# Glasswall Core 2 Wrapper Documentation

# Python

## Purpose

The purpose of the wrappers is to expose the Core 2 SDK functionality through Python, C#, JavaScript and Java.

Each wrapper consists of:

* The wrapper itself: a bridge between the Core 2 SDK and the desired language.
* A series of supporting files (language dependent).

## General Requirements

The following general requirements must be met to use the wrappers and their test apps:

* The Glasswall Core 2 libraries and their dependencies
  + glasswall\_core2.dll
  + \*\_camera.dll
* A designated folder containing files to be input into Core 2
* A designated folder to hold the output from Core 2
* A policy file to modify the default Core 2 file processing behaviour
* The wrapper itself.

## Test Application Overview

Each wrapper is provided with a test application. This application is designed to call each of the Core2 APIs from the chosen language and generate a log file of the results.   
The execution steps are:

* All supporting files, folders and dependencies are checked
* The contents of the output directory are erased in preparation for file processing
* For each file in the input directory:
  + A new folder is created in the output directory and is named for the file currently being processed
  + A series of 23 tests are performed, as detailed in the Wrapper Test Calls document
  + The files generated by Glasswall are saved in the specified output directory
  + A log file detailing the result of each test is generated and saved as local\_process\_log.txt
* When all files have been processed, a final log file named process\_log.txt is saved in the root of the output directory.

## Python Wrapper

* This has been tested on Python 3.6 and Python 2.7.

### Environment

* You need to ensure that you have the Glasswall.py wrapper file accessible.
* Ensure the Core 2 libraries and all required folders are accessible to the wrapper.

### Example Code

The following code uses the Python wrapper to process a file and place the managed file in a buffer. A policy file is specified, and an analysis report is generated. Note that the memory buffer is non-persistent and will have to be processed, analysed, or stored, before the script finishes. The policies file, config\_sanitise.xml, will be placed in the specified directory.

|  |
| --- |
| from Glasswall import Glasswall  import ctypes as ct    class PythonWrapper:  buffer = ct.c\_void\_p()  buffer\_length = ct.c\_size\_t()    # create Glasswall object.  gw = Glasswall(r'e:\Core2\_dlls')    # open session  session\_id = gw.open\_session()    # register inputfile  return\_status = gw.register\_input\_file(session\_id, r'e:\Input\A.xlsx')    # register outputmemory: the processed data is non-persistent  return\_status = gw.register\_output\_memory(session\_id, buffer, buffer\_length)    # register policies file  return\_status = gw.register\_policies\_file(session\_id, r'e:\Config\ config\_sanitise.xml', 0)    # register analysis file  return\_status = gw.register\_analysis\_file(session\_id, r'e:\Output\Analysis.xml')    # run the session  return\_status = gw.run\_session(session\_id)    # close the session  return\_status = gw.close\_session(session\_id) |

## 