Glasswall SDK Documentation

Glasswall Solutions Ltd
Thur May 30 2019

Table of Contents

Glasswall API Overview

This documents the Application Programming Interface (API) within the Windows Dynamic Link Library `glasswall.classic.dll` and the Linux shared object `libglasswall.classic.so`.

The API provides the following functionality:

- The production of reports detailing deep analysis of documents
- The management of documents against a configurable content management policy
- The force regeneration of documents against a configurable content management policy
- The production of detailed process reporting
- The modification of the content management policy

Where an API argument type is `wchar_t`, the argument must be converted to a wide character type before passing in to the API function.

The code snippet below provides a framework showing how this might be done on a GNU/Linux system, where the default encoding of the file system is UTF-8 (which can be verified by using the `locale(1P)` command: `locale charmap` in a terminal window). For the purpose of the example below, a string literal in UTF-8 encoding has been used.

```
    /* Standard C headers */
#include <limits.h>
#include <locale.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

    /* platform dependent headers */
#include <iconv.h>

    /* define error handler to print message and exit */
#define handle_error(msg) \
        do { perror(msg); exit(EXIT FAILURE); } while (0)

int main(void)
{
    char path[PATH_MAX] = "testfile/test.doc";
    char *ppath = path;

    wchar t wcpath[PATH MAX * sizeof(wchar t)];
    wchar t * const pwcpath start = wcpath;  /* iconv() changes param values; preserve
start address */
    char *pwcpath = (char *)wcpath;

    size t bpathremain = strlen(ppath)+1;                /* bytes, path remaining; +1 to
include terminating null character */
    size_t bwcpathremain = (PATH_MAX) * sizeof(wchar_t); /* bytes, wide character path
remaining */
    size_t retval;
    eGwFileStatus rv;

        /* convert from UTF-8 to wchar t for API interface */
    iconv_t cd = iconv_open("WCHAR_T", "UTF-8");
    if (cd == (iconv_t)-1)
        handle_error("iconv_open() FAILED");

    /* Set the locale for LC CTYPE to the implementation-defined native locale
     * so the way the print and scan functions declared in <stdlib.h> and <stdio.h>
     * correctly parse and translate multibyte strings
     */
    setlocale(LC_CTYPE, "");

    /* convert the file name from UTF-8 string to wide character string */
```

```
    retval = iconv(cd, &ppath, &bpathremain, &pwcpath, &bwcpathremain);

    if (retval == (size t)-1)
        handle error("iconv() FAILED");

        /* set up the policy */
    rv = GWFileConfigXML(cmPolicy);
    if (rv != eGwFileStatus Success)
    {
        fprintf(stderr, "pGWFileConfigXML() FAILED with status %d\n", rv);
        exit(EXIT_FAILURE);
    }

        /* process the file */
    rv = GWFileToFileProtectAndReport(pwcpath start, L"doc", L"/tmp/generatedfile.doc",
L"/tmp/generatedfile.log");
    if (rv != eGwFileStatus_Success)
    {
        fprintf(stderr, "pGWFileToFileProtectAndReport() FAILED with status %d\n", rv);
        exit(EXIT FAILURE);
    }

        /* tidy up */
    rv = GWFileDone();
    if (rv != eGwFileStatus Success)
    {
        fprintf(stderr, "pGWFileDone() FAILED with status %d\n", rv);
        exit(EXIT_FAILURE);
    }

    exit(EXIT SUCCESS);
} /* end of function main */
```

Glasswall Configuration

Overview

The various document processing options (document analysis, document processing, process reporting) provided by the Glasswall library are predominantly available through the discrete functions of the API. In addition to these high-level processing options there are a number of additional processing options. These options enable third-party users to fine-tune the operation of the library to their specific needs and are described in the following sections.

Changes are made to the Glasswall configuration through the `Configuration Management API`.

Watermarking

This feature is only applicable to PDF documents

Watermarking enables a visible indication that the document has been processed by Glasswall. The specified text is added to a corner of each page within the document.

Content Management

`Content Management Policies` allow control of various document element types such as file attachments, executable code, interactive form content and a number of actions (e.g external links, execution of Javascripts). These document element types are known to be common attack vectors and when they are encountered within a document the `Content Management Policy` will define how Glasswall should process them. Each document type has its own `Content Management Policy`.

The active `Content Management Policy` can be updated on a document by document basis, but it **must** be set prior to Glasswall being able to process any document. In the event of the `Content Management Policy` not being set before processing documents, documents will be classified as 'non-conforming'.

Configuration Management API

The Glasswall Configuration Management API provides 3 principle operations

- Set the configuration (**GWFileConfigXML()**)
- Retrieve the current configuration(GWFileConfigGet())
- Reset the current configuration to default settings (**GWFileConfigRevertToDefaults()**)

A Glasswall API function **GWFileConfigXML()** is provided to enable the Glasswall library to be configured with a greater degree of flexibility. The API takes the required configuration as an XML null terminated string, as defined by the `Configuration Management XSD`.

GWFileConfigGet() can be called at any time to retrieve the current configuration in use by the Glasswall library. A call to **GWFileConfigGet()** after the Glasswall library has been loaded gives access to the available configuration items and their default values. If these default values are suitable, this XML string can be used to configure the library by passing it into the **GWFileConfigXML()** function.

It is not necessary to send a full configuration string to **GWFileConfigXML()**, only those configuration items that need to be changed are required.

Terminology

Content Management Policy

Content Management Policies are used to specify the set of `content management switches` that should be applied to a particular document type. In the `example below`, there are two policies specified, one for `pdfConfig` and one for `wordConfig`.

Content Management Switch

The content management switch is used to identify a document element type and associated action. In the `example below`, the full set of allowable content management switches for `pdfConfig` has been specified.

Content Management Switch Setting

The content management setting is used to specify what actions carried out by the Glasswall DLL for a particular content management switch. There are three available Content Management Switch Settings:

Switch Setting	Description
allow	Glasswall processes any associated document element types and they remain in the managed document
disallow	If any of the associated document element types are identified in a document, Glasswall identifies this document as being non-conforming, the presence of this document element type is logged in the Analysis report as an Issue Item.
sanitise	If any of the associated document element types are identified in a document, Glasswall removes them from the managed document, the removal is logged in the Analysis report as a Sanitisation Item.

In the `example below`, for content management policy `pdfConfig`:

- The `javascript` content management switch enables Javascript to be permitted in a managed document, if encountered.
- The `acroform` content management switch requires that any Acroform elements are flagged as issues, if encountered.
- The `internal_hyperlinks`, `external_hyperlinks`, `metadata`, `embedded_files` and `actions_all` content management switches ensure that any associated document elements are removed, if encountered.

In the `example below`, for content management policy `wordConfig`:

- The `metadata` content management switch enables Metadata to be permitted in a managed document, if encountered.

Configuration Management Example

The XML below is an example of a Glasswall Configuration string:

```
<?xml version="1.0" encoding="utf-8" ?>
<config>
    <pdfConfig>
        <watermark>Glasswall Approved</watermark>
        <javascript>allow</javascript>
        <acroform>disallow</acroform>
        <internal hyperlinks>sanitise</internal hyperlinks>
        <external_hyperlinks>sanitise</external_hyperlinks>
        <embedded_files>sanitise</embedded_files>
        <metadata>sanitise</metadata>
        <actions all>sanitise</actions all>
    </pdfConfig>
    <wordConfig>
        <metadata>allow</metadata>
    </wordConfig>
</config>
```

Glasswall Features

Determine File Type

Overview

The Glasswall library provides an API function known as *DetermineFileTypeFromFile* which provides the caller with the ability to determine the file type for a given file, provided that the file format is supported by Glasswall. This enables the user to determine and supply the Glasswall processing API functions with the correct file type for the file that is to be processed, regardless of the presence of a file extension or an incorrect extension.

Determine File Type API

The following link contains information on the API.

- **DetermineFileTypeFromFile()**

Glasswall DLL Lite Mode

Overview

The Glasswall library provides a lightweight version of the manage & protect APIs known as Glasswall DLL Lite. The purpose of Glasswall DLL Lite is to illustrate Glasswall's capability to generate a managed document sanitised of any macros. Glasswall DLL Lite APIs will utilise the configuration management policies that are in place for macros.

Glasswall DLL Lite APIs

The Glasswall DLL Lite APIs are as follows:

- **GWFileProtectLite()**
- **GWFileProtectLiteAndReport()**
- **GWFileToFileProtectLite()**
- **GWFileToFileProtectLiteAndReport()**

Glasswall DLL Lite Cameras

The following Glasswall DLL Lite cameras are available:

- Open Office XML.
- Word Binary.
- Excel Binary.
- PowerPoint Binary.

Glasswall Content Import/Export extensions

Overview

The Glasswall library provides facilities for image and text content to be extracted from files during processing for external processing and for image content to be re-inserted in its original location. This allows Glasswall to be used as a component of, or in conjunction with other parts of a Cross Domain Solution.

Glasswall Content Import/Export APIs

- **GWFileToFileAnalysisProtectAndExport()**
- **GWFileToFileProtectAndImport()**
- **GWFileToMemoryAnalysisProtectAndExport()**
- **GWFileToMemoryProtectAndImport()**

Glasswall XML Schema Definitions

Glasswall Library Configuration Management

A description on how this XML configuration is used can be found in the `Configuration Management API`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name ="watermarkType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="20"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="contentManagementFlag">
    <xs:restriction base="xs:string">
      <xs:enumeration value="sanitise"/>
      <xs:enumeration value="allow"/>
      <xs:enumeration value="disallow"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="pdfConfig">
    <xs:complexType>
      <xs:all>
        <xs:element name="watermark" type="watermarkType" minOccurs="0" maxOccurs="1" />
        <xs:element    name="metadata"    type="contentManagementFlag"    minOccurs="0"
maxOccurs="1" />
        <xs:element   name="javascript"   type="contentManagementFlag"   minOccurs="0"
maxOccurs="1" />
        <xs:element    name="acroform"    type="contentManagementFlag"    minOccurs="0"
maxOccurs="1" />
        <xs:element   name="actions_all"   type="contentManagementFlag"   minOccurs="0"
maxOccurs="1" />
        <xs:element  name="embedded files"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
        <xs:element       name="internal_hyperlinks"       type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
        <xs:element       name="external_hyperlinks"       type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
        <xs:element name="embedded images" type="contentManagementFlag" minOccurs="0"
maxOccurs="1" />
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="wordConfig">
    <xs:complexType>
      <xs:all>
        <xs:element     name="macros"     type="contentManagementFlag"     minOccurs="0"
maxOccurs="1" />
        <xs:element  name="embedded files"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
        <xs:element    name="metadata"    type="contentManagementFlag"    minOccurs="0"
maxOccurs="1" />
        <xs:element  name="review_comments"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
        <xs:element       name="internal hyperlinks"       type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
```

```
        <xs:element       name="external_hyperlinks"       type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
        <xs:element      name="dynamic data exchange"      type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
        <xs:element  name="embedded_images"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="pptConfig">
    <xs:complexType>
      <xs:all>
        <xs:element  name="embedded_files"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
        <xs:element    name="metadata"    type="contentManagementFlag"    minOccurs="0"
maxOccurs="1" />
        <xs:element    name="macros"    type="contentManagementFlag"    minOccurs="0"
maxOccurs="1" />
        <xs:element  name="review_comments"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
        <xs:element       name="internal hyperlinks"       type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
        <xs:element       name="external_hyperlinks"       type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
        <xs:element  name="embedded images"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="xlsConfig">
    <xs:complexType>
      <xs:all>
        <xs:element    name="macros"    type="contentManagementFlag"    minOccurs="0"
maxOccurs="1" />
        <xs:element  name="embedded_files"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
        <xs:element    name="metadata"    type="contentManagementFlag"    minOccurs="0"
maxOccurs="1" />
        <xs:element  name="review_comments"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
        <xs:element       name="internal_hyperlinks"       type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
        <xs:element       name="external hyperlinks"       type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
        <xs:element      name="dynamic_data_exchange"      type="contentManagementFlag"
minOccurs="0" maxOccurs="1" />
        <xs:element  name="embedded images"  type="contentManagementFlag"  minOccurs="0"
maxOccurs="1" />
      </xs:all>
    </xs:complexType>
  </xs:element>
  <xs:element name="config">
    <xs:complexType>
      <xs:all>
        <xs:element ref="pdfConfig" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="wordConfig" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="pptConfig" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="xlsConfig" minOccurs="0" maxOccurs="1"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Analysis Report

A description of this report can be found in the `Analysis Report Overview`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema                              targetNamespace="http://glasswall.com/namespace"
xmlns:mstns="http://glasswall.com/namespace"    xmlns="http://glasswall.com/namespace"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"   attributeFormDefault="qualified"
elementFormDefault="qualified">
  <!-- Definition of ContentGroup elements -->
  <xs:element name="ContentItem" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element  name="TechnicalDescription"  msdata:Prefix="gw"  type="xs:string"
minOccurs="1" maxOccurs="1"/>
        <xs:element    name="InstanceCount"    msdata:Prefix="gw"    type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element   name="TotalSizeInBytes"   msdata:Prefix="gw"   type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element   name="AverageSizeInBytes"  msdata:Prefix="gw"  type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element    name="MinSizeInBytes"    msdata:Prefix="gw"    type="xs:integer"
minOccurs="1" maxOccurs="1" />
        <xs:element    name="MaxSizeInBytes"    msdata:Prefix="gw"    type="xs:integer"
minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ContentItems" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element      ref="ContentItem"      msdata:Prefix="gw"      minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="itemCount" form="unqualified" type="xs:integer"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="SanitisationItem" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element  name="TechnicalDescription"  msdata:Prefix="gw"  type="xs:string"
minOccurs="1" maxOccurs="1"/>
        <xs:element    name="SanitisationId"    msdata:Prefix="gw"    type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element    name="InstanceCount"    msdata:Prefix="gw"    type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element   name="TotalSizeInBytes"   msdata:Prefix="gw"   type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element   name="AverageSizeInBytes"  msdata:Prefix="gw"  type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element    name="MinSizeInBytes"    msdata:Prefix="gw"    type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element    name="MaxSizeInBytes"    msdata:Prefix="gw"    type="xs:integer"
minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="SanitisationItems" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
```

```
        <xs:element     ref="SanitisationItem"     msdata:Prefix="gw"     minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="itemCount" form="unqualified" type="xs:integer" />
    </xs:complexType>
  </xs:element>
  <xs:element name="RemedyItem" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element  name="TechnicalDescription"  msdata:Prefix="gw"  type="xs:string"
minOccurs="1" maxOccurs="1"/>
        <xs:element    name="InstanceCount"    msdata:Prefix="gw"    type="xs:integer"
minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="RemedyItems" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element       ref="RemedyItem"       msdata:Prefix="gw"       minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="itemCount" form="unqualified" type="xs:integer" />
    </xs:complexType>
  </xs:element>
  <xs:element name="IssueItem" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element  name="TechnicalDescription"  msdata:Prefix="gw"  type="xs:string"
minOccurs="1" maxOccurs="1"/>
        <xs:element name="IssueId" msdata:Prefix="gw" type="xs:integer" minOccurs="1"
maxOccurs="1"/>
        <xs:element    name="InstanceCount"    msdata:Prefix="gw"    type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element name="RiskLevel" msdata:Prefix="gw" type="xs:string" minOccurs="1"
maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="IssueItems" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element       ref="IssueItem"       msdata:Prefix="gw"       minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="itemCount" form="unqualified" type="xs:integer" />
    </xs:complexType>
  </xs:element>
  <!-- Definition of ContentManagementPolicy elements -->
  <xs:element name="ContentSwitch" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="ContentName" msdata:Prefix="gw" type="xs:string" minOccurs="1"
maxOccurs="1"/>
        <xs:element     name="ContentValue"     msdata:Prefix="gw"     type="xs:string"
minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Camera" msdata:Prefix="gw">
```

```
    <xs:complexType>
      <xs:sequence>
        <xs:element      ref="ContentSwitch"      msdata:Prefix="gw"      minOccurs="1"
maxOccurs="100"/>
      </xs:sequence>
      <xs:attribute name="cameraName" form="unqualified" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <!-- Definition of ContentGroups elements -->
  <xs:element name="ContentGroup" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element   name="BriefDescription"   msdata:Prefix="gw"   type="xs:string"
minOccurs="1" />
        <xs:element ref="ContentItems" msdata:Prefix="gw" minOccurs="0" maxOccurs="1"/>
        <xs:element    ref="SanitisationItems"    msdata:Prefix="gw"    minOccurs="0"
maxOccurs="1"/>
        <xs:element ref="RemedyItems" msdata:Prefix="gw" minOccurs="0" maxOccurs="1"/>
        <xs:element ref="IssueItems" msdata:Prefix="gw" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="itemCount" form="unqualified" type="xs:integer" />
    </xs:complexType>
  </xs:element>
  <!-- Definition of ExternalData elements -->
  <xs:element name="Metadata" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element    name="MetadataName"    msdata:Prefix="gw"    type="xs:string"
minOccurs="1" maxOccurs="1"/>
        <xs:element    name="MetadataValue"    msdata:Prefix="gw"    type="xs:string"
minOccurs="1" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- Definition of Document Statistics elements -->
  <xs:element name="DocumentSummary" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element   name="TotalSizeInBytes"   msdata:Prefix="gw"   type="xs:integer"
minOccurs="1" maxOccurs="1"/>
        <xs:element name="FileType" msdata:Prefix="gw" type="xs:string" minOccurs="1"
maxOccurs="1"/>
        <xs:element name="Version" msdata:Prefix="gw" type="xs:string" minOccurs="0"
maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ContentManagementPolicy" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Camera" msdata:Prefix="gw" minOccurs="1" maxOccurs="100"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="ContentGroups" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element      ref="ContentGroup"      msdata:Prefix="gw"      minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
```

```
      <xs:attribute name="groupCount" form="unqualified" type="xs:integer" />
    </xs:complexType>
  </xs:element>
  <xs:element name="ExternalData" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Metadata" msdata:Prefix="gw" minOccurs="1" maxOccurs="100"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- Definition of GWallInfo elements -->
  <xs:element name="DocumentStatistics" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element     ref="DocumentSummary"     msdata:Prefix="gw"     minOccurs="1"
maxOccurs="1"/>
        <xs:element   ref="IssueInformationGroups"   msdata:Prefix="gw"   minOccurs="0"
maxOccurs="1"/>
        <xs:element   ref="ContentManagementPolicy"   msdata:Prefix="gw"   minOccurs="0"
maxOccurs="1"/>
        <xs:element ref="ContentGroups" msdata:Prefix="gw" minOccurs="1" maxOccurs="1"/>
        <xs:element     ref="ExtractedItems"     msdata:Prefix="gw"     minOccurs="1"
maxOccurs="1"/>
        <xs:element ref="ExternalData" msdata:Prefix="gw" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- Definition of IssueInformationGroups elements -->
  <xs:element name="IssueInformationGroups" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element  ref="IssueInformationGroup"  minOccurs="1"  maxOccurs="unbounded"
msdata:Prefix="gw"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- Definition of IssueInformationGroup elements -->
  <xs:element name="IssueInformationGroup" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="IssueIDRangeStart" type="xs:integer" msdata:Prefix="gw"/>
        <xs:element name="IssueIDRangeEnd"   type="xs:integer" msdata:Prefix="gw"/>
        <xs:element name="GroupDescription"  type="xs:string"  msdata:Prefix="gw"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- Definition of ExtractedItems elements -->
  <xs:element name="ExtractedItems" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element     ref="ExtractedItem"     minOccurs="0"     maxOccurs="unbounded"
msdata:Prefix="gw"/>
      </xs:sequence>
      <xs:attribute name="itemCount" form="unqualified" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <!-- Definition of ExtractedItem elements -->
  <xs:element name="ExtractedItem" msdata:Prefix="gw">
    <xs:complexType>
```

```
      <xs:sequence>
        <xs:element    ref="ExtractedImage"    minOccurs="0"    maxOccurs="unbounded"
msdata:Prefix="gw"/>
      </xs:sequence>
      <xs:attribute name="offset" form="unqualified" type="xs:string" />
      <xs:attribute name="itemIndex" form="unqualified" type="xs:string" />
      <xs:attribute name="location" form="unqualified" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <!-- Definition of ExtractedImage elements -->
  <xs:element name="ExtractedImage" msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Meta" minOccurs="0" maxOccurs="unbounded" msdata:Prefix="gw"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!-- Definition of Meta elements -->
  <xs:element name="Meta" msdata:Prefix="gw">
    <xs:complexType>
      <xs:attribute name="meta" form="unqualified" type="xs:string" />
    </xs:complexType>
  </xs:element>
  <xs:element    name="GWallInfo"    msdata:IsDataSet="true"    msdata:Locale="en-US"
msdata:Prefix="gw">
    <xs:complexType>
      <xs:sequence>
        <xs:element   ref="DocumentStatistics"   minOccurs="1"   maxOccurs="unbounded"
msdata:Prefix="gw"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Glasswall Analysis Report

Overview

The Glasswall library operates in one of two modes: analysis or protect. The analysis mode enables the content of documents to be assessed with a report generated to summarise the non-conformances and potential threats. The Protect mode addresses the issues that are identified, returning an updated document with the remedies and sanitisation changes necessary to remove any threat incorporated.

What follows is a file-type agnostic description of the data logged by Glasswall during document analysis. The analysis report is produced in XML. The structure of XML report is is defined by `Analysis Report XSD`.

Analysis Report Structure

gw:DocumentStatistics

Within the highest level XML node `gw:GWallInfo` is the `gw:DocumentStatistics` node. This encloses the data for the document being analysed. The document specific information is held within three sub-nodes:

- `gw:DocumentSummary`
- `gw:ContentManagementPolicy`
- `gw:ContentGroups`

An optional fourth node `gw:ExternalData` can exist for client applications. It is not populated by the Glasswall DLL

An extract from an analysis XML report for this section:

```
<gw:GWallInfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:gw="http://glasswall.com/namespace"xsi:schemaLocation="http://glasswall.com/name
space/GWallInfo.xsd">
    <gw:DocumentStatistics>
        <gw:DocumentSummary>...</gw:DocumentSummary>
        <gw:ContentManagementPolicy>...</gw:ContentManagementPolicy>
        <gw:ContentGroups groupCount="21">...</gw:ContentGroups>
    </gw:DocumentStatistics>
</gw:GWallInfo>
```

gw:DocumentSummary

This section provides the highest level of data in the analysis of the document. It includes:

- `gw:TotalSizeInBytes` which reports the size of the file being processed,
- `gw:FileType` which reports the type of file being processed and, where applicable,
- `gw:Version` which reports the version of the file being processed.

An extract from an analysis XML report for this section:

```
<gw:DocumentSummary>
    <gw:TotalSizeInBytes>4148344</gw:TotalSizeInBytes>
    <gw:FileType>pdf</gw:FileType>
    <gw:Version>PDF-1.6</gw:Version>
</gw:DocumentSummary>
```

gw:ContentManagementPolicy

The `gw:ContentManagementPolicy` section provides a snapshot of the relevant content management policy that was in place when the document was processed. It includes:

- `gw:ContentName` which reports the content management switch name for the policy, for example `GWFILE_MANAGE_REMOVE_JAVA_SCRIPT_ACTIONS`
- `gw:ContentValue` which reports the content management switch setting for the policy.

An extract from an analysis XML report for this section:

```
<gw:ContentManagementPolicy>
    <gw:Camera cameraName = "pdfConfig">
        <gw:ContentSwitch>
            <gw:ContentName>javascript</gw:ContentName>
            <gw:ContentValue>sanitise</gw:ContentValue>
        </gw:ContentSwitch>
        <gw:ContentSwitch>
            <gw:ContentName>acroform</gw:ContentName>
            <gw:ContentValue>sanitise</gw:ContentValue>
        </gw:ContentSwitch>
        <gw:ContentSwitch>
            <gw:ContentName>embedded files</gw:ContentName>
            <gw:ContentValue>sanitise</gw:ContentValue>
        </gw:ContentSwitch>
        <gw:ContentSwitch>
            <gw:ContentName>metadata</gw:ContentName>
            <gw:ContentValue>sanitise</gw:ContentValue>
        </gw:ContentSwitch>
        <gw:ContentSwitch>
            <gw:ContentName>actions_all</gw:ContentName>
            <gw:ContentValue>sanitise</gw:ContentValue>
        </gw:ContentSwitch>
        <gw:ContentSwitch>
            <gw:ContentName>internal hyperlink</gw:ContentName>
            <gw:ContentValue>sanitise</gw:ContentValue>
        </gw:ContentSwitch>
        <gw:ContentSwitch>
            <gw:ContentName>external_hyperlink</gw:ContentName>
            <gw:ContentValue>sanitise</gw:ContentValue>
        </gw:ContentSwitch>
    </gw:Camera>
</gw:ContentManagementPolicy>
```

gw:ContentGroups

The `gw:ContentGroups` node is a collection of `gw:ContentGroup` nodes. An attribute, `groupCount`, is associated with the node to record how many `gw:ContentGroup` nodes are in the collection.

An extract from an analysis XML report for this section:

```
<gw:ContentGroups groupCount="21">
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
```

```
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
    <gw:ContentGroup>...</gw:ContentGroup>
</gw:ContentGroups>
```

gw:ContentGroup

A content group is a category of document element, or section of a document. It can be defined at any level of abstraction and enables statistics to be gathered, and problems associated, in a manner that is sensible for the type of document being processed. The content group consists of:

- `gw:BriefDescription` which provides a description of the information that is associated with this content group
- A collection of sub-nodes each with a collection of data items. Each of these data items provides a 'Technical Description' explaining the reason for its existence and a counter recording the number of instances of that data item found in the document.

An extract from an analysis XML report for this section:

```
</gw:ContentGroup>
    <gw:BriefDescription>PDF document has Basic File Section structure
instances</gw:BriefDescription>
    <gw:ContentItems itemCount="5">...</gw:ContentItems>
    <gw:SanitisationItems itemCount="1">...</gw:SanitisationItems>
    <gw:RemedyItems itemCount="1">...</gw:RemedyItems>
    <gw:IssueItems itemCount="0"/>
</gw:ContentGroup>
```

The data stored in each of the sub-nodes is defined in the following sections.

gw:ContentItems

The `gw:ContentItems` node is a collection of `gw:ContentItem` nodes that log the checks associated with the enclosing `gwContentGroup`. Metrics are logged against each content item. Associated with the `gw:ContentItems` node is a set of statistics summarising the collection of `gw:ContentItem` nodes:

- `itemCount` records how many content items are in the collection.
- `TechnicalDescription` identifies the content item.
- `InstanceCount` records how many instances of this item occur in the document.
- `TotalSizeInBytes, AverageSizeInBytes, MinSizeInBytes, MaxSizeInBytes` summarise information associated with the size of the document element being reported upon.

An extract from an analysis XML report for this section:

```
<gw:ContentItems itemCount="5">
<gw:ContentItem>
    <gw:TechnicalDescription>PDF Header Instances</gw:TechnicalDescription>
    <gw:InstanceCount>1</gw:InstanceCount>
    <gw:TotalSizeInBytes>16</gw:TotalSizeInBytes>
    <gw:AverageSizeInBytes>16</gw:AverageSizeInBytes>
    <gw:MinSizeInBytes>16</gw:MinSizeInBytes>
    <gw:MaxSizeInBytes>16</gw:MaxSizeInBytes>
</gw:ContentItem>
<gw:ContentItem>...</gw:ContentItem>
<gw:ContentItem>...</gw:ContentItem>
<gw:ContentItem>...</gw:ContentItem>
<gw:ContentItem>...</gw:ContentItem>
</gw:ContentItems>
```

gw:SanitisationItems

The `gw:SanitisationItems` node is a collection of `gw:SanitisationItem` nodes that log any sanitisation that is carried out within the enclosing `gwContentGroup`. An attribute of the node indicates how many `SanitisationItem` nodes are in the collection.

- `itemCount` reports how many sanitisation items are in the collection.
- `gw:TechnicalDescription` identifies the item subjected to sanitisation.
- `gw:SanitisationId` identifier for the item that that uniquely identifies this type of sanitisation.
- `gw:InstanceCount` reports the occurrence of items of this type of sanitisation that has been detected within the document.
- `TotalSizeInBytes, AverageSizeInBytes, MinSizeInBytes, MaxSizeInBytes` summarise information associated with the size of the document element being reported upon.

An extract from an analysis XML report for this section:

```
<gw:SanitisationItems itemCount="1">
<gw:SanitisationItem>
    <gw:TechnicalDescription>Pdf Object Instances</gw:TechnicalDescription>
    <gw:SanitisationId>16</gw:SanitisationId>
    <gw:InstanceCount>78</gw:InstanceCount>
    <gw:TotalSizeInBytes>1485875</gw:TotalSizeInBytes>
    <gw:AverageSizeInBytes>19049</gw:AverageSizeInBytes>
    <gw:MinSizeInBytes>80</gw:MinSizeInBytes>
    <gw:MaxSizeInBytes>328967</gw:MaxSizeInBytes>
</gw:SanitisationItem>
</gw:SanitisationItems>
```

gw:RemedyItems

The `gw:RemedyItems` node is a collection of `gw:RemedyItem` nodes that logs any remedial work that is carried out within the enclosing `gwContentGroup`. An attribute of the node indicates how many `gw:RemedyItem` nodes are in the collection.

- `itemCount` reports how many remedy items are in the collection.
- `gw:TechnicalDescription` identifies the item subjected to remedial work.
- `gw:InstanceCount` reports the occurrence of items of this type of remedy that has been detected within the document.

An extract from an analysis XML report for this section:

```
<gw:RemedyItems itemCount="1">
<gw:RemedyItem>
    <gw:TechnicalDescription>Remedy and insert missing Cross Reference Table
entries.</gw:TechnicalDescription>
    <gw:InstanceCount>1</gw:InstanceCount>
</gw:RemedyItem>
</gw:RemedyItems>
```

gw:IssueItems

The `gw:IssueItems` node is a collection of `gw:IssueItem` that log the checks associated with the enclosing `gwContentGroup`. An attribute of the node indicates how many `gw:IssueItems` nodes are in the collection.

- `itemCount` reports how many issue items are in the collection.
- `gw:TechnicalDescription` identifies the item that has been detected as an issue.
- `gw:IssueId` identifier for the item that uniquely identifies this type of issue.
- `gw:InstanceCount` reports the occurrence of items of this type of issue that has been detected within the document.

An extract from an analysis XML report for this section:

```
<gw:IssueItems itemCount="1">
<gw:IssueItem>
    <gw:TechnicalDescription>A marked content properties list is present in a resource
dictionary.</gw:TechnicalDescription>
    <gw:IssueId>4</gw:IssueId>
    <gw:InstanceCount>8</gw:InstanceCount>
    <gw:RiskLevel>High</gw:RiskLevel>
</gw:IssueItem>
</gw:IssueItems>
```

gw:ExtractedItems

The `gw:ExtractedItems` node is a collection of `gw:ExtractedItem` nodes. An attribute, `itemCount`, is associated with the node to record how many `gw:ExtractedItem` nodes are in the collection.

An extract from an analysis XML report for this section:

```
<gw:ExtractedItems itemCount="3">
    <gw:ExtractedItem>...</gw:ExtractedItem>
    <gw:ExtractedItem>...</gw:ExtractedItem>
    <gw:ExtractedItem>...</gw:ExtractedItem>
</gw:ExtractedItems>
```

gw:ExtractedItem

An extracted item node logs the details of each item extracted from the document.

- `gw:ExtractedImage` which provides the metadata information of the extracted image

An extract from an analysis XML report for this section:

```
<gw:ExtractedItem>
    <gw:ExtractedImage>
        <gw:Meta meta="WMF" />
        </gw:ExtractedImage>
</gw:ExtractedItem>
```

gw:ExternalData

This node is not populated by the Glasswall DLL. This node is provided within the specification of the report to enable client applications to add document specific information to the report, and have it remain within specification.

This node is a collection of `gw:MetaData` nodes. The MetaData node consists of a `gw:MetaDataName` field and a `gw:MetaDataValue` field, both of which contain string types. The number of `gw:Metadata` nodes is arbitrarily limited to 100. The use of these fields is outside the scope of this document.

An example of an occurrence of this section in an an analysis XML report:

```
<gw:ExternalData>
<gw:Metadata>
    <gw:MetadataName>primum item de notita</gw:MetadataName>
    <gw:MetadataValue>quidam valor</gw:MetadataValue>
</gw::Metadata>
<gw:Metadata>
    <gw:MetadataName>item alios magis notitia</gw:MetadataName>
    <gw:MetadataValue>0123456</gw:MetadataValue>
</gw::Metadata>
</gw:ExternalData>
```

File Specific Analysis

- **File Specific Analysis - PDF**
- **File Specific Analysis - GIF**
- **File Specific Analysis - JPEG**
- **File Specific Analysis - PNG**
- **File Specific Analysis - Office Binary**
- **File Specific Analysis - Office Open XML**
- **File Specific Analysis - EMF**
- **File Specific Analysis - WMF**

File Specific Analysis - PDF

gw:DocumentSummary

Node Name	Description
gw:TotalSizeInBytes	The size of the file passed in for processing
gw:FileType	“pdf”, passed in through the Glasswall API
gw:Version	Read from the document’s Data Stream Header

gw:ContentGroups

The contents groups for PDF fall into two broad categories:

File Structure

Group	Description
PDF Basic Object Structure	This content group reports on the basic PDF types: - ARRAY_end - ARRAY - BOOLEAN - COMMENT - DICTIONARY_end - DICTIONARY - NAME - NULL - NUMERIC_INTEGER - NUMERIC_REAL - POSTSCRIPT_EXPRESSION_end

	- POSTSCRIPT_EXPRESSION - POSTSCRIPT_OPERATOR_ARITHMETIC - POSTSCRIPT_OPERATOR_CONDITIONAL - POSTSCRIPT_OPERATOR_RELATIONAL - POSTSCRIPT_OPERATOR_STACK - REFERENCE - STREAM_end - STREAM_GRAPHICS_CMD_PAYLOAD - STREAM_GRAPHICS_CMD - STREAM - STRING_HEX - STRING_LITERAL - UNSPECIFIED
PDF Compressed XRef Structure	This content group reports on Compressed Cross Reference Table
PDF File Section Structure	This content group reports on the high level structure of the document: - PDF Comment Section Instances - PDF Cross Reference Table Instances - PDF Header Instances - PDF Object Instances - PDF StartXRef Section Instances - PDF Trailer Section Instances
PDF Graphic Content Stream Structure	This content group reports on Content Stream Structure
PDF Object Stream Structure	This content group reports on Object Stream Structure
PDF Trailer Structure	Not used
PDF XRef Structure	Not used

File Content

Group	Description
Common PDF Data Structures	Data structures for text strings, dates, rectangles, name trees, and number trees.
PDF Annotations Data	Associates objects such as notes, sound or movies with a location on page
PDF ASCII Filter Stream	Decodes data encoded in an ASCII hexadecimal representation, reproducing the original binary data..
PDF ASCII85 Filter Stream	Data encoded in an ASCII base-85 representation, reproducing the original binary data.
PDF Catalogue Data	References to other objects defining the document's contents, outline, article threads, named destinations, and other attributes.
PDF CCIT Filter Stream	Decompresses data encoded using the CCITT facsimile standard, reproducing the original data
PDF Colour Space Data	Specifies colors of graphics objects to be painted on the current page.
PDF Crypt Filter Stream	Decrypts data encrypted by a security handler,

	reproducing the original data as it was before encryption
PDF DCT Filter Stream	Decompresses data encoded using a DCT (discrete cosine transform) technique based on the JPEG standard, reproducing image sample data that approximates the original data.
PDF Embedded Font	A font program can be embedded in a PDF file as data contained in a PDF stream object. Such a stream object is also called a font file by analogy with font programs that are available from sources external to the consumer application.
PDF Filter Predictor	Transformation by a predictor function, which improves the compression of sampled image data.
PDF Font Data	A font is represented in PDF as a dictionary specifying the type of font, its PostScript name, its encoding, and information that can be used to provide a substitute when the font program is not available. Optionally, the font program can be embedded as a stream object in the PDF file.
PDF Function Data	Function objects that represent parameterized classes of functions, including mathematical formulas and sampled representations with arbitrary resolution.
PDF Graphic Content Stream	Sequence of graphics objects to be painted on the page.
PDF Graphic State Data	A PDF consumer application maintains an internal data structure called the graphics state that holds current graphics control parameters. These parameters define the global framework within which the graphics operators execute. For example, the f (fill) operator implicitly uses the current color parameter, and the S(stroke) operator additionally uses the current line width parameter from the graphics state.
PDF Halftone Data	Data related to rendering grayscale elements on a bilevel device.
PDF Huffman Filter Stream	Decompresses data encoded using the zlib/deflate compression method, reproducing the original text or binary data.
PDF JBIG Filter Stream	Decompresses data encoded using the JBIG2 standard, reproducing the original monochrome
PDF JPX Filter Stream	Decompresses data encoded using the wavelet-based JPEG2000 standard, reproducing the original image data.
PDF LZW Filter Stream	Decompresses data encoded using the LZW (Lempel-Ziv-Welch) adaptive compression method, reproducing the original text or binary data.

PDF Name Tree Data	Serves a similar purpose to a dictionary—associating keys and values—but by different means.
PDF Number Tree Data	A number tree is similar to a name tree, except that its keys are integers instead of strings and are sorted in ascending numerical order.
PDF Optional Content Data	Sections of content in a PDF document that can be selectively viewed or hidden by document authors or consumers.
PDF Outline Data	A PDF document may optionally display a document outline on the screen, allowing the user to navigate interactively from one part of the document to another. The outline consists of a tree-structured hierarchy of outline items (sometimes called bookmarks), which serve as a visual table of contents to display the document's structure to the user.
PDF Page Tree Data	The pages of a document are accessed through a structure known as the page tree, which defines the ordering of pages in the document.
PDF Pattern Data	Data relating to items such as a repeating figure or smooth gradient is called a pattern.
PDF Resource Data	The operands supplied to operators in a content stream may only be direct objects; indirect objects and object references are not permitted. In some cases, an operator needs to refer to a PDF object that is defined outside the content stream, such as a font dictionary or a stream containing image data. This can be accomplished by defining such objects as named resources and referring to them by name from within the content stream.
PDF Structure Tree Data	The logical structure of a document is described by a hierarchy of objects called the structure hierarchy or structure tree.
PDF Threads Data	Some types of documents may contain sequences of content items that are logically connected but not physically sequential. For example, a news story may begin on the first page of a newsletter and run over onto one or more nonconsecutive interior pages. To represent such sequences of physically discontiguous but logically related items, a PDF document may define one or more articles. The sequential flow of an article is defined by an article thread; the individual content items that make up the article are called beads on the thread. PDF viewer applications can provide navigation facilities to allow the user to follow a thread from one bead to the next.
PDF Trailer Data	The trailer of a PDF file enables an application reading the file to quickly find the cross-reference table and

	certain special objects.
PDF XObject Data	A transparency group is represented in PDF as a special type of group XObject called a transparency group XObject.
PDF Web Capture Data	Web Capture is a PDF 1.3 feature that allows information from Internet-based or locally resident HTML, PDF, GIF, JPEG, and ASCII text files to be imported into a PDF file.

File Specific Analysis - GIF

gw:DocumentSummary

Node Name	Description
gw:TotalSizeInBytes	The size of the file passed in for processing
gw:FileType	"gif", passed in through the Glasswall API
gw:Version	Read from the document's Data Stream Header

gw:ContentGroups

Group	Description
GIF Colourtable	Utilised to render raster-based graphics
GIF Extension Block Application XMP	Data consisting of XMP (Extensible Metadata Platform) packets
GIF Extension Block Application	Application identifier
GIF Extension Block	Control, graphic-rendering and special purpose blocks
GIF Image Descriptor	Contains parameters to process a table bsed image
GIF Local Colour Table	Associated with graphics that immediately follows. Supercedes global colour table
GIF Logical Screen Descriptor	Parameters defininging area of display device
GIF Table Based Image Data	Image data for a table based image

File Specific Analysis - JPEG

gw:DocumentSummary

Node Name	Description
gw:TotalSizeInBytes	The size of the file passed in for processing
gw:FileType	"jpg", passed in through the Glasswall API
gw:Version	Not applicable for this document type.

gw:ContentGroups

Group	Description
JPEGFileSectionMarker	SOI, EOI
JPEGFrameData	Captures document information associated with Start of Frame markers SOF0..SOF15
JPEGProcessData	Captures document processing markers DAC, DHT, DQT, DNL, DRI, DHP, EXP, APPn, JPGn, COM, TEM, RES
JPEGScanData	Captures scan marker information SOS, DRI, RSTm

File Specific Analysis - PNG

gw:DocumentSummary

Node Name	Description
gw:TotalSizeInBytes	The size of the file passed in for processing
gw:FileType	“png”, passed in through the Glasswall API
gw:Version	Not applicable for this document type.

gw:ContentGroups

Group	Description
BKGD	Miscellaneous information - background colour
CHRM	Colour space information
CMPP	Private chunk
FRAC	Fractal parameters
GAMA	Colour space information
GIFG	GIF conversion information
GIFX	GIF conversion information
HIST	Miscellaneous information - histogram
ICCP	Colour space information
IDAT	Image data chunk
IEND	Image trailer, which is the last chunk in a PNG stream
IHDR	Image header
MKBF	Private chunk
MKBS	Private chunk
MKBT	Private chunk
MKTS	Private chunk
OFFS	Image offset
PCAL	Pixel calibration
PHYS	Miscellaneous information - physical pixel dimension
PLTE	Palette table associated with indexed PNG images
PRVW	Private chunk
SBIT	Colour space information - significant bits
SCAL	Physical scale
SPLT	Miscellaneous information - suggested palette
SRGB	Colour space information
TEXT	Textual information - Latin 1 text annotations

TIME	Time stamp information
TRNS	Transparency information
ZTXT	Textual information - Latin 1 text annotations

File Specific Analysis - Office Binary

gw:DocumentSummary

Node Name	Description
gw:TotalSizeInBytes	The size of the file passed in for processing
gw:FileType	“doc”, “xls” or “ppt”, passed in through the Glasswall API
gw:Version	Read from the document’s appropriate stream

gw:ContentGroups Common

The table below describes the common items across each of the office products Word, Excel & Powerpoint.

Group	Description
CFB_Header	Compound file binary package header
CFB Directory	Compound file binary directory details
LZ77 INFLATE	LZ77 decompression information
OLE COMPOBJ	Object linking and embedding data structures
OLE CURRENT USER	Current user information
OLE DOCUMENT SUMMARY INFORMATION	Metadata properties and information
OLE META	OLE object metadata
OLE OBJ INFO	OLE object information
OLE SUMMARY INFORMATION	Metadata properties and information

gw:ContentGroups Word

Group	Description
Auto Language Detection	Auto Language Detection data structures
Auto Summary	Auto Summary data structures
Auto Text	Auto text data structures
Bookmarks Annotation Bookma	Bookmark data structures
Bookmarks Format Consistency Checker	Bookmark data structures
Bookmarks Range Level Protection Bookmark	Bookmark data structures
Bookmarks Repair BookMark	Bookmark data structures
Bookmarks Standard	Bookmark data structures
Bookmarks Structured Document Tag Bookmark	Bookmark data structures
Comments	Review comment data structures
Content Stream	Content stream data structures

Custom XML	Custom XML storage and properties data structures
Customization Command	Customization Command properties
Document Information	Metadata data information and properties
Document Properties	Metadata data information and properties
Email Data	Email information properties
Embedded Document	Embedded document properties
Endnote Document	Endnote document properties
External File Reference	External file reference information
File Information Block	Central data structure information source
Font Data	Font data properties and information
Footnote Document	Document footnote data structures
Grammar Checker	Grammar check settings
Hyperlink or Form Data	Hyperlink and form information data structures
Main Document	Document properties
Object Linking and Embedding	OLE object data structures
Office Art Drawing	Data structures containing images and Office drawing properties
Printer Setting	Printer settings
Revisions	Document revision data structures

gw:ContentGroups Excel

Group	Description
AutoFilter	AutoFilter property and information
Axis Group	Axis label properties in excel charts
Chart Information	Excel chart information data structures
Data Source Information	Data Source Information
Data Table	Data table properties
Data Validation	Excel data validation structures
Drawing Objects	Drawn object properties
Formatting	Excel formatting information
Formulas	Excel formula properties
Functions	Excel function properties
Hyperlink	Hyperlink properties
Metadata	Metadata properties
Office Art Drawing	Data structures containing images and Office drawing properties
Pivot Table Information	Pivot Table Information
Printer Setting	Printer settings
Query Table	Query table properties
Routing Slip	Routing slip information
Shared Workbook Informatio	Worksheet information
Sheet Information	Spreadsheet information
String Constants	String constant properties
Structure Instances	General data strcutures used across several groups
Unknown Group	Unknown or unrecognised data structures
User Interface	User interface properties
Version	Version number properties
Workbook Protection	Password protection properties

gw:ContentGroups PowerPoint

Group	Description
Animation Types	Data strcutures for Animation properties
Basic Types	Basic PowerPoint data structures
Broadcast Types	Presentation broadcast data strcutures
Comment Author Types	Data structures containing information on the Authors that create presentation comments
Common Types	Commonly used data structures
Document Comparison Types	Data structures used for the comparisons of states across PowerPoint
Document Tag Info Types	Data structures for programmable tags
Document Types	Data strcutures that contain infromation about the document
Encrypted Types	Data structures containing file encryption information
External Objects	OLE Object data structures
Header and footers	Header and footer data structures
HTML Publish Types	Data structures containing infromation for how to publish a document as a Web page
Office Art Drawing	Data structures containing images and Office drawing properties
Other Types	Other general data structures
PICTURE Instances	Image data structures
Printer Settings	Printer settings data structures
Shape Types	Data structures containing shape information
Slide List Types	Data structures containing presentation slide information
Slide Show Types	Data structures containing presentation slide show information
Slide Types	Data structures containing presentation slide information
Sound Types	Data strcutures containing sound information
Structure Instances	General data strcutures used across several groups
Summary Info Types	Data strcuture containing metadata information
Text Types	Data strcutures that contain text or text formatting properties
View Info Types	Data structures containing dsiplay property information
Visual Basic Data	Visual basic property information

File Specific Analysis - Office Open XML

gw:DocumentSummary

Node Name	Description
gw:TotalSizeInBytes	The size of the file passed in for processing
gw:FileType	“docx”, “xlsx” or “pptx”, passed in through the Glasswall API
gw:Version	Read from the document’s appropriate stream

gw:ContentGroups

The table below describes the groups across each of the office Open XML products Word, Excel & Powerpoint. The majority of the descriptions can be found in `http://officeopenxml.com/anatomyofOOXML.php`

Group
3D Object
Additional Document Characteristics
Animation
Annotations
Audio
AutoFilter Settings
Bibliography
Bookmarks
ByteOrder
Colors
Content Group
Coordinate System
Custom Properties
Custom XML and Smart Tags
Custom XML Mappings
Custom XML Properties
CustomXML
Document Section Group
Document Settings
Drawing Object Information
DrawingML Common Types
Embedded
Excel Common Types
Extended File Properties
External Data Connections
External Dependendancies
Fields and Hyperlinks
fill Group
Fonts
Footnotes and Endnotes
Formatting Group
Glossary Document
Headers and Footer
Language Settings
Locked Canvas
Mail Merge
Main Document Group
Math
Metadata
Numbering
Paragraphs and Rich Formatting
Pivot Tables

Powerpoint Common Types
Presentation Properties
Programmable Tags
QueryTable Data
Range Permissions
Revisions
Shape Definitions and Attributes
Shape Effects and Properties
Shared String Table
Shared Style Sheet
Shared Workbook Data
ShareML Common Types
Slide Synchronization Data
Spelling and Grammar
Structured Document Tags
Styles
Supplementary Workbook Data
Table Group
Table Measurement
Table Styles
Tables
Video
View Properties
Web Page Settings
Word Common Types
Workbook
Worksheets
EXCEL DOCUMENT
POWERPOINT DOCUMENT
WORD DOCUMENT

File Specific Analysis - EMF

gw:DocumentSummary

Node Name	Description
gw:TotalSizeInBytes	The size of the file passed in for processing
gw:FileType	“emf", passed in through the Glasswall API
gw:Version	Read from the document’s appropriate stream

gw:ContentGroups

Group	Description
Version	Version information
Bitmaps	Emf Bitmaps Instances
Clippings	Emf Clippings Instances
Comments	Emf Comments Instance

File Properties	Emf File Property Instances
Drawings	Emf Drawings Instances
Escape Functions	Emf Escape Functions Instances
Graphics Object Creation	Emf Graphics Object Creation Instances
Graphics Objects Manipulation	Emf Graphics Objects Manipulation Instances
OpenGL Functions	Emf OpenGL Functions Instances
State Definitions	Emf State Definitions Instances
Image Transformations	Emf Image Transformations Instances
Graphics Objects	Emf Graphics Objects Instances
Path Records	Emf Path Records Instances
Common Types	Emf Common Types Instances
Graphics Objects	Emf Plus Graphics Objects Instances
Common	Emf Plus Common Instances
Image Effects	Emf Plus Image Effects Instances
Clippings	Emf Plus Clippings Instances
Comments	Emf Plus Comments Instances
File Control	Emf Plus File Control Instances
Drawings	Emf Plus Drawings Instances
Graphics Properties	Emf Plus Graphics Properties Instances
State Definitions	Emf Plus State Definitions Instances
Terminal Servers	Emf Plus Terminal Servers Instances
Image Transformations	Emf Plus Image Transformations Instances

File Specific Analysis - WMF

gw:DocumentSummary

Node Name	Description
gw:TotalSizeInBytes	The size of the file passed in for processing
gw:FileType	“emf", passed in through the Glasswall API
gw:Version	Read from the document’s appropriate stream

gw:ContentGroups

Structure Instances	WMF Document File Structure Instances
Version	WMF Document Version Instances
Common Types	WMF Document Common Types
Bitmap Types	WMF Document Bitmap Types
File Control	WMF Document File Control Types
Drawings	WMF Document Drawings Instances
Graphics Objects Creation	WMF Document Graphics Objects Creation Instances
Graphics Objects Manipulation	WMF Document Graphics Object Manipulation Instances
Graphics Objects	WMF Document Graphics Objects Instances
State Definitions	WMF Document State Definitions Instances
Escape Functions	WMF Document Escape Functions Instances

File Specific Analysis - BMP

gw:ContentGroups

TBD

File Specific Analysis - TIFF

gw:ContentGroups

TBD

File Specific Analysis - MPG

gw:ContentGroups

TBD

File Specific Analysis - WAV

gw:ContentGroups

TBD

File Specific Analysis - MP3

gw:ContentGroups

TBD

File Specific Analysis - MP4

gw:ContentGroups

TBD

File Specific Analysis - Mach-O

gw:ContentGroups

TBD

File Specific Analysis - PE

gw:ContentGroups

TBD

File Specific Analysis - ELF

gw:ContentGroups

TBD

Module Index

Modules

Here is a list of all modules:

Module Documentation

Glasswall Library

Modules

- **Dependencies**
- **Glasswall Configuration**
- **Glasswall Document Processing**
- **Glasswall Resource Management**

Functions

- wchar_t * **GWFileVersion** (void)

Detailed Description

The interface for the Glasswall API is broken into three main sections covering the principle functionality provided by the library

- **Glasswall Configuration**
- **Glasswall Document Processing**
- **Glasswall Resource Management**

Function Documentation

wchar_t* GWFileVersion (void)

This function retrieves the version string of the Glasswall Library.

Returns:

a pointer to a printable Null-terminated string.

Dependencies

Glasswall is implemented for 64 bit architectures on GNU/Linux and Windows systems. The versions supported are detailed below.

Windows

The Glasswall application has been successfully tested on the following Windows versions:

- Windows 7
- Windows 10
- Windows Server 2008
- Windows Server 2012

The C++ 2017 Redistributable Package (x64) must be installed.

Linux

The Glasswall application has been successfully tested on the following GNU/Linux distributions:

- Red Hat Enterprise Linux 6.9
- Red Hat Enterprise Linux 7.2
- Red Hat Enterprise Linux 7.3

glibc-2.12 or later must be installed. freetype 2.7 or later must be installed.

Glasswall Configuration

Functions

- int **GWFileConfigXML** (wchar_t *xmlstring)
- int **GWFileConfigGet** (wchar_t **configurationBuffer, size_t *outputLength)
- int **GWFileConfigRevertToDefaults** (void)
- int **GWGetIdInfo** (uint32_t issueId, size_t *bufferLength, void **outputBuffer)
- int **GWGetAllIdInfo** (size_t *bufferLength, void **outputBuffer)

Detailed Description

These functions enable the operation of the Glasswall processing to be configured. Glasswall library Configuration can be updated using **GWFileConfigXML()**.

Functions are provided to retrieve the current configuration, and to reset the current configuration to Glasswall default values.

Glasswall is single threaded.

Function Documentation

int GWFileConfigGet (wchar_t ** *configurationBuffer*, size_t * *outputLength*)

This function retrieves the current Glasswall library configuration via an XML string that conforms to the `Configuration Management XSD`.

In the event of an error configurationBuffer will be returned as a NULL pointer and outputLength will be set to zero.

Parameters:

out	*configurationBuffer*	Location to which the configuration should be written.
out	*outputLength*	Size of the configuration buffer being returned in bytes.

Returns:

eGwFileStatus

int GWFileConfigRevertToDefaults (void)

This function reverts the Glasswall library's active configuration to the hard-coded default values, which is everything set to allow. This function removes any changes introduced by calls to **GWFileConfigXML()**.

Returns:

eGwFileStatus

This example code demonstrates the library configuration being reset to default values, which are then retrieved and printed by the client application.

```
// configuration buffer and length
wchar t *configurationBuffer     = NULL;
size_t configurationBufferLength = 0;

if (eGwFileStatus_Success == GWFileConfigRevertToDefaults())
{
    if(eGwFileStatus Success == GWFileConfigGet(&configurationBuffer,
&configurationBufferLength))
```

```
    {
        printLibraryConfig(configurationBuffer, configurationBufferLength);
    }
    else
    {
        // error handling
    }
}
else
{
    // error handling
}

// release resources including the global buffer 'fileImage'
GWFileDone();
```

int GWFileConfigXML (wchar_t * *xmlstring*)

This function applies the given content management settings to the Glasswall library, which are later used when processing files. The content management settings are specified as a XML string that conforms to the `Configuration Management XSD`.

This function should be called before any documents are processed. If this function is not called, then the content management settings will be left in their default state where everything is set to allow. Content management switches that have not been specified will not be updated and will remain in their existing state.

Parameters:

in	*xmlstring*	configuration string.

Returns:

eGwFileStatus

This example code demonstrates the required library configuration string being collected by the client application and used to configure the Glasswall library prior to processing a PDF document.

```
// managed file output buffer and length
void *fileImage        = NULL;
size t fileImageLength = 0;

// Library configuration
wchar_t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus Success == GWFileConfigXML(cmPolicy))
{
    if(eGwFileStatus_Success == GWFileProtect(L"C:\\data\\temp1.pdf", L"pdf",
&fileImage, &fileImageLength))
    {
        // process the managed file image
    }
}
else
{
    // error handling
}

// release resources including the global buffer 'fileImage'
GWFileDone();
```

int GWGetAllIdInfo (size_t * *bufferLength*, void ** *outputBuffer*)

This function returns a buffer containing an XML file populated with issue ID description for all issue ID ranges.

```
<gw:IssueInformationGroups>
   <gw:IssueInformationGroup>
       <gw:IssueIDRangeStart>100728832</gw:IssueIDRangeStart>
        <gw:IssueIDRangeEnd>100794367</gw:IssueIDRangeEnd>
       <gw:GroupDescription>Enhanced Metafile Format Instances</gw:GroupDescription>
   </gw:IssueInformationGroup>
   <gw:IssueInformationGroup>
       <gw:IssueIDRangeStart>100859904</gw:IssueIDRangeStart>
       <gw:IssueIDRangeEnd>100925439</gw:IssueIDRangeEnd>
       <gw:GroupDescription>Emf Bitmaps Instances</gw:GroupDescription>
   </gw:IssueInformationGroup>
...
</gw:IssueInformationGroups>
```

Parameters:

out	*outputBuffer*	The output file buffer containing the XML, which is encoded in ASCII.
out	*bufferLength*	Length of the output buffer that has been allocated in bytes.

Returns:

eGwFileStatus processing status

int GWGetIdInfo (uint32_t *issueId*, size_t * *bufferLength*, void ** *outputBuffer*)

This function returns a buffer containing the description for the given issue ID.

Parameters:

in	*issueId*	Issue ID value
out	*outputBuffer*	The output buffer containing description string, which is encoded in ASCII.
out	*bufferLength*	Length of the the output buffer that has been allocated in bytes.

Returns:

eGwFileStatus processing status

Glasswall Document Processing

Modules

- **Document Processing Arguments**
- **Glasswall Document Processing, Memory to Memory Location**
- **Glasswall Document Processing, File to Memory Location**
- **Glasswall Document Processing, File to File**
- **Glasswall Document Processing, Supporting Functions**
- **Glasswall Document Processing Results**

Detailed Description

These functions support the processing of documents. The API provides a number of functions that enable document processing to be carried out it different modes with a variety of output formats.

The Glasswall library supports two processing modes.

Mode	Description
Analysis Audit	provides XML report on the structure and content of the specified document
Protect	produces a managed version of the specified document that is conformant with its specification and the content management policy

In addition to document processing an optional report is available. This report is a detailed log of the processing that has been carried out on the specified file.

The output from the processing can be directed to either a **memory location** or to **file locations**. The processing functions are documented in more detail against these two classifications.

Regardless of whether the 'file to memory location' or 'file to file' interface is used, the managed files produced by the Glasswall library will be of the same type as the input file.

The return status from each of the processing functions provides a simple assessment of document processing, **eGwFileStatus**. Additional information can be retrieved from the Glasswall library through the **processing results** functions.

Document Processing Arguments

This section describes the common arguments used in the document processing functions. The function argument naming is applied consistently across the API. An argument name prefixed with `wc` indicates that the function expects the argument to contain a wide character string.

Input Arguments

These arguments inform the process functions which file to process and the file type to expect.

inputFilePathName

The input file path for Glasswall processing. Glasswall ignores any file suffix, the file type must be specified in **wcType**

wcType

Each of the processing functions requires the expected file type to be specified through the wcType argument. The following table details the expected values for wcType.

The values for wcType are case-sensitive.

wcType	File type
pdf	PDF documents
doc	Binary MS Word files
xls	Binary MS Excel files
ppt	Binary MS PowerPoint files
docx	XML MS Word files
xlsx	XML MS Excel files
pptx	XML MS PowerPoint files
jpg	JPEG image files
gif	GIF image files
png	PNG image files
emf	EMF image files
wmf	WMF image files
bmp	BMP image files
tiff	TIFF image files
elf	ELF files
wmf	WMF files
mp3	MP3 files
mp4	MP4 files
mpg	MPEG files
wav	WAV files
o	MACHO files
pe	PE files

File to Memory Location Output Arguments

These arguments are used to return process information from the Glasswall library when using the File to Memory Locations process functions.

Client applications do not need to manage the release of memory allocated to these return buffers.

outputFileBuffer

The Glasswall processing output file buffer.

This buffer contains an image of the Glasswall processed output file which will be of the same data type as the input file and will have managed content removed and Glasswall protected according to the standard content specification allowed for the file data type.

The argument can be set to NULL when the outputFileBuffer is not required and only validation of a file is required. In this case then the ref gwOutputLength "outputLength" argument should also be set to NULL.

outputLength

Length of the the **outputFileBuffer**.

The argument can be set to NULL when the **outputFileBuffer** is not required and only validation of a file required. In this case then the **outputFileBuffer** should also be set to NULL.

analysisFileBuffer

Analysis (XML) file buffer from Glasswall processing. This buffer contains an image of the Glasswall ANALYSIS report of the specified file.

analysisFileBufferLength

Length of the the **analysisFileBuffer**.

reportFileBuffer

Detailed log file buffer from Glasswall processing. The argument can be set to NULL when the report is not required.

In this case then the **reportFileBufferLength** should also be set to NULL.

reportFileBufferLength

Length of the the **reportFileBuffer** that has been allocated. The argument can be set to NULL when the report is not required.

In this case then the **reportFileBuffer** should also be set to NULL.

File to File Output Arguments

These arguments are used to return process information from the Glasswall library when using the File to File process functions.

Destination folders for these files must exist and be writable, processing will be reported in error if these conditions are not met.

outputFilePathName

Destination for the managed file from Glasswall processing.

This file contains the Glasswall processed output file which will be of the same type as the input file, will have managed content removed and be Glasswall protected according to the standard content specification allowed for the file type.

analysisFilePathName

Destination for Analysis Audit report (XML) from Glasswall processing.

This file contains an image of the Glasswall ANALYSIS of the processed file.

reportFilePathName

Destination for detailed processing log file buffer from Glasswall processing.

exportFilePathName

Destination for the exported zip archive. This zip archive containing the Managed file, the Analysis report, and extracted elements such as images and text.

Determine File Type Output Enumeration

File Type Enumeration

The following table details the possible enumeration values that can be returned with a call to *GWDetermineFileTypeFromFile* and what they represent.

Value	Representation
0	Unknown or unrecognised file type
16	**pdf**
17	**doc**
18	**docx**
19	**ppt**
20	**pptx**
21	**xls**
22	**xlsx**
23	**png**
24	**jpg**
25	**gif**
26	**emf**
27	**wmf**
28	**rtf**
29	**bmp**
30	**tiff**
31	**pe**
32	**macho**
33	**elf**
34	**mp4**
35	**mp3**
36	**wav**
37	**mpg**
38	**coff**

The following table is an extension to the table above and indicates issues that may occur during interrogation.

Value	Representation
1	File issues preventing file type detection
2	Memory allocation issues preventing file type detection
3	Internal processing issues preventing file type detection

4	Licensing issues preventing file type detection
5	OPC password protected file

Process Return Status

All the processing functions return values as defined in **eGwFileStatus**.

Status Value	Effects
eGwFileStatus_Success	Managed documents are published to the specified locations

- For file to memory location, the size of the managed document is reported in the specified argument
- For file to file the specified file location is loaded with the processed document

Analysis reports are published to the specified locations, for memory locations the size of the analysis report is reported in the specified argument. Detailed reports are published to the specified locations, for memory locations the size of the detailed report is reported in the specified argument.	
eGwFileStatus_Error	Managed documents are not published. Memory location functions return a NULL pointer for **outputFileBuffer** and set the **outputFileBuffer** to zero.
other	The effects of the failure case is described in the notes for **eGwFileStatus**

Glasswall Document Processing, Memory to Memory Location

Functions

- int **GWMemoryToMemoryAnalysisAudit** (void *inputBuffer, size_t inputBufferLength, wchar_t *wcType, void **analysisFileBuffer, size_t *analysisFileBufferLength)
- int **GWMemoryToMemoryProtect** (void *inputBuffer, size_t inputBufferLength, const wchar_t *wcType, void **outputFileBuffer, size_t *outputLength)

Detailed Description

These functions process the document provided in a specified memory location and publish the resulting output to memory locations. Each of the function signatures requires the client application to provide a location to which the report or regenerated document can be written, along with a variable in which the size of the output is recorded. If the output file is not required, null pointers can be specified.

Parameters of type pointer to pointer are used by Glasswall to allocate memory for storing processed files and analysis reports. The client accesses the buffers through these parameters but is otherwise not responsible for managing the allocated memory except that **GWFileDone** must be called when processing is finished in order to release the allocated memory.

Function Documentation

int GWMemoryToMemoryAnalysisAudit (void * *inputBuffer*, size_t *inputBufferLength*, wchar_t * *wcType*, void ** *analysisFileBuffer*, size_t * *analysisFileBufferLength*)

This function protects the specified input file buffer, returning the analysis report of the processed file to the specified memory location.

Parameters:

in	*inputBuffer*	Input buffer containing the file to be processed.
in	*inputBufferLength*	Length of the input buffer containing the file to be processed.
in	*wcType*	**file type**
in,out	*analysisFileBuffer*	**output analysis report buffer from Glasswall processing.**
out	*analysisFileBufferLength*	**length of the outputFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Remarks:

Glasswall GWMemoryToMemoryAnalysisAudit processing example

```
   // input file buffer and length
   void *inputFileBuffer          = NULL;
   size_t inputFileLength         = 0;

 // output analysis report buffer and length
   void *analysisFileBuffer        = NULL;
   size t analysisFileBufferLength = 0;

 // load file from disk
 if (eGwFileStatus_Success == gwLoadFileFromDisk(L"C:\\data\\temp1.pdf",
&inputFileBuffer, &inputFileLength))
 {
    // Library configuration
    wchar_t* cmPolicy = getClientApplicationConfigurationXML();

    if (eGwFileStatus Success == GWFileConfigXML(cmPolicy))
    {
        if(GWMemoryToMemoryAnalysisAudit(inputFileBuffer, inputFileLength,
L"pdf", &analysisFileBuffer, &analysisFileBufferLength))
        {
            // process the analysis report
        }
    }
    else
    {
        // error handling
    }

    // free memory allocated by gwLoadFileFromDisk
    gwFree(inputFileBuffer);
 }
 else
```

```
{
    // error handling
}

   // release resources including the global buffer 'analysisFileBuffer'
   GWFileDone();
```

int GWMemoryToMemoryProtect (void * *inputBuffer*, size_t *inputBufferLength*, const wchar_t * *wcType*, void ** *outputFileBuffer*, size_t * *outputLength*)

This function protects the specified input file buffer, returning the managed version of the file to the specified memory location.

Parameters:

in	*inputBuffer*	Input buffer containing the file to be processed.
in	*inputBufferLength*	Length of the input buffer containing the file to be processed.
in	*wcType*	**file type**
in,out	*outputFileBuffer*	**output file buffer from Glasswall processing.**
out	*outputLength*	**length of the outputFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Remarks:

Glasswall GWMemoryToMemoryProtect processing example

```
 // input file buffer and length
 void *inputFileBuffer        = NULL;
 size t inputFileLength       = 0;

 // managed file output buffer and length
 void *outputFileImage        = NULL;
 size_t outputFileImageLength = 0;

 // load file from disk
 if (eGwFileStatus Success == gwLoadFileFromDisk(L"C:\\data\\temp1.pdf",
&inputFileBuffer, &inputFileLength))
 {
    // Library configuration
    wchar t* cmPolicy = getClientApplicationConfigurationXML();

    if (eGwFileStatus Success == GWFileConfigXML(cmPolicy))
    {
        if(GWMemoryToMemoryProtect(inputFileBuffer, inputFileLength, L"pdf",
&outputFileImage, &outputFileImageLength))
        {
            // process the managed file image
        }
    }
    else
    {
        // error handling
    }

    // free memory allocated by gwLoadFileFromDisk
    gwFree(inputFileBuffer);
 }
 else
{
    // error handling
}

 // release resources including the global buffer 'outputFileImage'
 GWFileDone();
```

Glasswall Document Processing, File to Memory Location

Functions

- int **GWFileProtect** (const wchar_t *inputFilePathName, const wchar_t *wcType, void **outputFileBuffer, size_t *outputLength)
- int **GWFileToMemoryAnalysisProtectAndExport** (const wchar_t *inputFilePathName, void **outputFileBuffer, size_t *outputLength)
- int **GWFileToMemoryProtectAndImport** (const wchar_t *inputFilePathName, void **outputFileBuffer, size_t *outputLength)
- int **GWFileProtectLiteAndReport** (wchar_t *inputFilePathName, wchar_t *wcType, void **outputFileBuffer, size_t *outputLength, void **reportFileBuffer, size_t *reportFileBufferLength)
- int **GWFileProtectAndReport** (wchar_t *inputFilePathName, wchar_t *wcType, void **outputFileBuffer, size_t *outputLength, void **reportFileBuffer, size_t *reportFileBufferLength)
- int **GWFileProtectLite** (wchar_t *inputFilePathName, wchar_t *wcType, void **outputFileBuffer, size_t *outputLength)
- int **GWFileAnalysisAudit** (wchar_t *inputFilePathName, wchar_t *wcType, void **analysisFileBuffer, size_t *analysisFileBufferLength)
- int **GWFileAnalysisAuditAndReport** (wchar_t *inputFilePathName, wchar_t *wcType, void **analysisFileBuffer, size_t *analysisFileBufferLength, void **reportFileBuffer, size_t *reportFileBufferLength)

Detailed Description

These functions process the specified document and publish the resulting output to memory locations. Each of the function signatures requires the client application to provide a location to which the report or managed document can be written, along with a variable in which the size of the output is recorded. If the output file is not required, null pointers can be specified.

Parameters of type pointer to pointer are used by Glasswall to allocate memory for storing processed files and analysis reports. The client accesses the buffers through these parameters but is otherwise not responsible for managing the allocated memory except that **GWFileDone** must be called when processing is finished in order to release the allocated memory.

Function Documentation

int GWFileAnalysisAudit (wchar_t * *inputFilePathName*, wchar_t * *wcType*, void ** *analysisFileBuffer*, size_t * *analysisFileBufferLength*)

This function carries out an Analysis Audit on the specified file returning the analysis report to the specified memory location.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
out	*analysisFileBuffer*	**analysis (XML) file buffer from Glasswall processing.**

out	*analysisFileBufferLength*	**length of the the analysisFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Remarks:

Glasswall GWFileAnalysisAudit processing example

```
// analysis file image buffer and length
void *analysisFileImage        = NULL;
size_t analysisFileImageLength = 0;

// Library configuration
wchar t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    if(eGwFileStatus Success == GWFileAnalysisAudit(L"C:\\data\\temp1.pdf",
L"pdf", &analysisFileImage, &analysisFileImageLength))
    {
        // process the analysis image
    }
}

// release resources including global buffer 'analysisFileImage'
GWFileDone();
```

int GWFileAnalysisAuditAndReport (wchar_t * *inputFilePathName*, wchar_t * *wcType*, void ** *analysisFileBuffer*, size_t * *analysisFileBufferLength*, void ** *reportFileBuffer*, size_t * *reportFileBufferLength*)

This function carries out an Analysis Audit on the specified file and produces a detailed report of the processing carried out, saving both outputs to the specified memory locations.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
out	*analysisFileBuffer*	**analysis (XML) file buffer from Glasswall processing.**
out	*analysisFileBufferLength*	**length of the the analysisFileBuffer that has been allocated.**
out	*reportFileBuffer*	**detailed report file buffer from Glasswall processing.**
out	*reportFileBufferLength*	**length of the the reportFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Remarks:

Glasswall GWFileAnalysisAuditAndReport processing example

```
// analysis file image buffer and length
void *analysisFileImage        = NULL;
size t analysisFileImageLength = 0;

void *fileReportImage          = NULL;
size_t fileReportImageLength   = 0;

// Library configuration
wchar t* cmPolicy = getClientApplicationConfigurationXML();
```

```
if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    if(eGwFileStatus Success ==
GWFileAnalysisAuditAndReport(L"C:\\data\\temp1.pdf", L"pdf",
&analysisFileImage, &analysisFileImageLength,

&fileReportImage, &fileReportImageLength))
    {
        // process the analysis image
        // process the report image
    }
}

// release resources including global buffers 'analysisFileImageLength' and
'fileReportImage'
GWFileDone();
```

int GWFileProtect (const wchar_t * *inputFilePathName*, const wchar_t * *wcType*, void ** *outputFileBuffer*, size_t * *outputLength*)

This function protects the specified input file, returning the managed version of the file to the specified memory location.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
in,out	*outputFileBuffer*	**output file buffer from Glasswall processing.**
out	*outputLength*	**length of the outputFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Remarks:

Glasswall GWFileProtect processing example

```
// managed file output buffer and length
void *fileImage        = NULL;
size_t fileImageLength = 0;

// Library configuration
wchar t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    if(GWFileProtect(L"C:\\data\\temp1.pdf", L"pdf", &fileImage,
&fileImageLength))
    {
        // process the managed file image
    }
}
else
{
    // error handling
}

// release resources including the global buffer 'fileImage'
GWFileDone();
```

int GWFileProtectAndReport (wchar_t * *inputFilePathName*, wchar_t * *wcType*, void ** *outputFileBuffer*, size_t * *outputLength*, void ** *reportFileBuffer*, size_t * *reportFileBufferLength*)

This function protects the specified input file, returning the managed version of the file and a detailed report of the processing done on the file to the specified memory locations.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
out	*outputFileBuffer*	**output file buffer from Glasswall processing.**
out	*outputLength*	**length of the outputFileBuffer that has been allocated.**
out	*reportFileBuffer*	**detailed report file buffer from Glasswall processing.**
out	*reportFileBufferLength*	**length of the the reportFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Remarks:

Glasswall GWFileProtectAndReport processing example This example sets the Glasswall configuration then manages the specified file using **GWFileProtectAndReport()**.

```
// managed file output buffer and length
void *fileImage          = NULL;
size_t fileImageLength   = 0;
void *reportImage        = NULL;
size t reportImageLength = 0;

// Library configuration
wchar_t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    if( eGwFileStatus Success == GWFileProtectAndReport(L"C:\\data\\temp1.pdf",
L"pdf", &fileImage, &fileImageLength,

&reportImage, &reportImageLength )
    {
        // process the managed file image
        // process the report image
    }
}
else
{
    // error handling
}

// release resources including global buffers 'fileImage' and 'fileReportImage'
GWFileDone();
```

int GWFileProtectLite (wchar_t * *inputFilePathName*, wchar_t * *wcType*, void ** *outputFileBuffer*, size_t * *outputLength*)

This function is a restricted version of **GWFileProtect**

This function protects the specified input file, returning the managed version of the file to the specified memory location.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**

in	*wcType*	**file data type**
in,out	*outputFileBuffer*	**output file buffer from Glasswall processing.**
out	*outputLength*	**length of the outputFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Remarks:

Glasswall GWFileProtectLite processing example

```
// managed file output buffer and length
void *fileImage        = NULL;
size_t fileImageLength = 0;

// Library configuration
wchar_t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    if(eGwFileStatus Success == GWFileProtectLite(L"C:\\data\\temp1.pdf",
L"pdf", &fileImage, &fileImageLength))
    {
        // process the managed file image
    }
}
else
{
    // error handling
}

// release resources including the global buffer 'fileImage'
GWFileDone();
```

int GWFileProtectLiteAndReport (wchar_t * *inputFilePathName*, wchar_t * *wcType*, void ** *outputFileBuffer*, size_t * *outputLength*, void ** *reportFileBuffer*, size_t * *reportFileBufferLength*)

This function is a restricted version of **GWFileProtectAndReport**

This function protects the specified input file, returning the managed version of the file and a detailed report of the processing done on the file to the specified memory locations.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
out	*outputFileBuffer*	**output file buffer from Glasswall processing.**
out	*outputLength*	**length of the outputFileBuffer that has been allocated.**
out	*reportFileBuffer*	**detailed report file buffer from Glasswall processing.**
out	*reportFileBufferLength*	**length of the the reportFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Remarks:

Glasswall GWFileProtectLiteAndReport processing example This example sets the Glasswall configuration then manages the specified file using **GWFileProtectLiteAndReport()**.

```
// managed file output buffer and length
void *fileImage          = NULL;
size_t fileImageLength   = 0;
```

```
void *reportImage        = NULL;
size t reportImageLength = 0;

// Library configuration
wchar_t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    if( eGwFileStatus Success == GWFileProtectLiteAndReport(L"C:\\data\\temp1.pdf",
L"pdf", &fileImage, &fileImageLength,

&reportImage, &reportImageLength))
    {
        // process the managed file image
        // process the report image
    }
}
else
{
    // error handling
}

// release resources including global buffers 'fileImage' and 'fileReportImage'
GWFileDone();
```

int GWFileToMemoryAnalysisProtectAndExport (const wchar_t * *inputFilePathName*, void ** *outputFileBuffer*, size_t * *outputLength*)

This function manages the specified file, carries out an Analysis Audit, extracts any text and image content which can be isolated, and returns the results in a block of memory owned and allocated by the Glasswall DLL.

The output file will be a zip archive containing the following:

- Glasswall XML analysis extended to include additional information and metadata which can be used to identify a particular item of image or text data within the export package and identify its point of origin within the original document for re-insertion after further processing if required. Note: Re-insertion is currently only supported for image files.
- The managed file.
- The extracted image and/or text data with each item taking the form of a separate file within the zip archive.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
out	*outputFileBuffer*	**output file buffer from Glasswall processing.**
out	*outputLength*	**length of the outputFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Attention:

The memory buffer holding the output file image is allocated and owned by the Glasswall library. Input file type will be automagically detected.

Remarks:

Glasswall GWFileToMemoryAnalysisProtectAndExport processing example

```
// managed file output buffer and length
```

```
void *fileImage        = NULL;
size t fileImageLength = 0;

// Library configuration
wchar_t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    if (eGwFileStatus Success ==
GWFileToMemoryAnalysisProtectAndExport(L"C:\\data\\temp1.pdf", &fileImage,
&fileImageLength))
    {
        // extract content of output package
        // process managed file if it was managed successfully
        // process analysis file
        // Process exported content files
    }
}
else
{
    error handling
}
```

int GWFileToMemoryProtectAndImport (const wchar_t * *inputFilePathName*, void ** *outputFileBuffer*, size_t * *outputLength*)

This function takes a file in the form of a zip archive originally generated by GWFileToFileAnalysisProtectAndExport, extracts the managed file, the XML analysis file, and externally validated (and if neccessary and/or appropriate reconstructed) image data and uses the metadata encoded into the analysis file to regenerate the managed file replacing the original image content replaced with the externally processed versions. The regenerated file is returned in the form of a block of memory allocated and owned by the Glasswall DLL.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
out	*outputFileBuffer*	**output file buffer from Glasswall processing.**
out	*outputLength*	**length of the outputFileBuffer that has been allocated.**

Returns:

eGwFileStatus processing status

Attention:

The memory buffer holding the output file image is allocated and owned by the Glasswall library.

Remarks:

Glasswall GWFileToMemoryProtectAndImport processing example :-

```
// managed file output buffer and length
void *fileImage        = NULL;
size t fileImageLength = 0;

// Library configuration
wchar_t* cmPolicy      = getClientApplicationConfigurationXML();

if (eGwFileStatus Success == GWFileConfigXML(cmPolicy))
{
    if(eGwFileStatus_Success ==
GWFileToMemoryProtectAndImport(L"C:\\data\\output\\import1.zip", &fileImage,
&fileImageLength))
    {
        // process output file
    }
    else
    {
```

```
        // error handling
    }
}
else
{
    // error handling
}
```

Glasswall Document Processing, File to File

Functions

- int **GWFileToFileAnalysisProtectAndExport** (const wchar_t *inputFilePathName, const wchar_t *exportFilePathName)
- int **GWFileToFileProtectAndImport** (const wchar_t *inputFilePathName, const wchar_t *outputFilePathName)
- int **GWFileToFileProtectLite** (wchar_t *inputFilePathName, wchar_t *wcType, wchar_t *outputFilePathName)
- int **GWFileToFileProtect** (wchar_t *inputFilePathName, wchar_t *wcType, wchar_t *outputFilePathName)
- int **GWFileToFileAnalysisAudit** (wchar_t *inputFilePathName, wchar_t *wcType, wchar_t *analysisFilePathName)
- int **GWFileToFileProtectLiteAndReport** (wchar_t *inputFilePathName, wchar_t *wcType, wchar_t *outputFilePathName, wchar_t *reportFilePathName)
- int **GWFileToFileProtectAndReport** (wchar_t *inputFilePathName, wchar_t *wcType, wchar_t *outputFilePathName, wchar_t *reportFilePathName)
- int **GWFileToFileAnalysisAuditAndReport** (wchar_t *inputFilePathName, wchar_t *wcType, wchar_t *analysisFilePathName, wchar_t *reportFilePathName)

Detailed Description

These functions process the specified document and publish the resulting output to the specified file locations. Each of the function signatures requires the client application to specify file locations to which the reports or managed documents can be written.

It is the responsibility of the client application to ensure that the folder containing the specified file location exists. If the containing folder does not exist, or 'write privileges' are not enabled, the processing function will return a negative error status, **eGwFileStatus**.

Function Documentation

int GWFileToFileAnalysisAudit (wchar_t * *inputFilePathName*, wchar_t * *wcType*, wchar_t * *analysisFilePathName*)

This function carries out an Analysis Audit on the specfied file, saving the resultant report to the specified file location.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
in,out	*analysisFilePathNa*	**destination for analysis (XML) file from Glasswall processing.**

	me	

Returns:

eGwFileStatus processing status

Attention:

The folders specified in the paths of the output files must already exist and have appropriate write permissions

Remarks:

Glasswall GWFileToFileAnalysisAudit processing example with analysis :-

```
// Library configuration
wchar_t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    createFolder("C:\\data\\audit.issues");

    if(eGwFileStatus_Success ==
GWFileToFileAnalysisAudit(L"C:\\data\\temp1.pdf", L"pdf",
L"C:\\data\\audit.issues\\temp1_.xml"))
    {
        // process analysis report
    }
}
```

int GWFileToFileAnalysisAuditAndReport (wchar_t * *inputFilePathName*, wchar_t * *wcType*, wchar_t * *analysisFilePathName*, wchar_t * *reportFilePathName*)

This function carries out an analysis audit on the specified file saving the results and a detailed report to the specified files.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
out	*analysisFilePathName*	**destination for analysis (XML) file from Glasswall processing.**
out	*reportFilePathName*	**detailed report file buffer from Glasswall processing.**

Returns:

eGwFileStatus processing status

Attention:

The folders specified in the paths of the output files must already exist and have appropriate write permissions

Remarks:

Glasswall GWFileToFileAnalysisAuditAndReport processing example

```
// Library configuration
wchar_t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus Success == GWFileConfigXML(cmPolicy))
{
    if(eGwFileStatus Success ==
GWFileToFileAnalysisAuditAndReport(L"C:\\data\\temp1.pdf", L"pdf",
L"C:\\data\\output\\temp1_audit.xml", L"C:\\data\\output\\temp1_audit.log"))
    {
```

```
        // process analysis file
        // process report file
    }
}
```

int GWFileToFileAnalysisProtectAndExport (const wchar_t * *inputFilePathName*, const wchar_t * *exportFilePathName*)

This function manages the specified file, carries out an Analysis Audit, extracts any text and image content which can be isolated, and saves the results to the specified file location.

The output will be a zip archive file with the following:

- Glasswall XML analysis report extended to include additional information and metadata which can be used to identify a particular item of image or text data within the export package and identify its point of origin within the original document for re-insertion after further processing if required. Note: Re-insertion is currently only supported for image files.
- The managed file.
- The extracted image and/or text data with each item taking the form of a separate file within the zip archive.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
out	*exportFilePathNam e*	**destination for content export (zip) file from Glasswall processing**

Returns:

eGwFileStatus processing status

Attention:

The folders specified in the paths of the output files must already exist and have appropriate write permissions. Input file type will be automagically detected.

Remarks:

Glasswall GWFileToFileAnalysisProtectAndExport processing example

```
// Library configuration
eGwFileStatus status = eGwFileStatus_Error;
wchar_t* cmPolicy    = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    status = GWFileToFileAnalysisProtectAndExport(L"C:\\data\\temp1.pdf",
L"C:\\data\\output\\temp1.zip"));

    if( status != eGwFileStatus_SuccessExportWriteFailure
          && status != eGwFileStatus_ErrorExportWriteFailure )
    {
        // extract content of output package.
        // process managed file if it was managed successfully
        // process analysis file
        // process exported content files
    }
    else
    {
        // error handling
    }
}
else
{
```

```
    // error handling
}
```

int GWFileToFileProtect (wchar_t * *inputFilePathName*, wchar_t * *wcType*, wchar_t * *outputFilePathName*)

This function Manages the specified file saving the managed output to the specified file.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
in	*outputFilePathNam e*	**destination for output file from Glasswall processing.**

Returns:

eGwFileStatus processing status

Attention:

The folders specified in the paths of the output files must already exist and have appropriate write permissions

Remarks:

Glasswall GWFileToFileProtect processing example: -

```
// Library configuration
wchar t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    createFolder("C:\\output");

    if(eGwFileStatus Success == GWFileToFileProtect(L"C:\\data\\temp1.pdf",
L"pdf", L"C:\\output\\temp1.pdf"))
    {
        // process output file
    }
}
else
{
    // error handling
}
```

int GWFileToFileProtectAndImport (const wchar_t * *inputFilePathName*, const wchar_t * *outputFilePathName*)

This function takes a file in the form of a zip archive originally generated by GWFileToFileAnalysisProtectAndExport, extracts the managed file, the XML analysis file, and externally validated (and if neccessary and/or appropriate reconstructed) image data and uses the metadata encoded in the analysis file to regenerate the managed file replacing the original image content replaced with the externally processed versions.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*outputFilePathNam e*	**destination for output file from Glasswall processing.**

Returns:

eGwFileStatus processing status

Attention:

The folders specified in the paths of the output files must already exist and have appropriate write permissions. Input file type will be automagically detected.

Remarks:

Glasswall GWFileToFileProtectAndImport processing example: -

```
// Library configuration
wchar t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    if(eGwFileStatus_Success ==
GWFileToFileProtectAndImport(L"C:\\data\\output\\temp1.zip",
L"C:\\output\\import1.pdf" ))
    {
        // process output file
    }
    else
    {
        // error handling
    }
}
else
{
    // error handling
}
```

int GWFileToFileProtectAndReport (wchar_t * *inputFilePathName*, wchar_t * *wcType*, wchar_t * *outputFilePathName*, wchar_t * *reportFilePathName*)

This function Manages the specified file, saving the managed output and a detailed process log to the specified file locations.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
out	*outputFilePathNam e*	**destination for output file from Glasswall processing.**
out	*reportFilePathNam e*	**detailed report file buffer from Glasswall processing.**

Returns:

eGwFileStatus processing status

Attention:

The folders specified in the paths of the output files must already exist and have appropriate write permissions

Remarks:

Glasswall GWFileToFileProtectAndReport processing example :-

```
// Library configuration
wchar t* cmPolicy = getClientApplicationConfigurationXML();
```

```
if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    createFolder("C:\\data\\output");

    if(eGwFileStatus_Success ==
GWFileToFileProtectAndReport(L"C:\\data\\temp1.pdf", L"pdf",
L"C:\\data\\output\\temp1.pdf", L"C:\\data\\output\\temp1.log"))
    {
        // process managed file
        // process report file
    }
}
else
{
    // error handling
}
```

int GWFileToFileProtectLite (wchar_t * *inputFilePathName*, wchar_t * *wcType*, wchar_t * *outputFilePathName*)

This function is a restricted version of **GWFileToFileProtect**

This function Manages the specified file saving the managed output to the specified file.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
in	*outputFilePathNam e*	**destination for output file from Glasswall processing.**

Returns:

eGwFileStatus processing status

Attention:

The folders specified in the paths of the output files must already exist and have appropriate write permissions.

Remarks:

Glasswall GWFileToFileProtectLite processing example :-

```
 // managed file output buffer and length
 void *fileImage = NULL;
 fileImageLength = 0;

 // Library configuration
wchar t* cmPolicy = getClientApplicationConfigurationXML();

 if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
 {
     createFolder("C:\\output\\");

     if(eGwFileStatus Success == GWFileToFileProtectLite( "C:\\data\\temp1.pdf",
"pdf", "C:\\output\\temp1.pdf" )
     {
         // process output file
     }
 }
 else
 {
     // error handling
 }
```

int GWFileToFileProtectLiteAndReport (wchar_t * *inputFilePathName*, wchar_t * *wcType*, wchar_t * *outputFilePathName*, wchar_t * *reportFilePathName*)

This function is a restricted version of **GWFileToFileProtectAndReport**

This function Manages the specified file, saving the managed output and a detailed process log to the specified file locations.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
in	*wcType*	**file data type**
out	*outputFilePathName*	**destination for output file from Glasswall processing.**
out	*reportFilePathName*	**detailed report file buffer from Glasswall processing.**

Returns:

eGwFileStatus processing status

Attention:

The folders specified in the paths of the output files must already exist and have appropriate write permissions

Remarks:

Glasswall GWFileToFileProtectLiteAndReport processing example :-

```
// Library configuration
wchar_t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus Success == GWFileConfigXML(cmPolicy))
{
    createFolder("C:\\data\\output");

    if(eGwFileStatus_Success ==
GWFileToFileProtectLiteAndReport(L"C:\\data\\temp1.pdf", L"pdf",
L"C:\\data\\output\\temp1.pdf", L"C:\\data\\output\\temp1.log"))
    {
        // process managed file
        // process report file
    }
}
else
{
    // error handling
}
```

Glasswall Document Processing, Supporting Functions

Functions

- int **GWDetermineFileTypeFromFileInMem** (void *inputBuffer, size_t inputBufferSize)
- int **GWDetermineFileTypeFromFileInMemAndReport** (void *inputBuffer, size_t inputBufferSize, void **reportFileBuffer, size_t *reportFileBufferLength)
- int **GWDetermineFileTypeFromFile** (const wchar_t *inputFilePathName)
- int **GWDetermineFileTypeFromFileAndReport** (const wchar_t *inputFilePathName, void **reportFileBuffer, size_t *reportFileBufferLength)

Detailed Description

These functions may be used alongside the document processing functions for additional support on certain aspects or attributes such as determining the file extension for a given file.

For every function that relies on an input file, it is the responsibility of the client application to ensure that the folder containing the specified file location exists. If the containing folder does not exist, or 'read privileges' are not enabled, the processing function will return an error status.

Function Documentation

int GWDetermineFileTypeFromFile (const wchar_t * *inputFilePathName*)

This function determines the file type for a given file provided it is supported by Glasswall.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**

Returns:

File Type Enumeration Value

Remarks:

Glasswall GWDetermineFileTypeFromFile example

```
// Input file path, managed file output buffer and length
const wchar t* inputFilePath = L"C:\\data\\temp1.pdf";
void *fileImage              = NULL;
size_t fileImageLength       = 0;

// Library configuration
wchar t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus_Success == GWFileConfigXML(cmPolicy))
{
    wchar_t* fileType = NULL;
    int eFileType     = 0;

    // Determine the file type
    eFileType = GWDetermineFileTypeFromFile(inputFilePath);

    switch (eFileType)
    {
        case 16:
            fileType = L"pdf";
            break;

        case 17:
            fileType = L"doc";
            break;

        ...

        case 27:
            fileType = L"wmf";
            break;
    }

    if (fileType)
    {
        // Call Glasswall processiong function
        if(GWFileProtect(inputFilePath, fileType, &fileImage, &fileImageLength))
```

```
        {
            // Process the managed file image
        }
    }
    else
    {
        // Error handling
    }
}
else
{
    // Error handling
}

// Release resources including the global buffers
GWFileDone();
```

int GWDetermineFileTypeFromFileAndReport (const wchar_t * *inputFilePathName*, void ** *reportFileBuffer*, size_t * *reportFileBufferLength*)

This function determines the file type for a given file provided it is supported by Glasswall, it also populates an XML report with the approach used to determine the file type.

Parameters:

in	*inputFilePathName*	**input file path for Glasswall processing**
out	*analysisFileBuffer*	**analysis (XML) file buffer from Glasswall processing.**
out	*analysisFileBufferLength*	**length of the the analysisFileBuffer that has been allocated.**

Returns:

File Type Enumeration Value

Remarks:

Glasswall GWDetermineFileTypeFromFile example

```
// Input file path, managed file output buffer and length
const wchar t* inputFilePath            = L"C:\\data\\temp1.pdf";

void *analysisFileBuffer                = NULL;
size_t analysisFileBufferLength     = 0;

void *fileImage                     = NULL;
size t fileImageLength                  = 0;

// Library configuration
wchar t* cmPolicy = getClientApplicationConfigurationXML();

if (eGwFileStatus Success == GWFileConfigXML(cmPolicy))
{
    wchar_t* fileType = NULL;
    int eFileType     = 0;

    // Determine the file type
    eFileType =  GWDetermineFileTypeFromFileAndReport(inputFilePath,
&analysisFileBuffer, &analysisFileBufferLength);

    switch (eFileType)
    {
        case 16:
            fileType = L"pdf";
            break;

        case 17:
            fileType = L"doc";
            break;

        ...
```

```
        case 27:
            fileType = L"wmf";
            break;
    }

    if (fileType)
    {
        // Call Glasswall processiong function
        if(GWFileProtect(inputFilePath, fileType, &fileImage, &fileImageLength))
        {
            // Process the managed file image
         // Report file type detection approach using content in analysisFileBuffer
        }
    }
    else
    {
        // Error handling
    }
}
else
{
    // Error handling
}

// Release resources including the global buffers
GWFileDone();
```

int GWDetermineFileTypeFromFileInMem (void * *inputBuffer*, size_t *inputBufferSize*)

This function determines the file type for a file stored in memory provided that the file type is supported by Glasswall.

Parameters:

in	*inputBuffer*	input file buffer containing the files to be processed
in	*inputBufferLength*	input file buffer length

Returns:

File Type Enumeration Value

Remarks:

Glasswall GWDetermineFileTypeFromFileInMem example

```
 // input file buffer and length
void *inputFileBuffer               = NULL;
size t inputFileLength              = 0;

 // output buffer and length
void *outputFileBuffer              = NULL;
size_t outputFileLength             = 0;

 // load file from a memory location
 if (eGwFileStatus Success == LoadFileFromMemory(&inputFileBuffer,
&inputFileLength))
 {
    // Library configuration
    wchar t* cmPolicy = getClientApplicationConfigurationXML();

    if (eGwFileStatus Success == GWFileConfigXML(cmPolicy))
    {
        wchar_t* fileType = NULL;
        int eFileType     = 0;

        // Determine the file type
        eFileType = GWDetermineFileTypeFromFileInMem(inputFileBuffer,
inputFileLength);

        switch (eFileType)
        {
            case 16:
```

```
                fileType = L"pdf";
                break;

            case 17:
                fileType = L"doc";
                break;
            ...

            case 27:
                fileType = L"wmf";
                break;
        }

        if (fileType)
        {
            // Call Glasswall processiong function
            if(GWMemoryToMemoryProtect(inputFileBuffer, inputFileLength,
fileType, &fileImage, &fileImageLength))
            {
                // Process the managed file image
            }
        }
        else
        {
            // Error handling
        }
    }
    else
    {
        // Error handling
    }

    // free memory allocated by LoadFileFromMemory
    gwFree(inputFileBuffer);
}
else
{
       // Error handling
}

// Release resources including the global buffers
GWFileDone();
```

int GWDetermineFileTypeFromFileInMemAndReport (void * *inputBuffer*, size_t *inputBufferSize*, void ** *reportFileBuffer*, size_t * *reportFileBufferLength*)

This function determines the file type for a given file provided it is supported by Glasswall, it also populates an XML report with the approach used to determine the file type.

Parameters:

in	*inputBuffer*	input file buffer containing the files to be processed
in	*inputBufferLength*	input file buffer length
out	*analysisFileBuffer*	**analysis (XML) file buffer from Glasswalls file type detection mechanism.**
out	*analysisFileBufferLength*	**length of the the analysisFileBuffer that has been allocated.**

Returns:

File Type Enumeration Value

Remarks:

Glasswall GWDetermineFileTypeFromFileInMemAndReport example

```
// input file buffer and length
void *inputFileBuffer               = NULL;
size_t inputFileLength              = 0;
```

```
// determine file type output buffer and length
void *fileTypeReportBuffer          = NULL;
size t fileTypeReportBufferLength   = 0;

// output file buffer and length
void *outputFileBuffer              = NULL;
size_t outputFileLength             = 0;

// load file from a memory location
if (eGwFileStatus Success == LoadFileFromMemory(&inputFileBuffer,
&inputFileLength))
{
    // Library configuration
    wchar t* cmPolicy = getClientApplicationConfigurationXML();

    if (eGwFileStatus Success == GWFileConfigXML(cmPolicy))
    {
        wchar_t* fileType = NULL;
        int eFileType     = 0;

        // Determine the file type
        eFileType = GWDetermineFileTypeFromFileInMemAndReport(inputFileBuffer,
inputFileLength, &fileTypeReportBuffer, &fileTypeReportBufferLength);

        switch (eFileType)
        {
            case 16:
                fileType = L"pdf";
                break;

            case 17:
                fileType = L"doc";
                break;
            ...

            case 27:
                fileType = L"wmf";
                break;
        }

        if (fileType)
        {
            // Call Glasswall processiong function
            if(GWMemoryToMemoryProtect(inputFileBuffer, inputFileLength,
fileType, &fileImage, &fileImageLength))
            {
                // Process the managed file image
            }
        }
        else
        {
            // Error handling
        }

        // Process xml report from GWDetermineFileTypeFromFileInMemAndReport
stored in fileTypeReportBuffer
    }
    else
    {
        // Error handling
    }

    // free memory allocated by LoadFileFromMemory
    gwFree(inputFileBuffer);
}
else
{
       // Error handling
}

// Release resources including the global buffers
GWFileDone();
```

Glasswall Document Processing Results

Modules

- **Glasswall Return Status Definitions**
- **Process Status Bitmask Definitions**

Functions

- wchar_t * **GWFileProcessMsg** (void)
- int **GWFileProcessStatus** (unsigned int *glasswallProcessStatus)
- wchar_t * **GWFileErrorMsg** (void)

Detailed Description

Each function returns status information on the last document processed by the library. No historical data is persisted.

These functions may be used once a document has been processed to elicit additional information on what processing was carried out on the document and, in the event of a document being identified as not conforming with its specification, the reason for non-conformance.

The **eGwFileStatus** enumeration is returned from all document processing functions to give an indication of the processing results. From this enumeration it can be established whether the document is 'managed' or not. A managed document may still have had changes made to it, the functions **GWFileProcessMsg** or **GWFileProcessStatus** can be used to identify the types of changes, if any, that have have been carried out on the document.

In the event of a document processing function returning a status value indicating that the document was found to be non-conforming, the function **GWFileErrorMsg** can be used to return an explanatory line of text.

Function Documentation

wchar_t* GWFileErrorMsg (void)

This function retrieves the Glasswall Process error message. If there are multiple reasons for document processing to return an error code only one shall be reported from this function. If more details on the process failures are required, the document can analysed to produce a detailed report on the contents of the document. All issues will be logged in the report.

Note that this string will be set to L"" if the Glasswall process has succeeded.

Returns:

a pointer to a printable Null-terminated string.

wchar_t* GWFileProcessMsg (void)

The file process message is a text representation of the status value detailed in **GWFileProcessStatus()**.

Returns:

a pointer to a printable Null-terminated string.

int GWFileProcessStatus (unsigned int * *glasswallProcessStatus*)

This function retrieves the Glasswall Process Status. This status gives a high level indication of the processing that was carried out on the last document processed by the library.

The status value can be retrieved as a text string using the associated **GWFileProcessMsg()** function

Parameters:

out	*glasswallProcessStatus*	Glasswall process status is an unsigned integer. The possible values and meanings are detailed in **Process Status Bitmask Definitions** NB: Parameter 'glasswallProcessStatus' may be passed as NULL if only Glasswall process status availability checking is required.)

Returns:

1(Success) 0(Error)

Glasswall Resource Management

Functions

- int **GWFileDone** (void)

Detailed Description

These utilities may be used by the client to release allocated resource once Glasswall processing has been completed.

Function Documentation

int GWFileDone (void)

Called by the client to release any remaining dynamic resource when Glasswall processing and output files creation has been completed for a process file.

This is equivalent to releasing all dynamic resources in preparation for unloading the dynamic link library.

Note that this includes the release of any global buffers which have been allocated from the heap to provide a single level of "client data separation".

Returns:

eGwFileStatus

Glasswall Return Status Definitions

Enumerations

- enum **eGwFileStatus** {
  eGwFileStatus_Success = 1,
  eGwFileStatus_Error = 0,
  eGwFileStatus_SuccessDocumentWriteFailure = -1,
  eGwFileStatus_SuccessAnalysisWriteFailure = -2,
  eGwFileStatus_ErrorAnalysisWriteFailure = -3,
  eGwFileStatus_SuccessReportWriteFailure = -4,
  eGwFileStatus_SuccessDocumentReportWriteFailure = -5,
  eGwFileStatus_ErrorReportWriteFailure = -6,
  eGwFileStatus_SuccessAnalysisReportWriteFailure = -7,
  eGwFileStatus_ErrorAnalysisReportWriteFailure = -8,
  eGwFileStatus_InternalError = -9
  }

Detailed Description

The document processing functions use the enumerations defined on these pages to describe the processing results.

For the **Glasswall Document Processing, File to Memory Location** functions only **eGwFileStatus_Success**, **eGwFileStatus_Error** and **eGwFileStatus_InternalError** are used. For the **Glasswall Document Processing, File to File** functions the negative values are utilised to report issues with file output (possibly in addition to file processing issues)

Attention:

Some of the **Glasswall Document Processing, File to File** functions' negative return values indicate documents have been successfully managed. It may be additional outputs that have caused the error condition to be reported. Care should be taken to make the managed document available if possible.

Enumeration Type Documentation

enum eGwFileStatus

Enumerations report the following conditions

Enumerator:

eGwFileStatus_Success	This value indicates the operation completed successfully. Any required Analysis or Protection was carried out and completed. Any required output files were written.
eGwFileStatus_Error	This value indicates that the document was non-conformant in some way, but any requested output files were written.
eGwFileStatus_Suc	This value indicates that the document was managed successfully, but

cessDocumentWriteFailure	a failure occured when writing the managed version of the document to file.
eGwFileStatus_SuccessAnalysisWriteFailure	This value indicates that the document was analysed successfully, but a failure occured when writing the analysis of the document to file.
eGwFileStatus_ErrorAnalysisWriteFailure	This value indicates that the document was non-conformant in some way, and a failure occured when writing the analysis of the document to file.
eGwFileStatus_SuccessReportWriteFailure	This value indicates that the document was processed successfully, but that a failure occured when writing the processing report to file.
eGwFileStatus_SuccessDocumentReportWriteFailure	This value indicates that the document was managed successfully, but a failure occured when writing both the managed version of the document and the processing report to file.
eGwFileStatus_ErrorReportWriteFailure	This value indicates that the document was non-conformant in some way, and that a failure occured when writing the processing report to file.
eGwFileStatus_SuccessAnalysisReportWriteFailure	This value indicates that the document was analysed successfully, but a failure occured when writing both the analysis of the document and the processing report to file.
eGwFileStatus_ErrorAnalysisReportWriteFailure	This value indicates that the document was non-conformant in some way, but a failure occured when writing both the analysis of the document and the processing report to file.
eGwFileStatus_InternalError	This value indicates an uncategorised error

Process Status Bitmask Definitions

Macros

- #define **GLASSWALL_PROCESS_STATUS_CLEAN** ((unsigned int) 0x0000)
- #define **GLASSWALL_PROCESS_STATUS_NONCONFORMING** ((unsigned int) 0x0080)
- #define **GLASSWALL_PROCESS_STATUS_REMEDIED** ((unsigned int) 0x0040)
- #define **GLASSWALL_PROCESS_STATUS_SANITISED** ((unsigned int) 0x0020)
- #define **GLASSWALL_PROCESS_STATUS_NOPROCESSFILE** ((unsigned int) 0xffff)

Detailed Description

Macro Definition Documentation

#define GLASSWALL_PROCESS_STATUS_CLEAN ((unsigned int) 0x0000)

Original document conforms to the specification, no changes have been made

#define GLASSWALL_PROCESS_STATUS_NONCONFORMING ((unsigned int) 0x0080)

Document either doesn't conform to its specification or does not comply with content management policy currently in force

#define GLASSWALL_PROCESS_STATUS_NOPROCESSFILE ((unsigned int) 0xffff)

No Glasswall Process file

#define GLASSWALL_PROCESS_STATUS_REMEDIED ((unsigned int) 0x0040)

Document has been updated in order to make it compliant with the specification

#define GLASSWALL_PROCESS_STATUS_SANITISED ((unsigned int) 0x0020)

Document has been updated in order to make it compliant with the content management policy

Disclaimer

Index