

Glasswall API Language Wrapper Documentation

Java

© 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](#_Toc53560929)

[1.1 Issues to consider during implementation 4](#_Toc53560930)

[2. Java 5](#_Toc53560931)

[2.1 Files provided 5](#_Toc53560932)

[2.2 Framework dependencies 5](#_Toc53560933)

[2.3 Wrapper integration 5](#_Toc53560934)

[2.4 Code example 5](#_Toc53560935)

[2.5 Issues to consider 7](#_Toc53560936)

# 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.

# 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 example above, 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.