

Glasswall API Language Wrapper Documentation

© 2018 Glasswall Solutions Ltd ALL RIGHTS RESERVED

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

Glasswall Solutions Ltd.

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

Creation Date – 21/09/2018

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 2018

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](#_Toc530563316)

[1.1 Issues to consider during implementation 4](#_Toc530563317)

[2. C# 5](#_Toc530563318)

[2.1 Files provided 5](#_Toc530563319)

[2.2 Framework dependencies 5](#_Toc530563320)

[2.3 Wrapper integration 5](#_Toc530563321)

[2.4 Code example 5](#_Toc530563322)

[2.5 Issues to consider 6](#_Toc530563323)

[3. Java 7](#_Toc530563324)

[3.1 Files provided 7](#_Toc530563325)

[3.2 Framework dependencies 7](#_Toc530563326)

[3.3 Wrapper integration 7](#_Toc530563327)

[3.4 Code example 7](#_Toc530563328)

[3.5 Issues to consider 9](#_Toc530563329)

[4. JavaScript 10](#_Toc530563330)

[4.1 Files provided 10](#_Toc530563331)

[4.2 Dependencies 10](#_Toc530563332)

[4.2.1 Framework dependencies 10](#_Toc530563333)

[4.2.2 Module dependencies 10](#_Toc530563334)

[4.3 Wrapper integration 10](#_Toc530563335)

[4.4 Code example 10](#_Toc530563336)

[4.5 Issues to consider 11](#_Toc530563337)

[5. Python 12](#_Toc530563338)

[5.1 Files provided 12](#_Toc530563339)

[5.2 Framework dependencies 12](#_Toc530563340)

[5.3 Wrapper integration 12](#_Toc530563341)

[5.4 Code example 12](#_Toc530563342)

[5.5 Issues to consider 13](#_Toc530563343)

# 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.

# C#

## Files provided

glasswall.classic.csharp.dll – The C# wrapper is provided as a .Net Framework Class Library that can be integrated into a project and used to interact with the Glasswall library. There are two OS specific versions of the library: one for the Windows OS and one for the Linux OS. Please ensure that you reference the correct version depending on your target platform otherwise you will get a DllNotFoundException when trying to interact with Glasswall.

## Framework dependencies

The C# wrapper requires the .Net Framework 4.5 to be installed or a Mono equivalent.

## Wrapper integration

The wrapper can be integrated into a project by simply adding a reference to the wrapper Library. The wrapper library uses the DllImport attribute to load the Glasswall library, which means that the Glasswall library needs to be in the search path in order to be executed. More information can be found at <https://www.mono-project.com/docs/advanced/pinvoke/>

## Code example

using System;

using System.IO;

using GlasswallSolutions;

namespace glasswall.classic.csharp.testing

{

class Program

{

public static void Main(string[] args)

{

string xmlConfigPath = args[0]; // The path to the XML content management configuration

string inputDirectory = args[1]; // The input directory

string outputDirectory = args[2]; // The output directory

// Print the Glasswall library version

Console.WriteLine("Library version: {0}", Glasswall.GWFileVersion());

Directory.CreateDirectory(outputDirectory);

string xmlConfig = File.ReadAllText(xmlConfigPath);

// Apply Content Management Configuration

if (Glasswall.GWFileConfigXML(xmlConfig) != 1)

{

// Print the failure reason

Console.WriteLine("Failed to load the XML config file: {0}", Glasswall.GWFileErrorMsg());

return;

}

foreach (string path in Directory.EnumerateFiles(inputDirectory, "\*", SearchOption.AllDirectories))

{

Console.WriteLine("Processing file: {0}", path);

string outputPath = Path.Combine(outputDirectory, Path.GetFileNameWithoutExtension(path));

string extension = Path.GetExtension(path).Trim('.'); // We use the file extension as the file type

int status = 0; // The file process status

byte[] fileProtect; // The buffer for the protected file

status = Glasswall.GWFileProtect(path, extension, out fileProtect); // Run the file through File to Memory Protect

PrintProcessInfo(status); // Print out the status

WriteAllBytes(outputPath + "." + extension, fileProtect); // Write the protected file

string analysisFile; // The analysis report

status = Glasswall.GWFileAnalysisAudit(path, extension, out analysisFile); // Analyse the file with File to Memory Analysis

PrintProcessInfo(status); // Print out the status

WriteAllText(outputPath + ".xml", analysisFile); // Write the analysis report

}

// Clean up any resources allocated by Glasswall

Glasswall.GWFileDone();

}

static void WriteAllBytes(string path, byte[] data)

{

if (data == null) File.Create(path);

else File.WriteAllBytes(path, data);

}

static void WriteAllText(string path, string data)

{

if (string.IsNullOrEmpty(data)) File.Create(path);

else File.WriteAllText(path, data);

}

static void PrintProcessInfo(int fileProcessStatus)

{

uint processStatus = 0;

Glasswall.GWFileProcessStatus(ref processStatus); // Get the process status

Console.WriteLine("Process status: {0}", processStatus);

Console.WriteLine("Process message: {0}", Glasswall.GWFileProcessMsg()); // Print the file process message

if (fileProcessStatus != 1) Console.WriteLine("Glasswall error: {0}", Glasswall.GWFileErrorMsg()); // Print non-conformance reason

}

}

}

## Issues to consider

* Very large files can potentially cause Out of Memory Exceptions to occur when running in File to Memory mode. This is due to certain results being returned as byte arrays. Potential remedy is to process files in File to File mode.
* In very exceptional circumstances certain PDF documents may cause a StackOverflowException to occur. This can be mitigated by increasing the stack size of the application.

# Java

## Files provided

The Java wrapper is supplied in two parts: A C++ library that interacts with the Glasswall library and a Jar file that allows Java to interact with the C++ library.

libglasswall.classic.java.so.1.1.0 – The C++ library for the Linux OS.

glasswall.classic.java.dll – The C++ library for the Windows OS.

glasswall.classic.java.jar – The Jar file that enables interop between Java and the C++ library. It contains two classes: Glasswall which is used for executing the Glasswall APIs and GlasswallResult which contains the results from most calls to Glasswall.

## Framework dependencies

The Java wrapper requires a JVM installed that supports Java 8.

## Wrapper integration

The Jar file will need to be added to your project, which will allow you to call the Glasswall library from Java. You will also need to add the path of the glasswall.classic.java library to the **java.library.path** property. The Glasswall library will also need to be stored in the library search path otherwise you will get an UnsatisfiedLinkError. The Glasswall library uses signal handling in order to prevent crashes from occurring, but this can interfere with the JVM. Signal chaining will also need to be setup in order to prevent errors in Glasswall from propagating into the JVM. Information on how to set this up can be found at <https://www.oracle.com/technetwork/java/javase/signals-139944.html>

For example, on our Linux machines we may use something like this when running the Java wrapper:

**export LD\_PRELOAD=/lib/jvm/jre-1.8.0/lib/amd64/libjsig.so**

**export LD\_LIBRARY\_PATH=.**

**java –cp glasswall.classic.java.jar:. –Djava.library.path=. …**

In the above example the Glasswall library, the glasswall.classic.java library, the glasswall.classic.java.jar and the Java executable are all in the current directory.

## Code example

**import** java.io.IOException;  
**import** java.io.PrintWriter;  
**import** java.nio.file.Files;  
**import** java.nio.file.Path;  
**import** java.nio.file.Paths;  
**import** java.nio.file.StandardOpenOption;  
  
**import** com.glasswallsolutions.Glasswall;  
**import** com.glasswallsolutions.GlasswallResult;  
  
**public class** GlasswallExample {  
  
 **public static void** main(String[] args) **throws** IOException  
 {  
 Path xmlConfigPath = Paths.*get*(args[0]); *// Path to the XML content management file* Path inputDirectoryPath = Paths.*get*(args[1]); *// Path to the input directory* Path outputDirectoryPath = Paths.*get*(args[2]); *// Path to the output directory* String absoluteOutputDirectory = outputDirectoryPath.toAbsolutePath().toString();  
  
 Files.*createDirectories*(outputDirectoryPath);  
  
 *// Print the library version* System.***out***.println(String.*format*(**"Library version: %s"**, Glasswall.*GWFileVersion* ()));  
  
 **byte**[] xmlConfigBuffer = Files.*readAllBytes*(xmlConfigPath);  
 String xmlConfig = **new** String(xmlConfigBuffer, **"UTF-8"**);  
  
 *// Apply the content management configuration* GlasswallResult configResult = Glasswall.*GWFileConfigXML*(xmlConfig);  
  
 **if** (configResult.**fileStatus** != 1)  
 {  
 System.***out***.println(String.*format*(**"Failed to load the xml config file for the following reason: %s"**, Glasswall.*GWFileErrorMsg*()));  
 **return**;  
 }  
  
 Files.*walk*(inputDirectoryPath)  
 .forEach((Path file) -> {  
 **if** (Files.*isRegularFile*(file))  
 {  
 String fullFilePathName = file.toAbsolutePath().toString();  
 System.***out***.println(String.*format*(**"Processing file: %s"**, fullFilePathName));  
  
 String name = file.getFileName().toString();  
 *// We use the extension as the file type of the input file* String extension = name.substring(name.lastIndexOf(**"."**) + 1);  
  
 System.***out***.println (**"Calling GWFileProtect"**);  
 *// Process the file through File to Memory Protect mode* GlasswallResult manageAndProtectResult = Glasswall.*GWFileProtect* (fullFilePathName, extension);  
 *printProcessInfo* (manageAndProtectResult);  
 *writeAllBytes* (Paths.*get* (absoluteOutputDirectory, name), manageAndProtectResult.**manageAndProtectBuffer**);  
  
 System.***out***.println (**"Calling GWFileAnalysisAudit"**);  
 *// Analyse the file with File to Memory Analysis mode* GlasswallResult analysisResult = Glasswall.*GWFileAnalysisAudit* (fullFilePathName, extension);  
 *printProcessInfo* (analysisResult);  
 *writeAllText* (Paths.*get* (absoluteOutputDirectory, name + **".xml"**), analysisResult.**analysisReport**);  
 }  
 });  
  
 Glasswall.*GWFileDone* ();  
 }  
  
 **static void** printProcessInfo(GlasswallResult result)  
 {  
 System.***out***.println (String.*format*(**"Status is: %s"**, result.**fileStatus**));  
 GlasswallResult processResult = Glasswall.*GWFileProcessStatus*();  
 System.***out***.println(String.*format*(**"Process status: %d"**, processResult.**processStatus**));  
 System.***out***.println(String.*format*(**"Process message: %s"**, Glasswall.*GWFileProcessMsg*()));  
  
 **if** (result.**fileStatus** != 1)  
 {  
 System.***out***.println(String.*format*(**"Glasswall error: %s"**, Glasswall.*GWFileErrorMsg*()));  
 }  
 }  
  
 **static void** writeAllBytes(Path filepath, **byte**[] data)  
 {  
 **try** {  
 **if** (data == **null**)  
 {  
 **if** (Files.*notExists*(filepath))  
 {  
 Files.*createFile*(filepath);  
 }  
 }  
 **else** {  
 Files.*write*(filepath, data, StandardOpenOption.***CREATE***);  
 }  
 }  
 **catch** (IOException ex)  
 {  
 System.***out***.println(String.*format*(**"Exception occurred: %s"**, ex.getMessage()));  
 }  
 }  
  
 **static void** writeAllText(Path filepath, String data)  
 {  
 **try** {  
 **if** (data == **null**) {  
 **if** (Files.*notExists*(filepath)) {  
 Files.*createFile*(filepath);  
 }  
 } **else** {  
 **try** (PrintWriter writer = **new** PrintWriter(filepath.toAbsolutePath().toString())) {  
 writer.write(data);  
 }  
 }  
 }  
 **catch** (IOException ex)  
 {  
 System.***out***.println(String.*format*(**"Exception occurred: %s"**, ex.getMessage()));  
 }  
 }  
}

## Issues to consider

* Very large files can potentially cause OutOfMemoryError to occur when running in File to Memory mode. This is due to certain results being returned as byte arrays. Potential remedy is to process files in File to File mode.
* In very exceptional circumstances certain PDF documents may cause a crash to occur. Increasing the stack size of the Java application fixes this issue.

# JavaScript

## Files provided

The JavaScript wrapper is provided as a single JavaScript file that you include in you script.

glasswall.classic.javascript.js – The JavaScript file containing the Glasswall module that is used to interact with the Glasswall library.

## Dependencies

### Framework dependencies

The JavaScript wrapper requires NodeJS 8 to be installed.

### Module dependencies

The JavaScript wrapper depends on the following npm modules:

* ffi – 2.20
* ref – 1.3.5
* ref-wchar – 1.02

A JSON file with all specific dependencies required is deployed with the SDK wrapper, therefore the ‘npm install’ command can be executed to automatically acquire all required dependencies.

## Wrapper integration

The wrapper can be integrated into an existing project by adding the module dependencies and then calling the **require** function from your JavaScript script. The path to the Glasswall library is then passed in as an argument when constructing a Glasswall object.

It’s possible to install the modules in an offline environment with the Yarn package manager. More information can be found at <https://yarnpkg.com/blog/2016/11/24/offline-mirror/>

## Code example

let fs = require('fs');

let path = require('path')

let getFiles = function (dir, filelist) {

let files = fs.readdirSync(dir);

filelist = filelist || [];

files.forEach(function (file) {

if (fs.statSync(dir + "/" + file).isDirectory()) {

filelist = getFiles(dir + '/' + file, filelist);

}

else {

filelist.push(dir + "/" + file);

}

});

return filelist;

};

let writeToFile = function (outDirectory, fileName, content) {

fs.writeFileSync(outDirectory + "/" + fileName, content, function (err) {

if (err)

return console.log(err);

});

};

let main = function () {

const args = process.argv;

let inputDirectory = args[2]; // The input directory

let outputDirectory = args[3]; // The output directory

let pathToConfig = args[4]; // The path to the XML content management configuration

let pathToGwLib = args[5]; // The path to the Glasswall library

let glasswall = require("./glasswall.classic.javascript.js");

console.log("Loading Library...");

let gw = new glasswall(pathToGwLib);

let xmlContent = fs.readFileSync(pathToConfig);

// Apply the XML content management configuration

let configXMLStatus = gw.GWFileConfigXML(xmlContent.toString());

if (configXMLStatus != 1) {

// Print the failure reason

console.log("Failed to apply the content management configuration for the following reason: " + gw.GWFileErrorMsg());

return;

}

fs.mkdirSync(outputDirectory);

let files = getFiles(inputDirectory);

for (let i in files) {

let filePath = files[i];

let extension = path.extname(filePath).substring(1);

let filename = path.basename(filePath);

// Process the file in File to Memory Protect

let manageAndProtectResult = gw.GWFileProtect(filePath, extension);

if (manageAndProtectResult.status === 1) {

writeToFile(outputDirectory, filename, manageAndProtectResult.fileBuffer);

}

// Analyse the file in File to Memory Analysis

let analysisResult = gw.GWFileAnalysisAudit(filePath, extension);

console.log(analysisResult.status);

writeToFile(outputDirectory, filename + ".xml", analysisResult.xmlReport);

}

gw.GWFileDone();

};

if (require.main === module) {

main();

}

## Issues to consider

* The JavaScript wrapper has not been tested on NodeJS 10

# 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.