

Glasswall API Language Wrapper Documentation

C#

© 2020 Glasswall Solutions Ltd ALL RIGHTS RESERVED

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

Glasswall Solutions Limited

[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](#_Toc53560387)

[1.1 Issues to consider during implementation 4](#_Toc53560388)

[2. C# 5](#_Toc53560389)

[2.1 Files provided 5](#_Toc53560390)

[2.2 Framework dependencies 5](#_Toc53560391)

[2.3 Wrapper integration 5](#_Toc53560392)

[2.4 Code example 5](#_Toc53560393)

[2.5 Issues to consider 7](#_Toc53560394)

# Introduction

This is an introductory guide on how to use the C# language wrapper 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 a 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.