

Glasswall API Language Wrapper Documentation

Python

© 2020 Glasswall Solutions Ltd ALL RIGHTS RESERVED

Information contained herein is the property of Glasswall Limited and is proprietary and confidential.

Glasswall Solutions Ltd.

[support@glasswallsolutions.com](mailto:support%40glasswallsolutions.com)

Creation Date – 14 October 2020

Version – 1

**Copyright and Contact Details**

The copyright in this work is vested in Glasswall Solutions Ltd, and the document is issued in confidence for the purpose for which it is supplied. It must not be reproduced in whole or in part or used for tendering or manufacturing purposes except under agreement or with the consent in writing of Glasswall Solutions Limited and then only on condition that this notice is included in any such reproduction. No information as to the contents or subject matter of this document or any part thereof arising directly or indirectly there from shall be given orally or in writing or communicated in any manner whatsoever to any third part being an individual firm or company or any employee thereof without the prior consent in writing of Glasswall Solutions Limited.

© Glasswall Solutions Limited 2020

If there are any questions related to this report, these should be addressed to:

Glasswall Solutions Limited

e-mail: [support@glasswallsolutions.com](mailto:support%40glasswallsolutions.com)

**Index**

[1. Introduction 4](#_Toc53562336)

[1.1 Issues to consider during implementation 4](#_Toc53562337)

[2. Python 5](#_Toc53562338)

[2.1 Files provided 5](#_Toc53562339)

[2.2 Framework dependencies 5](#_Toc53562340)

[2.3 Wrapper integration 5](#_Toc53562341)

[2.4 Code example 5](#_Toc53562342)

[2.5 Issues to consider 6](#_Toc53562343)

# Introduction

This is an introductory guide on how to use the language wrappers for the Glasswall API.

For each language we have provided the following:

* The required dependencies and the target platform or framework required for the language. The mentioned platforms and frameworks are the ones that were used during testing, but other versions may or may not work.
* A general overview on integrating Glasswall into a project.
* A code example demonstrating how the Glasswall library can be used to process a directory of files. Each example shows the file being processed in Manage and Protect mode as well as being analysed in Analysis mode.

## Issues to consider during implementation

* The Glasswall library is not thread safe, which means that the language wrappers are not thread safe. This can be overcome by running the Glasswall library in a separate process.
* We recommend that the Glasswall library is run in separate process in case unforeseen issues arise.

# Python

## Files provided

The Python wrapper is provided as a single Python file that you reference in your application.

Glasswall.py – The Python file containing the Glasswall module that is responsible for interacting with the Glasswall library.

## Framework dependencies

The Python wrapper requires either Python 2.7 or Python 3 to be installed.

## Wrapper integration

The Python wrapper can be integrated by importing the Glasswall module.

## Code example

|  |
| --- |
| import os  import argparse  from Glasswall import \*  def getCmdArgs():  parser = argparse.ArgumentParser(description='Glasswall Python Wrapper Example')  parser.add\_argument('-i', action="store", dest="i", help="Input Directory", type=str)  parser.add\_argument('-o', action="store", dest="o", help="Output Directory", type=str)  parser.add\_argument('-c', action="store", dest="c", help="Path to CM config file", type=str)  parser.add\_argument('-p', action="store", dest="p", help="Path to .DLL or .SO", type=str)  args = parser.parse\_args()  return args.i, args.o, args.c, args.p  def writeFile(fileName, outputDir, content):  fileHandler = open(os.path.join(outputDir, fileName), "wb")  fileHandler.write(content)  fileHandler.close()  def main():  inputDirectory, outputDirectory, pathToConfig, pathToLib = getCmdArgs()  print("Loading Library...")  # Load Glasswall Lib  gw = Glasswall(pathToLib)  print("Done!")  os.makedirs(outputDirectory)  # GWFileConfigXML Test  configFile = open(pathToConfig, "r")  xmlContent = configFile.read()  configFile.close()  # Apply the content management configuration  configXMLResult = gw.GWFileConfigXML(xmlContent)  if configXMLResult.returnStatus != 1:  print("Failed to apply the content management configuration for the following reason: " + gw.GWFileErrorMsg())  return  for root, folders, files in os.walk(unicode(inputDirectory, 'utf-8')):  for eachFile in files:  filepath = os.path.join(root, eachFile)  print("Processing file: " + filepath)  # We use the extension as the file type of the file to be processed  filename, fileExtension = os.path.splitext(eachFile)  # Process the file in File to Memory Protect mode  manageAndProtectResult = gw.GWFileProtect(filepath, fileExtension[1:])  if manageAndProtectResult.returnStatus == 1:  writeFile(eachFile, outputDirectory, manageAndProtectResult.fileBuffer)  # Analyse the file in File to Memory Analysis mode  analysisResult = gw.GWFileAnalysisAudit(filepath, fileExtension[1:])  writeFile(eachFile + ".xml", outputDirectory, analysisResult.fileBuffer)  gw.GWFileDone()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

## Issues to consider

* File paths with Unicode characters need to be correctly encoded otherwise unexpected behaviour may occur.