# Glasswall Core 2 Wrapper Documentation

# Java

## 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.

## Java Wrapper

### Environment

* The Java wrapper requires a Java Runtime Environment (JRE) to be installed. This wrapper has been tested with Java 8. Other versions may not be compatible.
* Ensure that a 64-bit version of the JRE is used.
* The Java wrapper requires all supporting files to be stored in the same directory as the wrapper itself. This means that the Core 2 libraries and wrapper support files cannot be stored in subdirectories.
* Ensure that the Core 2 Libraries and all required folders are accessible to the wrapper.

### Notes on Use

* Ensure that strings used for specifying filenames are null terminated. The following example includes a procedure named StringToCBytes() to perform the necessary conversion.
* Retrieval of buffer data is achieved by calling one of the GetBuffer() procedures.
* Buffer data for a session must be retrieved before the session is closed.

### Example Code

The following code uses the Java 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.

|  |
| --- |
| import java.io.ByteArrayOutputStream;  import java.io.IOException;  public class Core2JavaExample  {  private static byte[] StringToCBytes(String input\_string)  {  // Procedure to convert Java String to NULL-terminated CString  ByteArrayOutputStream cstring\_stream = new ByteArrayOutputStream();  byte[] return\_value = input\_string.getBytes("UTF-8");    // Terminate with /0  cstring\_stream.write(return\_value, 0, return\_value.length);  cstring\_stream.write(0);  return\_value = cstring\_stream.toByteArray();  return return\_value;  }  public static void main(String argv[]) throws IOException  {  System.loadLibrary("Core2JavaBridge");  Core2JavaBridge gw = new Core2JavaBridge();    try  {  // open session  int session\_id = gw.GW2OpenSession();    // register inputfile  int return\_status = gw.GW2RegisterInputFile(session\_id,  StringToCBytes("C:\\temp\\input\\Test.jpg"));    // register outputmemory: the processed data is non-persistent  return\_status = gw.GW2RegisterOutputMemory(session\_id);    // register policies file  return\_status = gw.GW2RegisterPoliciesFile(session\_id,  StringToCBytes("C:\\temp\\xmlconfig.xml"), 0); // 0 = PF\_XML    // register analysis file  return\_status = gw.GW2RegisterAnalysisFile(session\_id,  StringToCBytes("C:\\temp\\Analysis\_output.xml"), 0); // 0 = AF\_XML    // run the session  return\_status = gw.GW2RunSession(session\_id);    // close the session  return\_status = gw.GW2CloseSession(session\_id);    }  catch (Exception error)  {  System.out.println("error - exception caught: " + error.getMessage());  }  }  } |