Reference: 2007/04/16; v 2007.1

ORACLE

Oracle Outside In Clean Content SDK

Quick Start Guide

This document provides an overview of the Outside In Clean Content Software Developer's Kit (SDK). It includes general information, system requirements, download details, and steps to integration.

Contents

Product Overview

Oracle Outside In Clean Content SDK provides all the components, documentation, samples and other resources required by third party developers to integrate Oracle's document analysis, scrubbing, extraction and export technology into their own applications.

The technology provides the ability to identify (analyze) and/or remove (sc rub) various features or parts (targets) of Microsoft Word, Excel and PowerPoint documents (versions 1997—2003). Most targets relate to well known security risks in these file formats although some, like identifying a document as being encrypted, are more general.

The target elements are as follows:

- Audio and Video Paths
- Author History
- Comments
- Content Properties
- Custom Properties
- Database Queries
- Embedded Objects
- Fast Save Data
- Hidden Slides
- Hidden Text
- Linked Objects
- Macros and Code
- Office GUID Property
- Outlook Properties
- Presentation Notes
- Printer Information
- Routing Slip
- Scenario Comments
- Sensitive Hyperlinks
- Sensitive INCLUDE Fields
- Statistic Properties
- Summary Properties
- Template Name
- Tracked Changes
- User Names
- Versions
- Weak Protections

It is delivered as a Software Development Kit (SDK) with Java and C/C++ Application Programming Interfaces (APIs). It is available for Linux, Solaris and Windows operating systems.

Target Audience

The product is for Software Developers who wish to integrate this functionality into their applications.

Use Cases

Applications that manage, store and present content including:

- Email
- Content & Document Management
- Groupware & Collaboration
- Archive & Storage
- Enterprise Portals
- Records Management
- Contract Management

- Content Publishing
- Policy Management
- Regulatory/Compliance (e.g., Sarbanes Oxley)

Security-related content analysis and inspection applications including:

- Email & Messaging Security
- Content Filtering & Inspection
- Policy Management
- Compliance
- Encryption

System Requirements

Pentium 166Mhz with 128MB of memory and 100MB free disk space.

Download Details

AVAILABLE FILES

Each of the following downloads include all the files needed to evaluate/implement the technology for that operating system.

CleanContentSDK_2007_1_linux.tar.gz (29.99 MB)

CleanContentSDK_2007_1_solaris.tar.gz (25.92 MB)

CleanContentSDK_2007_1_windows.ZIP (26.10 MB)

DIRECTORY STRUCTURE

The SDK's directory structure provides easy access to all the components, samples, documentation and other files needed to integrate the Clean Content SDK into your application.

ROOT

Root directory of the SDK

CleanContentSDKDemo.exe
CleanContentSDKDemo.sh

The Clean Content SDK demo application. Depending on the operating system one of these two files will be present. This application is designed to demonstrate the full potential of the Clean Content API and allow developers to explore the analysis and scrubbing behavior and options of this SDK in a full featured GUI environment.

app

Directory containing components and documentation for the CleanContentSDKDemo application

c

Directory containing libraries, include files, samples and other files required to use the Clean Content C/C++ API.

include
Directory containing include files required to use the C/C++ API

lib
Directory containing native code libraries needed to use the C/C++ API

windows
Directory containing DLLs and LIBs needed to use the C/C++ API on Microsoft Windows. This directory will include one or more sub-directories that correspond to the processor architecture for which Clean Content is available. Currently only Intel x86 is supported so there will only be a **x86** sub-directory.

linux
Directory containing library archives needed to use the C/C++ API on Linux. This directory will include one or more sub-directories that correspond to the processor architecture for which Secure is available. Currently only Intel x86 is supported so there will only be a **x86** sub-directory.

sample
Directory containing C/C++ API sample applications.

csample
Directory containing a cross-platform, pure C, sample application. Including Visual Studio project files for Windows and GCC-based makefile for Linux.

cppsample
Directory containing a cross-platform, C++, sample application. Including Visual Studio project files for Windows and GCC-based makefile for Linux.

dumptext
Directory containing a cross-platform, C++ sample application that shows how to retrieve the text out of a document using an element handler. Including Visual Studio project files for Windows and GCC-based makefile for Linux.

docs
Directory containing documentation for the SDK

cdoc
Directory containing C/C++ API documentation

javadoc
Directory containing Java API documentation

java
Directory containing components, samples and other files required to use the Clean Content Java API

lib
Directory containing CleanContent.jar that should be shipped with your application. See installation instructions.

sample
Directory containing Java API sample applications. Sample directories include batch files and shell scripts to build and run each application.

AnalyzeDirectorySample

Directory containing a command line sample application that analyzes all the documents in a given directory.

jre

Directory containing the Java Runtime Environment needed to run the CleanContentSDKDemo application and the Java sample applications. Stellent chooses to ship this JRE along with its SDK instead of requiring developers to "install Java" before using these applications.

samplefiles

Directory containing a series of "fake" documents created to give the developer some files to test the CleanContentSDKDemo application and other sample applications against. These files collectively contain all the scrub and analyze targets.

exception

Directory containing a series of Microsoft Word documents built specifically to trigger Secure to generate certain exceptions, including null pointer exception and out of memory exception. The document names indicate the Java exception they generate. These documents were developed to help OEMs build QA processes that include exception testing.

Understanding the Product

ARCHITECTURE

The core of the SDK is a Java component that performs the actual analysis and scrubbing. The component can currently be accessed in one of two ways. If your application is written in Java or has direct access to Java classes (a web site using Java Server Pages for example) the component can be used directly through its Java API. If your application is running on Windows or Linux and is using C or C++ the component can be accessed through its C/C++ API.

INTEGRATION STEP BY STEP

What follows is a walk through of the steps a developer must take in order to successfully integrate this technology into their application.

1. Explore Clean Content's functionality using the CleanContentSDKDemo application

One of the best ways to "get your head around" this technology is to play with the sample applications, especially the very functional CleanContentSDKDemo (`CleanContentSDKDemo.exe` or `CleanContentSDKDemo.sh`) application at the root of the SDK's directory structure. This application exposes the entire breadth of the API in an easy to use GUI application.

2. Choose an interface

For most developers this decision will be fairly obvious. If your application is Java-based or you use some other technology to directly access Java then choose the Java interface. If your application is written in C or C++ and runs on Windows or Linux then choose the C/C++ interface.

See the Java API or C/C++ API documentation for details on usage and installation. Please carefully read the installation section in your chosen API for details on exactly what parts of this SDK need to be shipped with your product.

3. Review sample applications for that interface

Running and inspecting the code of the sample applications for the interface you have selected is a great way to get a feel for the API. The sample applications are written to provide simple examples of API usage and should not be taken as the basis for a production application. Important issues like maximizing performance and robust error checking are not fully addressed by the sample application code.

4. Choose a method of providing the source document

Starting with the 2005.2 version of this API, developers have several ways to provide the source document.

Normal file

If the file to be processed is on a local or remote storage then a path name (or in Java's case a File object) is the easiest way to provide a source document to the API. See the `SecureRequest.setOption(FileOption option, java.io.File value)` method in Java, the `BFSetFileOption` function in C or the `BFSecureRequest::SetOption(FileOptions option, std::wstring & path)` method in C++.

In memory

In some instances the developer has the source document already in memory. "On the wire" email attachment processing is a good example of this. In these cases the document can be passed directly to the API without the need to persist it to storage. See the `SecureRequest.setOption(FileOption option, java.nio.ByteBuffer value)` method in Java, the `BFSetFileOptionUsingMemory` function in C or the `BFSecureRequest::SetOption(FileOptions option, void * memory, long length)` method in C++.

Anywhere else

Sometimes a file exists in a non-traditional storage medium that cannot be referenced by an operating system path. A file saved in a database BLOB is an example of this. In this case, the application can provide its own "channel" to the document by implementing a few simple functions like Read, Size, Close, etc. See the `SecureRequest.setOption(FileOption option, SimpleChannel value)` method in Java, the `BFSetFileOptionUsingChannel` function in C or the `BFSecureRequest::SetOption(FileOptions, BFChannel *)` method in C++.

5. Choose a method of providing the result document

If the source document is being scrubbed (not just analyzed) the developer must choose how the result document is saved.

Note that in the current system the result document is always the same length as the source document. This allows features like scrubbing a source document provided in memory, in place, since the block of memory does not need to expand. Future versions may break this rule and require some changes to the rules for "in memory, in place" scrubbing.

In place

The developer may choose to scrub the source document "in place" instead of creating a scrubbed copy. This will work regardless of which method was chosen in step 4 above. If the source document was provided through a channel, the channel's Write function must be implemented. See the `ScrubInPlace` option.

New file

The developer may choose to create a scrubbed copy of the source document. In this case, all the options outlined above for providing the source document are available for providing the result document through the `SecureOptions.ScrubbedDocument` option.

6. Choose scrub targets

One of the primary decisions a developer must make when integrating this API into their application is what specific, potentially insecure, parts of the document (called targets) they need or want to identify (analyze) and/or remove (scrub). The following is a breakdown of the most likely paths a developer can take. This decision should be made only after a thorough analysis of the available targets, your customer's need and your application's goals.

Analysis only

Some applications only need to analyze documents, not scrub them. In these cases, the developer has no need to decide which targets to scrub.

Fixed targets

Some applications will be so well focused that a single set of scrub targets can be decided upon at development time and 'hardwired' into the application. This can be done in one of two ways. Either by setting the targets individually when a SecureRequest is created, for example (using the Java interface):

```
SecureRequest request = new SecureRequest();
request.setOption(SecureOptions.AuthorHistory,ScrubOption.Action.SCRUB);
request.setOption(SecureOptions.DatabaseQueries,ScrubOption.Action.SCRUB);
request.setOption(SecureOptions.UserNames,ScrubOption.Action.SCRUB);
```

Or by saving a set of options to an XML file then loading it when the SecureRequest is created, for example:

```
SecureRequest request = new SecureRequest(new File("targets.xml"));
```

The later has the advantage that even the 'hardwired' targets can be changed by shipping the customer a different options file or modifying the file in the field.

Target sets

Many applications will want to offer its users more control, especially if those users are IT or IS professionals. One way to do this is to support a small number of named option sets. For example, an email gateway application might have sets like "Outbound email", "Internal email", "Executive email", etc. which could be used based on the type, source, destination, author, etc. of the email or any other policy.

As with the "Fixed targets" above, such sets could be 'hardwired' directly into the application or be pulled from options files.

Full control

Some applications may wish to offer administrators or end users full control of what targets get scrubbed. Such an application would present as part of its management interface all the scrub targets allowing the user to pick which ones will be scrubbed. This model may be combined with the "Target sets" model above to give users the ability to define their own sets and linked into whatever policy system the application uses.

To facilitate such a model all the options (including the scrub targets) have associated human readable names and descriptions. See Option and TargetOption for details.

7. Decide on extraction capabilities

Some applications may need the text or other information from the document for further processing such as dirty word search, entity identification, indexing, etc. If your application does NOT need this feature skip to step 9.

8. Choose an output type for extraction

If your application needs the text in the document the primary decision to make is how your application will get access to the text and other data the API provides.

Through an ElementHandler class

When the application can directly process the text and has no need for a "persisted" version this is by far the best performing and most integrated choice. By providing the API with an instance of ElementHandler interface the developer can process the text and other elements directly as they are produced in a very XML/SAX like fashion.

To an XML file

In some cases an application may want to receive the data in "true" XML form. In this case the same data that would be provided through the ElementHandler can be written to an XML file. The API provides several ways for the developer to provide the output "XML file" including a way for the developer to intercept the write calls so the "XML file" never needs to touch the disk or other storage device.

To a text file

In a few cases an application may want to receive the data as a simple stream of UNICODE text. In this case some of the document's structural elements may be lost or transformed (for example table cell and row breaks into 0x0D characters). The API provides several ways for the developer to provide the output "text file" including a way for the developer to intercept the write calls so the "text file" never needs to touch the disk or other storage device.

9. Decide on other options and features

The API supports many other options and features. Some of the major ones to consider are:

Embedding recursion

This feature recursively processes embedded objects of types that are supported. For example, if an Excel spreadsheet is embedded in a Word document, it can be recursively processed using the same options as the parent document. That is if the Word document is being scrubbed the Excel embedding will be scrubbed also, if text is being extracted from the Word document, text will be extracted from the Excel embedding and so on. Options include which formats to recur into

(`ExtractOptions.EmbeddingRecurseList`) and how deep the recusion should go (`ExtractOptions.EmbeddingRecurseDepth`).

Embedding export

The feature allows embedded objects and images to be exported from the document as standalone files. This allows the developer to further process those documents and images for whatever purpose. For example, a developer might have a seperate technology that knows how to deal with Microsoft Visio documents. In this case, embedded Visio documents can be exported and processed externally as standalone files. Options include which formats to export (`ExtractOptions.EmbeddingExportList`), directory where the exported files should go (`ExtractOptions.EmbeddingExportDirectory`) and what the base name of the exported files should be (`ExtractOptions.EmbeddingExportBaseFileName`).

10. Choose a threading model

In deciding how to introduce this API into your code, one major factor to consider is how your application uses or will use threads to process documents. While a complete discussion of this topic is outside the scope of this document, the following guidelines may provide some direction.

- For each thread in which you want to process documents, create a separate SecureRequest object and reuse it for multiple documents but only within the thread that created it. This is not a hard requirement but it is the safest way to proceed.
- Do not process documents in threads handling UI activity unless you are willing to force the user (and the UI) to wait for completion of the analysis or scrub.
- A very general performance guideline is to use four threads per processor to process documents. This assumes (perhaps incorrectly) that the developer wants all the power of the machine directed at analysis or scrubbing. This is a guideline only. Actual scalability is based on a large number of factors including the machine architecture, IO subsystem performance, exactly what API calls are used, if an result document is being produced, if reporting is enabled and many other factors.
- If the guidelines above are unclear or do not cover your situation contact Stellent support for assistance.

11. Code the integration

Once the decisions above are made, actually coding to this API should be a fairly easy task. The individual API documents for Java and C/C++ have numerous examples of the basic process of creating a SecureRequest, populating it with options, executing the request and getting results.

12. Test the integration

The **`samplefiles`** subdirectory structure includes documents that contain all the scrub targets in various combinations. These documents are a good place to start when testing the basic analysis or scrub functionality in your application. These documents, however,

do not represent complete code coverage of Stellent's technology and should be used only as a starting point for your testing process.