessary inputs)

\*Weasyprint VERSION file will be automatically copied by Pyinstaller and need not be specified in the build.spec file

### Weasyprint’s pyphen dependencies

Weayprint depends on the files in /site-packages/pyphen/dictionaries

These files however are not copied by Pyinstaller by default.

To tell Pyinstaller that these files are needed. Write in the build.spec file

import pyphen  
pyphen\_root = os.path.dirname(pyphen.\_\_file\_\_)  
pyphen\_dictionaries = Tree(os.path.join(pyphen\_root, 'dictionaries'), prefix = 'pyphen/dictionaries')

and put the variable pyphen\_dictionary into

exe = EXE(pyz,  
 pyphen\_dictionaries,  
 ... (other necessary inputs)

### Weasyprint’s cairo dependencies

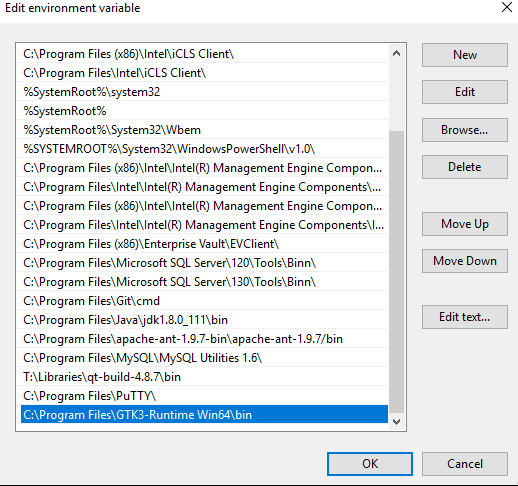
Initially we add the GTK bin folder (it should be C:\Program Files\GTK3-Runtime Win64\bin) to the PATH environment variable.

The problem is that these dll files are not included by pyinstaller and you cannot distribute this exe files unless the other computer has the same program installed.

To prevent this issue, you must dynamically change the PATH environment variable to point to these dll files when the program is running

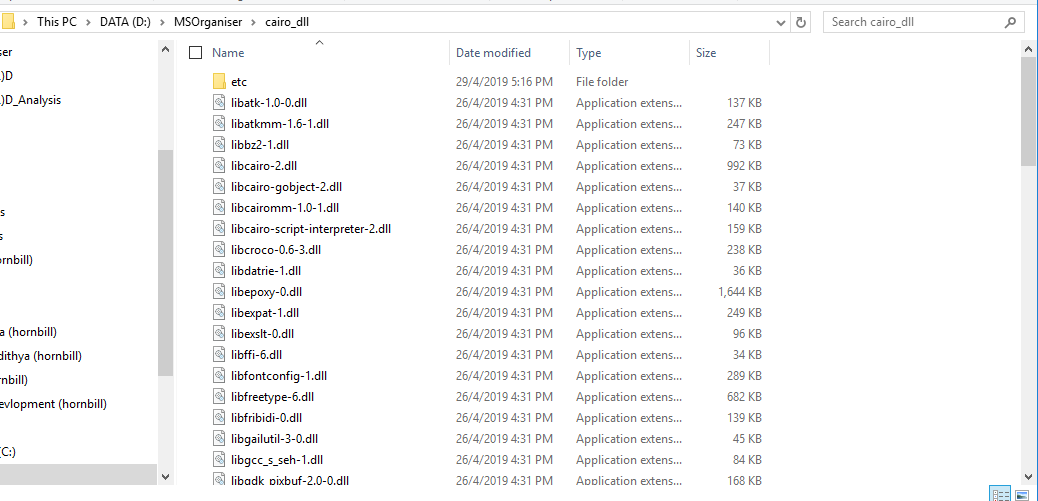
First, remove the path of the GTK bin folder in the global PATH environment variable. This is done by clicking on the path

C:\Program Files\GTK3-Runtime Win64\bin and then click delete



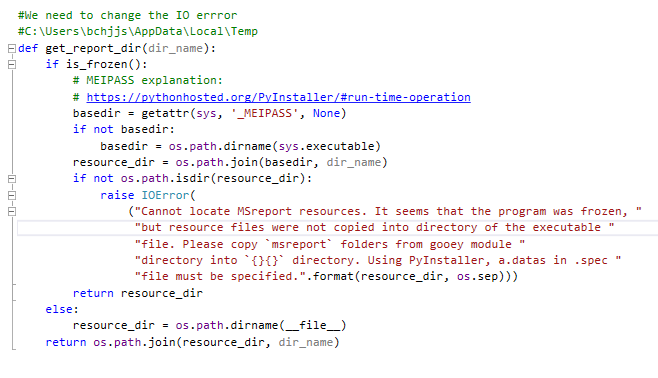
This is to ensure that the program works because the script updates the PATH variable by itself.

Copy all the dll files in C:\Program Files\GTK3-Runtime Win64\bin and the etc folder into a new folder. I called it "cairo\_dll"

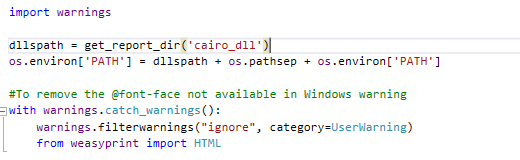


Place this folder in the same directory as the script MSOrganiser.py

In your MSDataReport.py python script, make sure this function is available

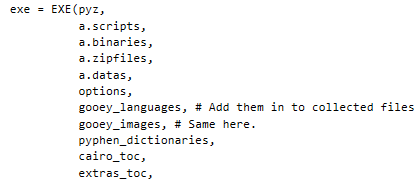


Add the dll folder path to the PATH environmental variable before using any modules in weasyprint



Finally, there is a need to update the spec file





### Gooey’s languages and images dependencies

See <http://chriskiehl.com/article/packaging-gooey-with-pyinstaller/> for more info

Gooey depends on files in /site-package/gooey/images and /site-package/gooey/images/languages

These files however are not copied by Pyinstaller by default.

To tell Pyinstaller that these files are needed. Write in the build.spec file

import gooey  
gooey\_root = os.path.dirname(gooey.\_\_file\_\_)  
gooey\_languages = Tree(os.path.join(gooey\_root, 'languages'), prefix = 'pyphen/languages')

gooey\_images = Tree(os.path.join(gooey\_root, ' images '), prefix = 'pyphen/images ')

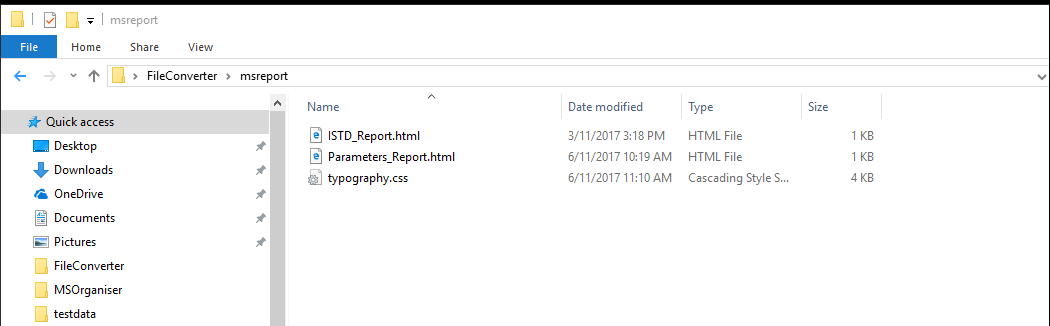
and put the variables gooey\_languages and gooey\_images into

exe = EXE(pyz,  
 gooey\_languages,

gooey\_images,  
 ... (other necessary inputs)

### MSOrganiser’s own file dependencies

The program depends on some html and css files in the folder msreport



To ensure that pyinstaller takes these files into consideration, update the spec file

extras\_toc = Tree(os.path.join(os.getcwd(), ' msreport '), prefix = 'msreport')

and put the variables extras\_toc into

exe = EXE(pyz,

extras\_toc,  
 ... (other necessary inputs)

To ensure that the script, knows where to look for these file as a standalone exe file, use the function “get\_report\_dir”



### Version Number

There is a need to create a version.txt file with the following details

VSVersionInfo(

ffi=FixedFileInfo(

filevers=(0, 0, 1, 0),

prodvers=(0, 0, 1, 0),

mask=0x3f,

flags=0x0,

OS=0x40004,

fileType=0x1,

subtype=0x0,

date=(0, 0)

),

kids=[

StringFileInfo(

[

StringTable(

u'040904b0',

[StringStruct(u'CompanyName', u'Singapore Lipidomics Incubator'),

StringStruct(u'FileDescription', u'MSOrganiser'),

StringStruct(u'FileVersion', u'0.0.1.0'),

StringStruct(u'LegalCopyright', u'Singapore Lipidomics Incubator'),

StringStruct(u'ProductName', u'MSOrganiser'),

StringStruct(u'ProductVersion', u'0.0.1.0')])

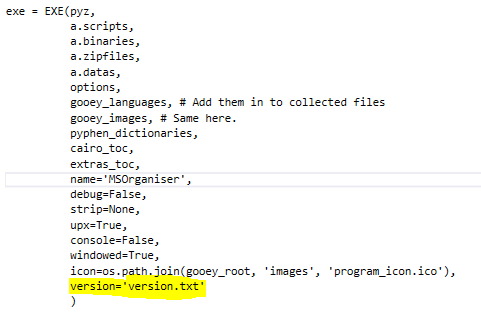
]),

VarFileInfo([VarStruct(u'Translation', [1033, 1200])])

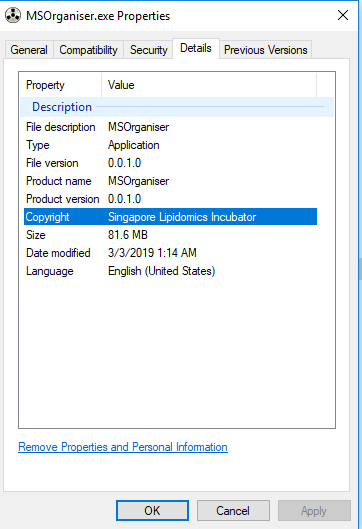
]

)

In the build.spec file, add this highlighted information



This will cause the MSOrganiser to have a version number



# Using Pyinstaller to convert from python codes to exe files

## Pyinstaller Documentation

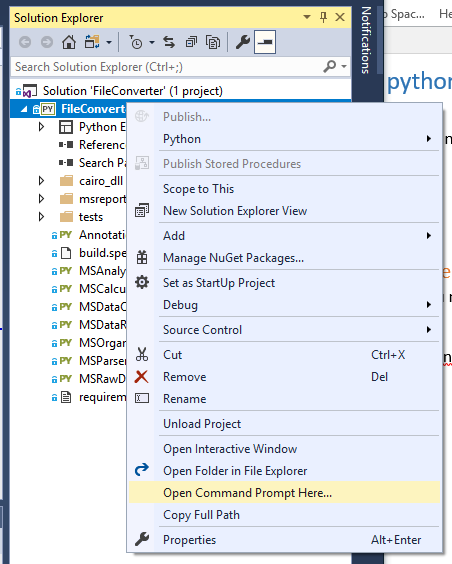
It can be found in <https://pythonhosted.org/PyInstaller/>

## Pyinstaller Command

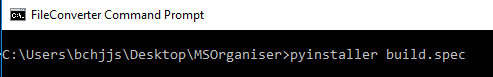
Ensure that you are online before proceeding to the next step

Once the changes in the last section are made, open the cmd in the same working directory as the script MSOrganiser.py using VS2017.

Right click on the python project, click on “Open Command Prompt Here…”



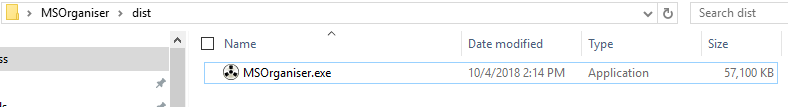
Call the command “pyinstaller build.spec”



Pyinstaller will print some things on the screen.

When it is completed, there will be two folders that are being created. One is the “build” folder and the other is the “dist” folder.

Go to the “dist” folder followed by the “MSOrganiser” folder. You will see the stand alone executable file



## Troubleshooting when Pyinstaller notify a warning or error.

If the python code s bug free, Pyinstaller should run without errors. We do occasionally faced some warnings issues due to the new environment system in Windows 10.

### api-ms-win-crt related dll files are missing

When using pyinstaller to create your exe file, you may face a warning

WARNING: lib not found: api-ms-win-crt-heap-l1-1-0.dll dependency of c:\python35\python.exe  
2044 WARNING: lib not found: api-ms-win-crt-stdio-l1-1-0.dll dependency of c:\python35\python.exe  
2068 WARNING: lib not found: api-ms-win-crt-math-l1-1-0.dll dependency of c:\python35\python.exe  
2087 WARNING: lib not found: api-ms-win-crt-runtime-l1-1-0.dll dependency of c:\python35\python.exe

The problem is that in Windows 10, these dll files are no longer available.

You must get the dll file and tell pyinstaller of its existence

#### Get the dll files

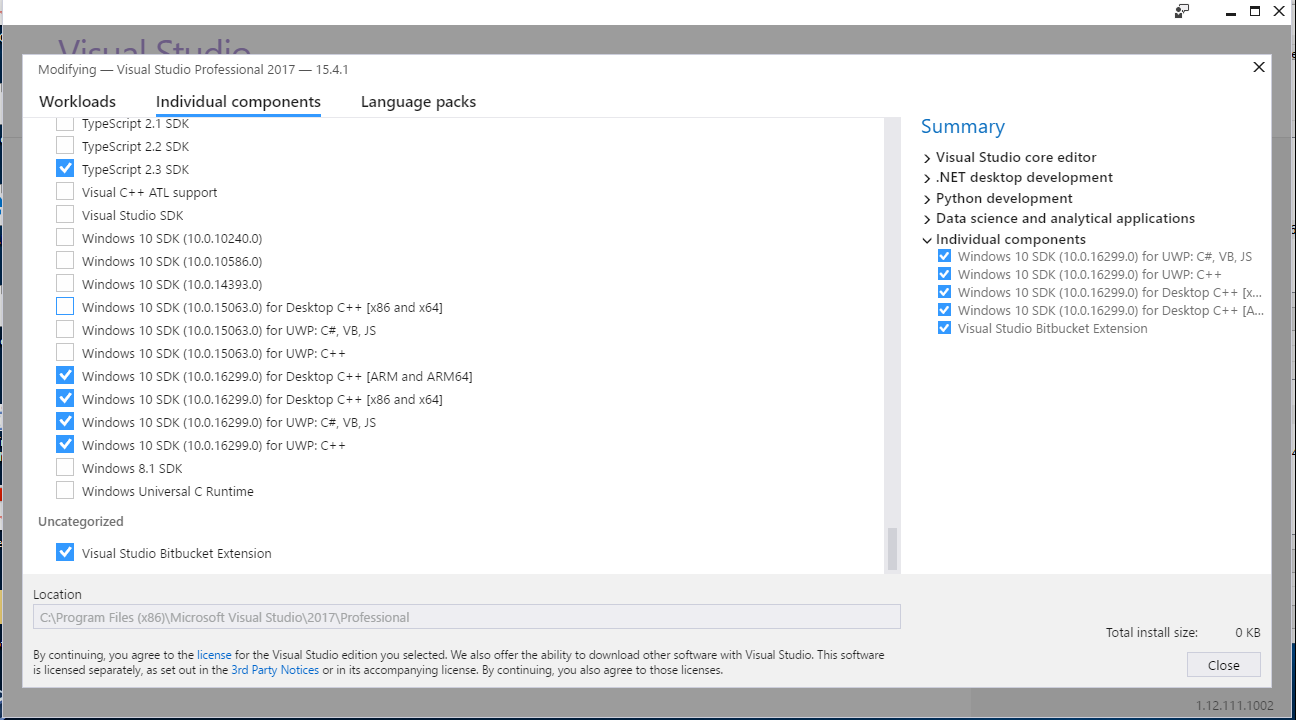
Method 1

The dll files can be obtained by the following methods:

They can be found in this file path if you are using Visual Studio 2017 C:\Program Files (x86)\Microsoft Visual Studio\2017\Professional\Common7\Tools

Method 2

One can also install these dll files using Visual Studio Installer



The dll files will be installed in C:\Program Files (x86)\Windows Kits\10\Redist\ucrt\DLLs\x64 for 64bit configuration

#### Tell Pyinstaller of the dll files existence

Method 1

https://pythonhosted.org/PyInstaller/spec-files.html

You can write in the \*.spec file the location of these binary dll files

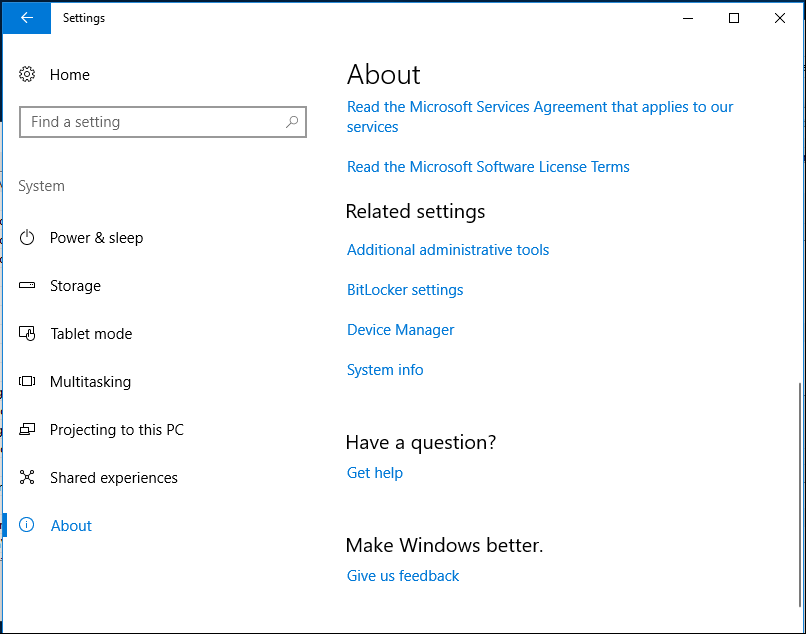
a = Analysis(... binaries=[ ( 'Location of dll file 1', 'Location of dll file 2' ) ],

If writing them is a chore, consider using the Tree Class. See <https://pythonhosted.org/PyInstaller/advanced-topics.html#the-toc-and-tree-classes>

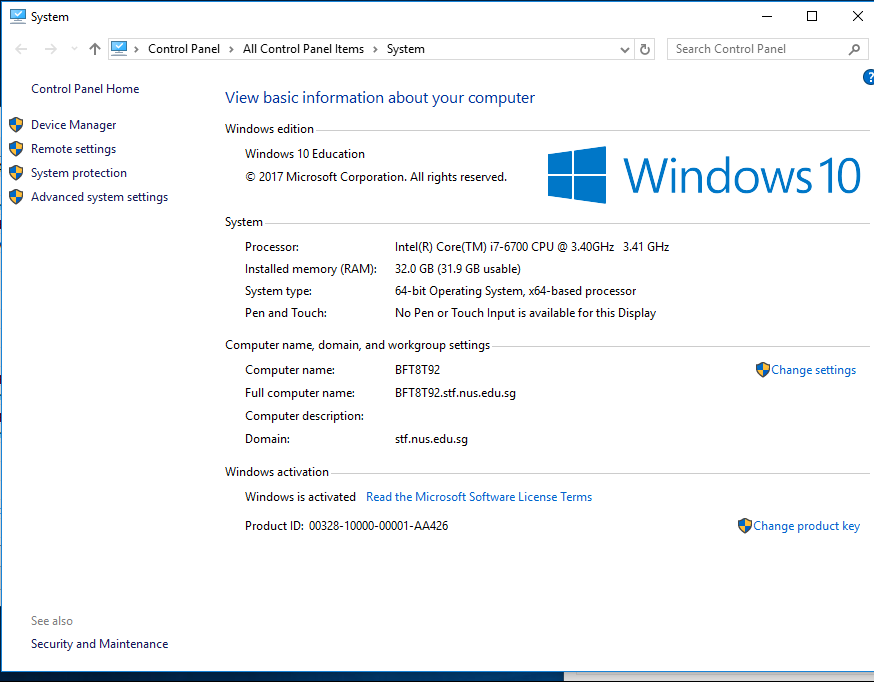
Method 2

We can put the dll path onto the environment variable PATH if you have admin rights

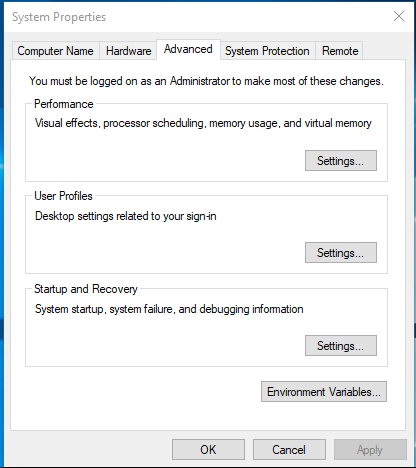
Go to Settings -> System -> About -> System info



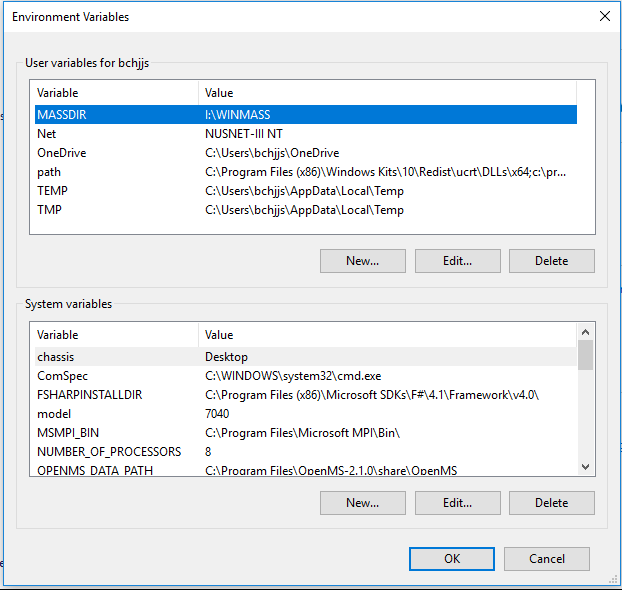
You should see this



Click on Advanced system settings



Click on Environment Variables

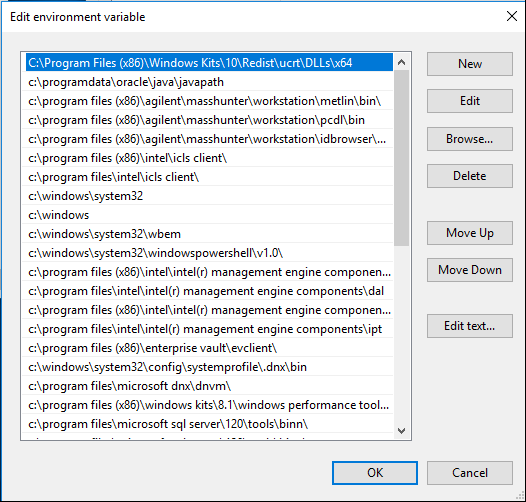


Go to the user variables and click path, it you wish to apply this for all users, go to System variables and click Path

Then click Edit…

Click on New and add your dll file path, Move it up to the top and click ok

c:\program files (x86)\windows kits\10\redist\ucrt\dlls\x64



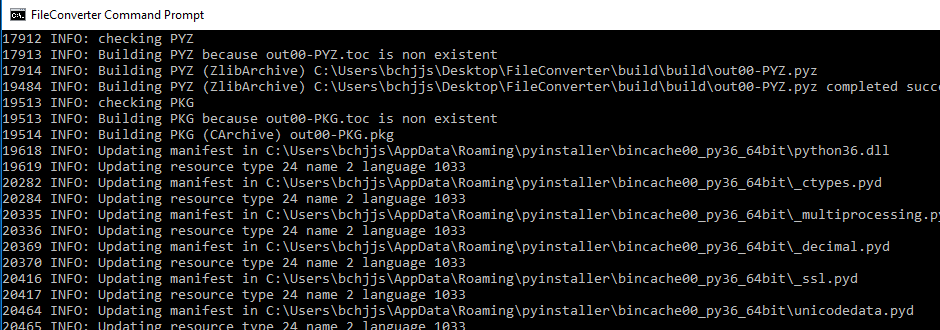
You may need to restart your computer for the change to take place.

 To test that it works,

 Go to cmd and type "echo %PATH%", check if the filepath you inserted is presence

#### Run Pyinstaller again

This time you should see this instead without any warnings nor errors

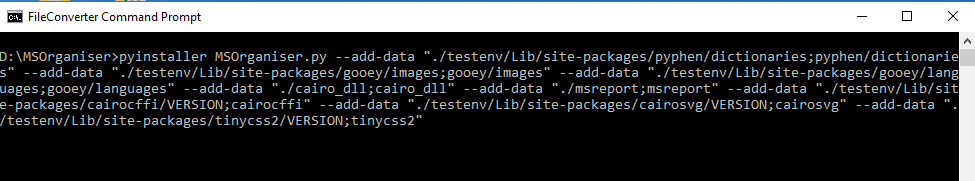


## Troubleshooting when the standalone exe file fails to run.

In times when the standalone exe file still could not start, you may have to look into details on what dependencies are missing.

To see the dependencies that are currently transferred by Pyinstaller. Open the cmd in the MSOrganiser directory and call the below command

pyinstaller MSOrganiser.py --add-data "./testenv/Lib/site-packages/pyphen/dictionaries;pyphen/dictionaries" --add-data "./testenv/Lib/site-packages/gooey/images;gooey/images" --add-data "./testenv/Lib/site-packages/gooey/languages;gooey/languages" --add-data "./cairo\_dll;cairo\_dll" --add-data "./msreport;msreport" --add-data "./testenv/Lib/site-packages/cairocffi/VERSION;cairocffi" --add-data "./testenv/Lib/site-packages/cairosvg/VERSION;cairosvg" --add-data "./testenv/Lib/site-packages/tinycss2/VERSION;tinycss2"

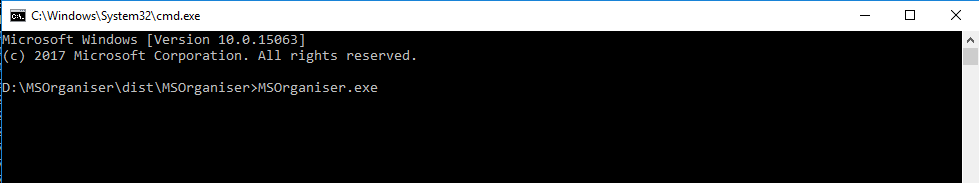


There will be two folders that are being created. One is the “build” folder and the other is the “dist” folder. One file called MSOrganiser.spec will also be created in the directory where the command is called.

Pyinstaller will print some things on the screen. Make sure that all the log status is INFO, not WARNING or ERROR

Go to the “dist” folder followed by the “MSOrganiser” folder. You will see all the dependencies Pyinstaller has created.

Open a cmd in this folder and call MSOrganiser.exe



The error messages will be displayed.

Please resolve this error by adding new dependencies (mentioned in the error message) and update the next section.

## MSOrganiser errors

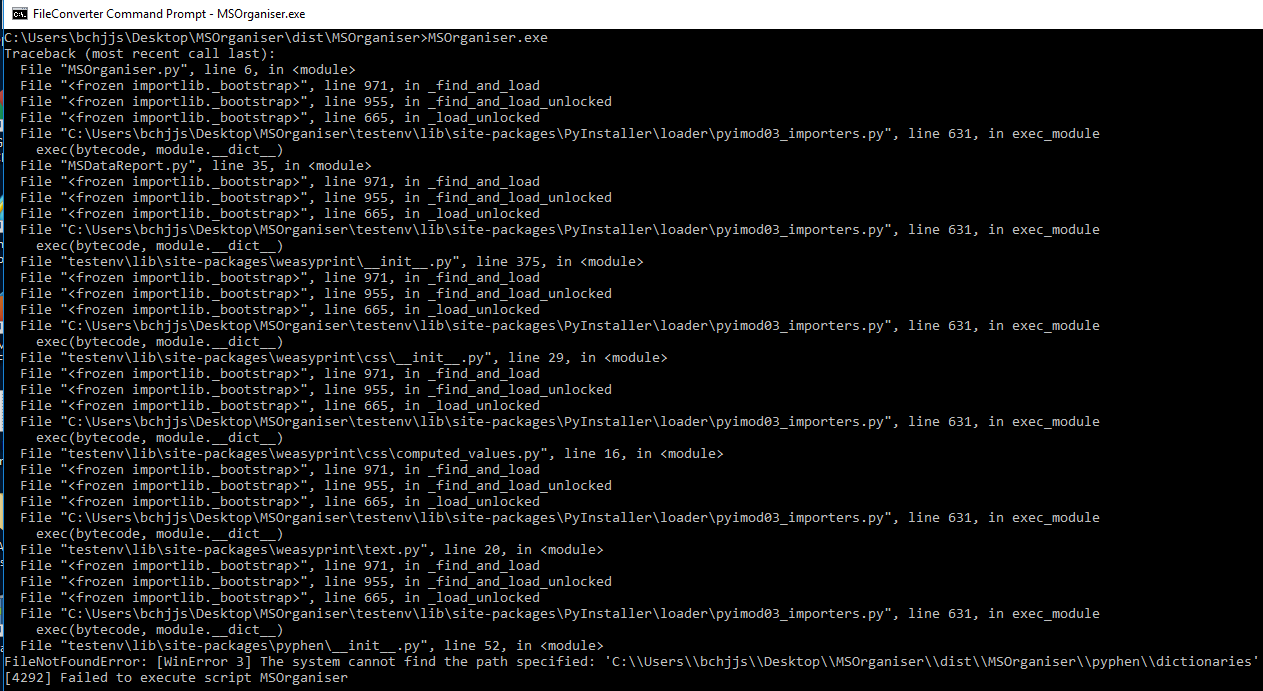
### No gooey folder



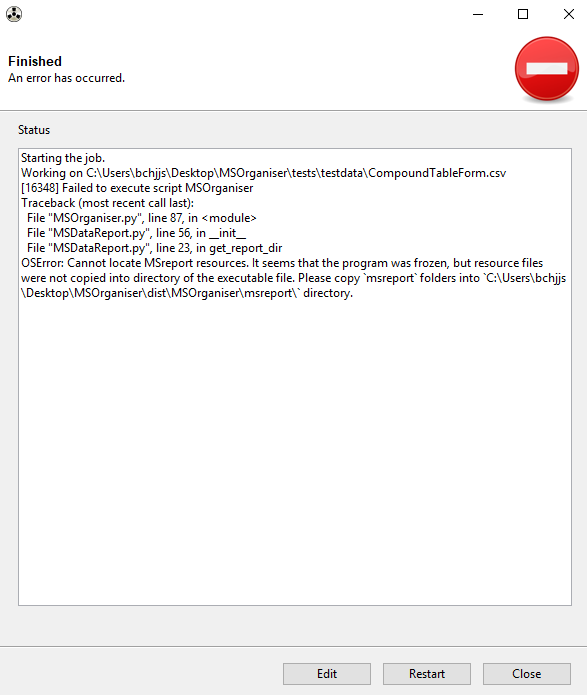
### No cairo\_dll folder



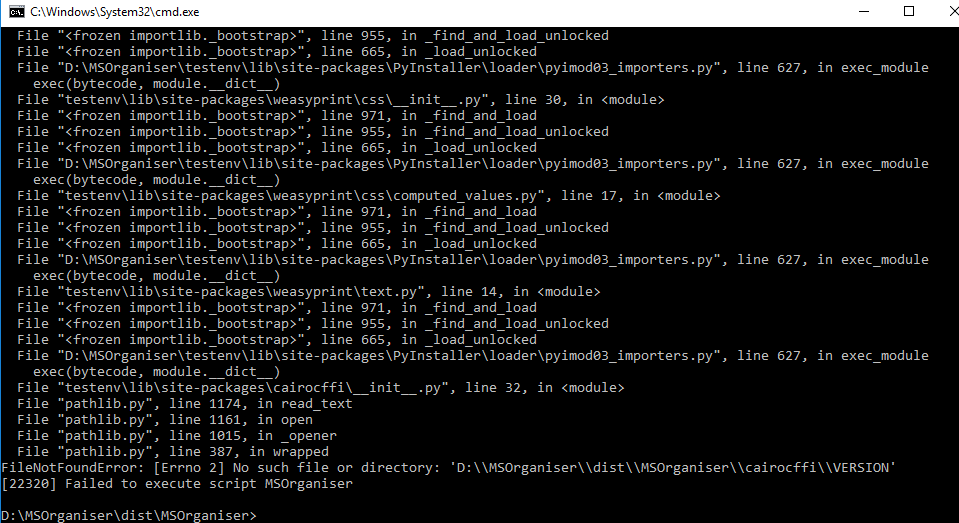
### No pyphen folder



### No msreport folder



### VERSION File missing



In the above example, ensure that the cairocffi folder is created with the VERSION file copied from the cairocffi package.

### Fontconfig error/warning

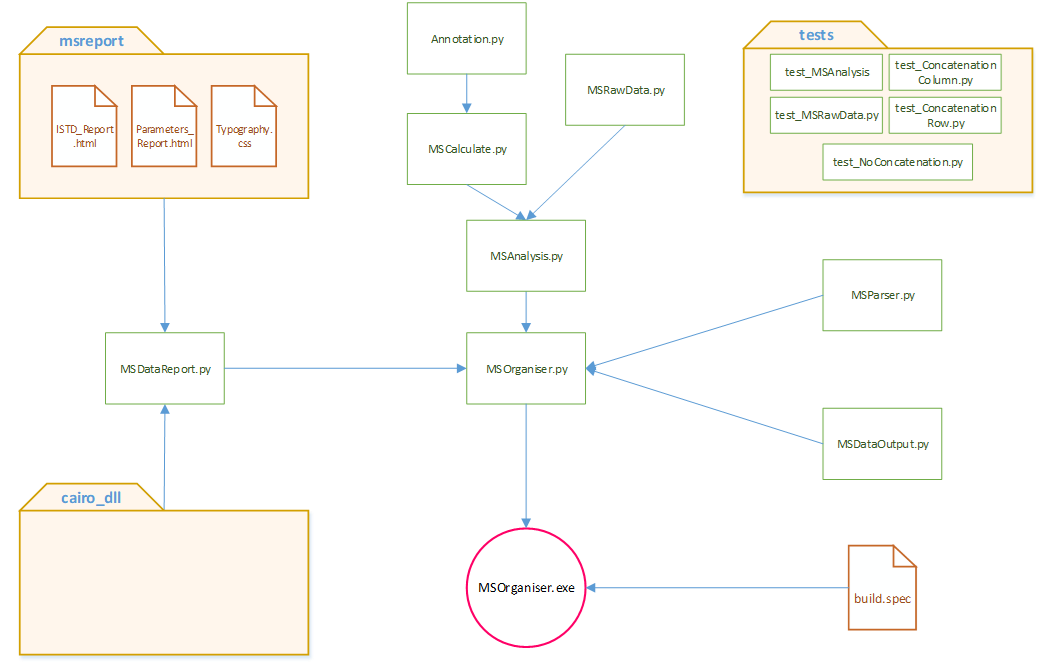


Make sure that the latest version of GTK3 is installed.

Ensure that the etc folder from the GTK3 is copied into “cairo\_dll” folder.

Copy all the dll files in C:\Program Files\GTK3-Runtime Win64\bin

# Code Structure Outline



# Code Documentation using Sphinx

The documentation is written in the folder docs. It looks like this



The documentation can also be found in the code respository.

We use python Sphinx <http://www.sphinx-doc.org/en/master/> to create this documentation

## Installation of Sphinx and related packages.

Use pip install to install the package Sphinx and one of its extensions sphinx\_rtd\_theme

## Creating the docs directory template from scratch

When sphinx is successfully installed, you will be able to use the command “sphinx-quickstart” to start a new documentation

Just create a new folder and call “sphinx-quickstart” in that folder.

Sphinx will then ask you a few questions on how to set up your new documentation template from scratch.

Press Enter to accept the default values. However, please note of the following non-default values

* > Separate source and build directories (y/n) [n]: y
* > Project name: MSOrganiser
* > Author name(s): SLING
* > Project release []: 0.0.1
* > Create Makefile? (y/n) [y]: y
* > Create Windows command file? (y/n) [y]: y

After answering the questions, Sphinx will create the following files and your new template is created

*Creating file .\source\conf.py.*

*Creating file .\source\index.rst.*

*Creating file .\Makefile.*

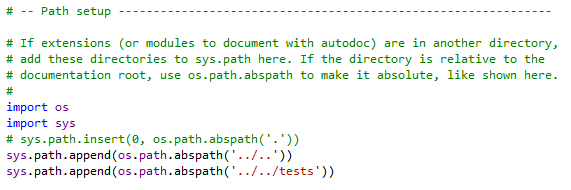
*Creating file .\make.bat.*

*Finished: An initial directory structure has been created.*

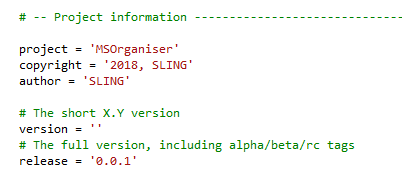
## Important configuration changes

Sphinx default configuration is written in conf.py. Please make the following changes

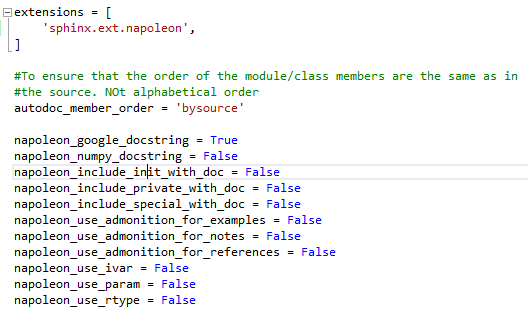
* Set up your paths to your relevant python codes and modules so that Sphinx can find them



* Update your project information



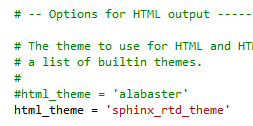
* Add these Sphinx extensions and their corresponding configuration

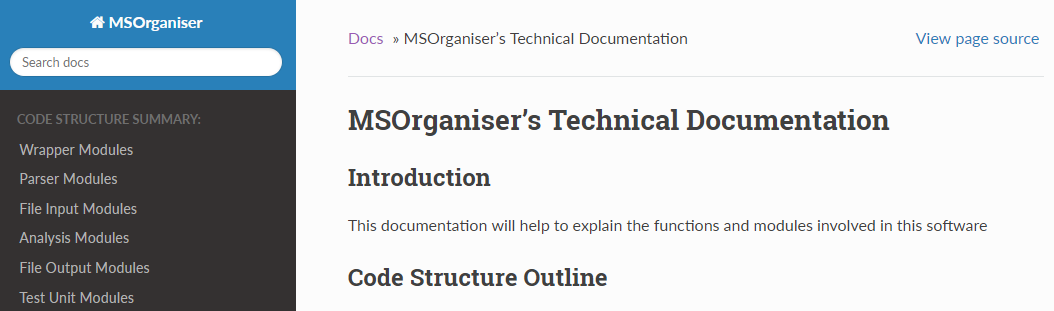


This is to ensure that we can write human legible docstrings (like Google or Numpy) in our code that is understandable to Sphinx

See <http://www.sphinx-doc.org/en/master/ext/napoleon.html> for more info

* Change the html theme to “sphinx\_rtd\_theme” for a nice documentation look online





## reStructuredText

Sphinx default documentation language is reStructuredText (RST). It will take some time to learn this language but cheat sheets are available.

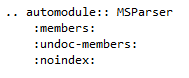
<http://aiida-core.readthedocs.io/en/latest/developer_guide/sphinx_cheatsheet.html#main-titles-and-subtitles>

<http://aiida-core.readthedocs.io/en/latest/developer_guide/sphinx_cheatsheet.html#this-page>

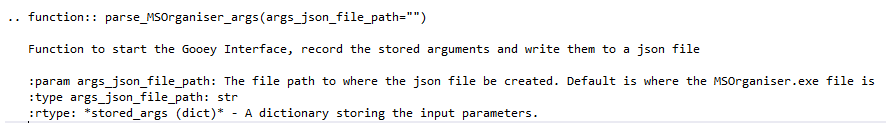
<https://thomas-cokelaer.info/tutorials/sphinx/rest_syntax.html>

Important things are as follows

* Ensure that the automodule configuration is set to noindex

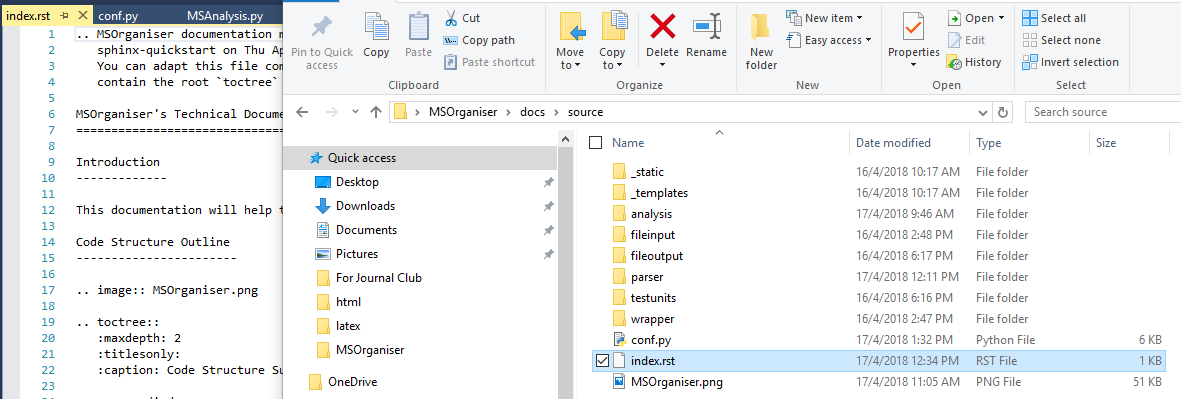


* Due to a strange bug in the Gooey class, we are unable to document parse\_MSOrganiser\_args from MSParser.py using the Google docstring automatically, so we had to write the function explicitly in MSParser.rst



* Ensure that the index.rst in the docs/source folder has connection to the image file MSOrganiser.png via

*.. image:: MSOrganiser.png*

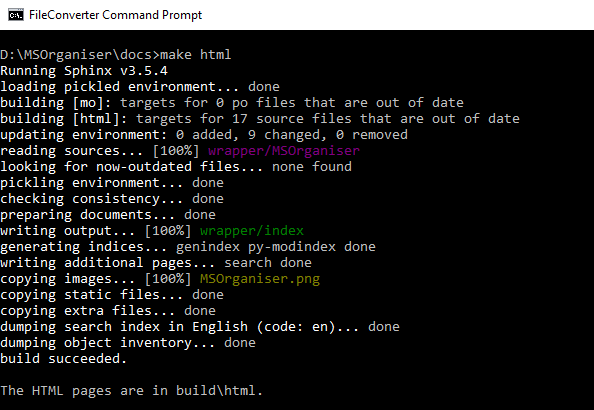


## Creating an online documentation

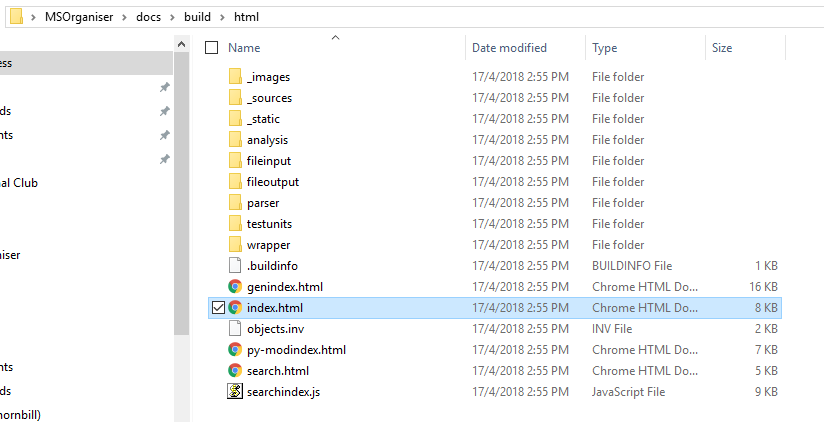
Once you are done writing the RST code, you will need to convert them to html.

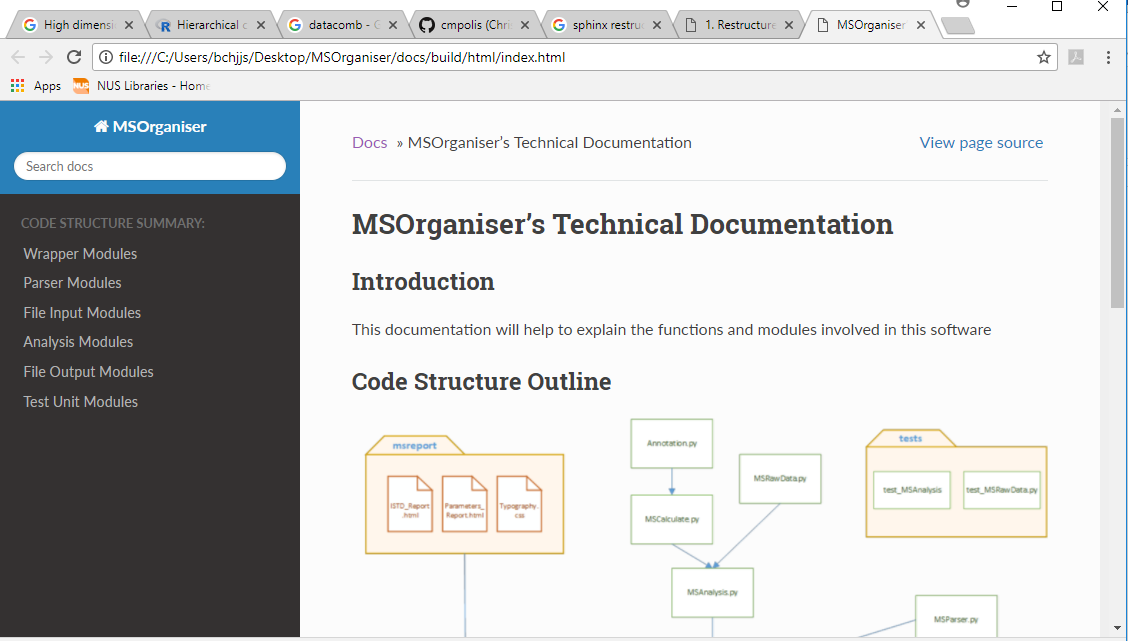


To do so, go to this folder and type make html



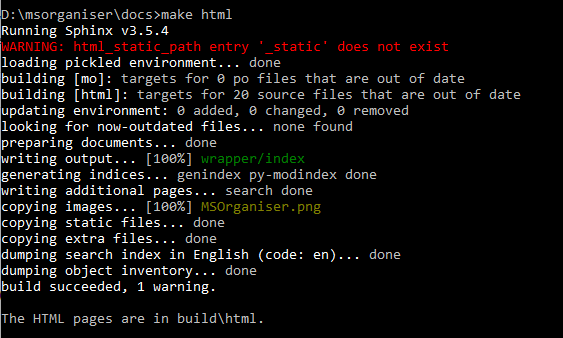
Sphinx will create a build/html folder. Click on index.html to see the results.



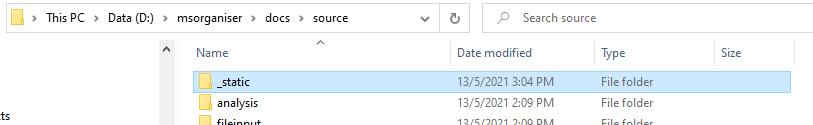


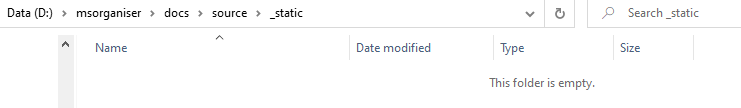
### \_statics Warning

During the creation of the html documentation, the following warning may be seen



This warning can be removed by adding an empty “\_static” folder,





Unfortunately, I do not know why this empty folder did not get detected by Git.

## Creating a pdf documentation

Once you are done writing the RST code, you will need to convert them to pdf.

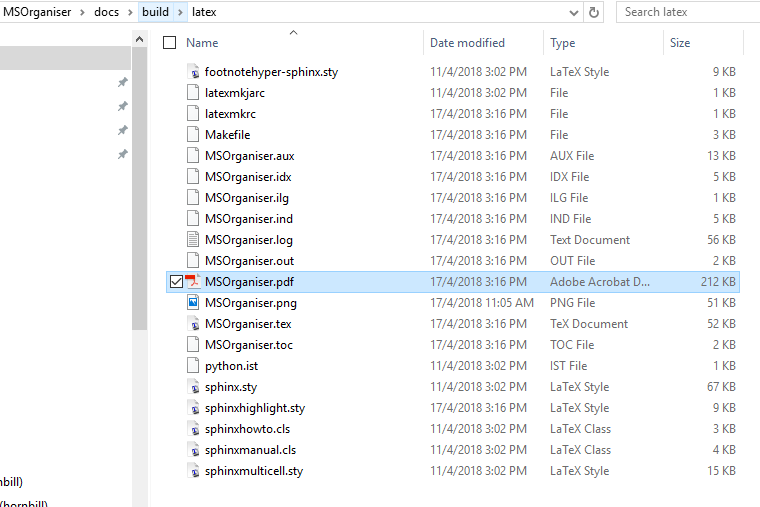
Ensure that LaTeX/MiKTeX is installed



To do so, go to this folder and type make latexpdf



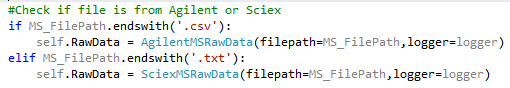
Sphinx will create a build/latex folder. Click on MSOrganiser.pdf to see the results.



# Code Update Guidelines

## Extracting MRM transition data taken from a new machine

Currently, the way to identify Agilent to Sciex is by the file extension csv and text respectively. This is because this is the default output file for MassHunter and MultiQuant respectively.



This code can be found in file MSAnalysis.py.

To handle a MRM transition data from a new machine,

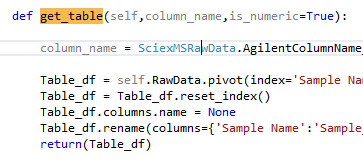
Firstly, you will need to first identify what features is able to differentiate MRM transition data from different software. The new function in MSAnalysis.py may need to be updated to identify them.

Secondly, you will need to create a new inherited class (from MSRawData) in the module MSRawData.py



You will need code to check that the file is valid, check that the output options is valid and a function to change Agilent Output options names to suit the one for your machine. You can use the class SciexMSData as an example

Thirdly, you will need to create a wrapper function called “get\_table” in the new class. This function will be used by MSAnalysis.py to extract data and convert them into a df.



Next, you will need to update the Gooey Parser Gui interface in MSParser.py



This is to indicate to the users what are the compulsory columns required for your program to work.

By doing this the program should work. Remember to update the documentation.

## Adding a new output option that requires extraction from MRM transition data

For the output options names, we set it to Agilent’s name by default. The current output options are [Area,RT,FWHM,S/N, Precursor Ion, Product Ion]. Please note that “Precursor Ion” and “Product Ion” is not available for SciEx as it cannot be outputted using MultiQuant.

In MSParser, update the choices in output options parser



### Agilent MassHunter

If the raw data is from Agilent, update the class AgilentMSRawData variables below



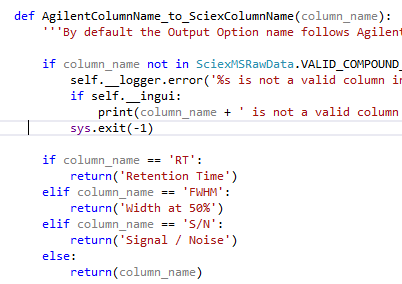
The code should be working

### Sciex MultiQuant

If the raw data is from Sciex, update the class SciexMSRawData variables below



If the output option name is different from the the Sciex, you may need to update the function AgilentColumnName\_to\_SciexColumnName



## Adding a new output option that requires calculation

The current output options are [normArea from ISTD, normConc from ISTD].

In MSParser, update the choices in output options parser



You will need to update the following two files.

MSAnalysis.py

This file should contain functions that act as a wrapper to perform the calculations. The code should be simple and short as possible.

MSCalculate.py

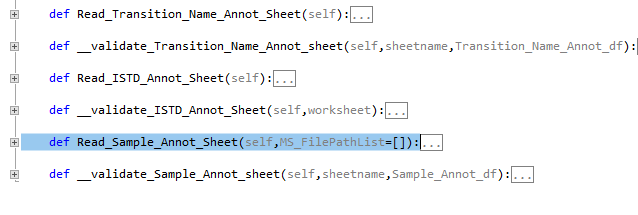
This file should contain the code that does the check if the df input is valid, logging of calculation status and errors, calculations.

We currently have a class called ISTD\_Calculation since all our calculation require the ISTD information. A new class can be created for other calculations. A new file/module can also be created if the code is too long.

## MSOrganiser to read a new sheet from MS Template Creator as input

We set to main function for each sheet.

* Read\_{some sheet}
* \_\_Validate\_{some sheet}



Please stick to this organisation.

## MSOrganiser to read a new file as input

Firstly, you may need to update the MSParser.py, if the usage of the file is very different from the current version. If a new text box input is required, you may need to update the function “get\_Parameter\_df” to store that new text box input.

If it is related to reading an annotation file, you may update the file “Annotation.py”. A new class may need to be created if necessary.

You may need to create a new module/file and link it to MSAnalysis.py and MSCalculate.py if necessary.

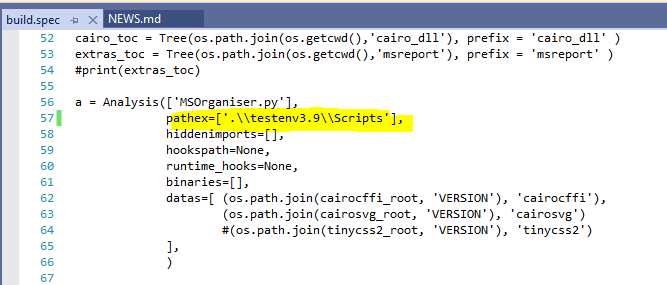
## Upgrading GTK software

After upgrading the GTK software, remember to update the cairo\_dll folder as mentioned in 9.1.3 Weasyprint’s cairo dependencies.

## Upgrading Python or Change of Virtual Environment folder.

Ensure that 7 Changes of codes in the python package after installation is done.

Also, ensure that the build.spec file points to the correct virtual environment folder



# Other Matters

## Importance of print flush=True

In Python 3, we need to set flush to “True” for each print function we use. This is to ensure that the Gooey GUI Interface output the print immediately when the software reaches this part of the code.

Failure to do so will cause the code to store the output into a buffer which only be printed out when the code stop/finish running. This is a waste of memory space.