IDOL Panopticon

Software Version 12.10

Panopticon C Programming Guide

MICRO FOCUS

Document Release Date: October 2021
Software Release Date: October 2021

Legal notices

Copyright notice

Documentation updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for updated documentation, visit https://www.microfocus.com/support-and-services/documentation/.

Support

Visit the MySupport portal to access contact information and details about the products, services, and support that Micro Focus offers.

This portal also provides customer self-solve capabilities. It gives you a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the MySupport portal to:

- Search for knowledge documents of interest
- Access product documentation
- View software vulnerability alerts
- Enter into discussions with other software customers
- Download software patches
- Manage software licenses, downloads, and support contracts
- Submit and track service requests
- Contact customer support
- View information about all services that Support offers

Many areas of the portal require you to sign in. If you need an account, you can create one when prompted to sign in. To learn about the different access levels the portal uses, see the Access Levels descriptions.

Contents

Chapter 1: Introduction

This guide is for developers who want to incorporate Micro Focus KeyView Panopticon into their applications using a C/C++ development environment. It is intended for readers who are familiar with C/C++.

Overview

Micro Focus Panopticon enables you to decrypt files that have been protected by Microsoft Azure Rights Management System (RMS), which is part of Azure Information Protection, allowing your workflow to operate on the original, unencrypted file. You can use Panopticon with existing workflows to allow complete access to protected data for which the service has permission.

Panopticon is part of the KeyView suite of products. KeyView provides high-speed text extraction, conversion to web-ready HTML and well-formed XML, and high-fidelity document viewing.

Features

KeyView Panopticon enables the following features:

- Automatic detection of encryption type.
- Decryption of most file formats protected with RMS (text-only for PDF).

Known Limitations

Panopticon decrypts most RMS encrypted documents, with the following known limitations:

- Text is decrypted from RMS protected PDFs, but not text formatting, images or subfiles.
- Email clients such as Microsoft Outlook can protect email messages as rights-managed email

messages. In these cases, it stores the contents of the original message as an encrypted rpmsg attachment. Panopticon does not support decryption of these encrypted attachments.

Requirements

This section describes the supported platforms, compilers, and dependencies for Panopticon.

Supported Platforms

Panopticon is supported on the following platforms:

Microsoft Windows x86 64

- Windows Server 2019
- Windows Server 2016
- Windows Server 2012
- Windows 10
- Windows 7 SP1
- Windows Server 2008 R2
- Windows Server 2008 SP2

Linux x86 64

The minimum supported versions of particular Linux distributions are:

- Red Hat Enterprise Linux (RHEL) 6
- CentOS 6
- SuSE Linux Enterprise Server (SLES) 12

Supported Compilers

Platform	Architecture	Compiler Name	Compiler Version
Microsoft Windows	x86 64	cl	Microsoft C/C++ Optimizing Compiler for x64 Version 17 (Visual Studio 2012) to Version 19 (Visual Studio 2019).
Linux	x86 64	gcc/g++	4.1.0 to 4.9.2

Software Dependencies

To run Panopticon on Windows requires the Microsoft Visual C++ 2019 redistributables to be installed.

To run Panopticon on Linux/UNIX platforms requires `libstdc++.so.6` and `libgcc_s.so.1` from GCC 5.4. For your convenience, these are provided in the `redist` folder of your Panopticon installation.

> **NOTE:** The `kvoop`, `servant`, and `WKOOP` executables must be able to link to `libstdc++.so.6` and `libgcc_s.so.1`.
>
> - If these are installed in a system folder, like `/lib64`, KeyView will find them automatically.
> - If you prefer you can add the path of the folder containing these libraries to the environment variable `LD_LIBRARY_PATH`.

Some components require specific third-party software:

- A Java Development Kit (JDK) corresponding to OpenJDK version 1.8 or later is required to use the Java API.

Windows Installation

To install the Panopticon SDK on Windows, use the following procedure.

To install the Panopticon SDK

1. Run the installation program, `Panopticon_`*`VersionNumber`*`_`*`Platform`*`.exe`, where *`VersionNumber`* is the product version number, and *`Platform`*is the operating system platform.

   For example:

   `Panopticon_12.10_Windows_X86_64.exe`

   The installation wizard opens.
2. Read the instructions and click **Next**.

   The License Agreement page opens.
3. Read the agreement. If you agree to the terms, click I accept the agreement, and then click **Next**.

   The Installation Directory page opens.
4. Select the directory in which to install Panopticon. To specify a directory other than the default, click , and then specify another directory. After choosing where to install Panopticon, click **Next**.

   The Pre-Installation Summary opens.
5. Review the settings, and then click **Next**.

   The SDK is installed.
6. Click **Finish**.

UNIX Installation

To install the Panopticon SDK, use one of the following procedures.

To install the Panopticon SDK from the graphical interface

- Run the installation program and follow the on-screen instructions.

To install the Panopticon SDK from the console

1. Run the installation program from the console as follows:

   `./Panopticon_`*`VersionNumber_Platform`*`.exe --mode text`

   where:

VersionNumber	is the product version number.
Platform	is the name of the platform

2. Read the welcome message and instructions and press `Enter`.

   The first page of the license agreement is displayed.

3. Read the license information, pressing `Enter` to continue through the text. After you finish reading the text, and if you accept the agreement, type **Y** and press `Enter`.

   You are asked to choose an installation folder.

4. Type an absolute path or press `Enter` to accept the default location.

   The Pre-Installation summary is displayed.

5. If you are satisfied with the information displayed in the summary, press `Enter`.

   The SDK is installed.

Package Contents

The Panopticon installation contains:

- Libraries and executable files necessary for detecting the encryption type and decrypting files.
- The include files that define the functions and structures used by applications to establish an interface with Panopticon.
- A C sample program that demonstrates Panopticon functionality. See Panopticon Sample Program, on page 11.
- (Windows only) Microsoft Visual C++ 2019 redistributable files.

- The Java API (the JAR package `Panopticon.jar`), Javadoc documentation, and a sample program written in Java.

Chapter 2: Use Panopticon

This section describes how to perform some basic tasks using Panopticon.

Decrypt Microsoft Azure RMS Protected Files

This section describes the steps required to use Panopticon to decrypt files protected with Microsoft Azure Rights Management System (RMS).

To use Panopticon

1. Dynamically load the `panopticon` shared library.
2. Obtain a handle to KVPanopticonGetInterface().
3. Obtain function pointers for the library methods by calling KVPanopticonGetInterface().
4. Initialise Panopticon by calling init().
5. Configure Panopticon to use the RMS credentials for your application by calling configureRMS().
6. Determine the level of support for decrypting a particular file by calling encryptionInfo().
7. Free the memory allocated by encryptionInfo(), by calling resetEncryptionInfo(), on page 19.
8. If decryption is supported, decrypt the file by calling decryptFile().
9. Repeat steps 6 to 8 for any additional files.
10. Terminate the Panopticon session by calling shutdown().
11. Free the `panopticon` shared library.

Configure the Proxy for RMS

When Panopticon needs to access contents that are protected by RMS, it must make HTTP requests. By default, Panopticon uses the system proxy settings for these requests.

To use different proxy settings, you can configure them in the `[RMS]` section of the `cryptographyservices.cfg` configuration file. The following table describes the available options.

Parameter	Description
UseSystemProxy	Whether to obtain details about your HTTP proxy from the system. By default, this parameter is set to **TRUE**, which means: • On Microsoft Windows platforms, KeyView reads the proxy settings that are configured in the Windows Control Panel. • On Linux, KeyView reads the proxy settings from environment variables such as HTTP_PROXY and HTTPS_PROXY. You can use UseSystemProxy instead of setting the other proxy parameters (ProxyHost, ProxyPort, ProxyUsername, and ProxyPassword). When UseSystemProxy is set to **TRUE**, you must remove these other parameters from your configuration. Set UseSystemProxy to **FALSE** to use different proxy settings. In this case you must set at least ProxyHost and ProxyPort.
ProxyHost	The host name or IP address of the proxy server.
ProxyPassword	The password to use to authenticate with the proxy server.
ProxyPort	The port of the proxy server to use to access the repository. This port must be greater than 0, and less than 65535.
ProxyUsername	The user name to use to authenticate with the proxy server.

Panopticon Sample Program

Panopticon includes a sample program, written in C++, which demonstrates how to use Panopticon to decrypt RMS protected files. The sample program is called panopticon_test. It is intended to provide a starting point or reference for your own applications.

The source code and makefiles are provided in the samples/panopticon_test directory of your Panopticon installation directory.

The sample program passes license information to Panopticon by using init(). Before you can compile, you must replace the parameters YOUR_LICENSE_ORGANIZATION and YOUR_LICENSE_KEY in the init() function call with your license information.

To compile the sample program, use the makefiles provided in the program directory. You must ensure that the Panopticon include directory is in the include path of the project.

After you compile and build the executable, you must place it in the same directory as the Panopticon library.

> **NOTE:** A compiled executable is provided in the PLATFORM/bin directory. This sample has an embedded trial license, which expires approximately five months after release.

The panopticon_test sample program gets the encryption information for a file. If decryption is supported, it then decrypts the file.

The sample program includes the following files:

- `panopticon_test.cpp`. Contains the command-line interface.
- `panopticon_interface.cpp`. Contains a C++ class, which wraps the Panopticon C interface.
- `utils.cpp`. Contains utility functions used by `panopticon_interface.cpp`.

To run `panopticon_test`

1. Open a command prompt in the `bin` folder that contains the Panopticon library.
2. Type the following command:

   `panopticon_test [*options*] *credentialsfile inputfile outputfile*`

   The following table describes these arguments.

options	Zero or more of the options listed in the table Options for the panopticon_test sample program, below.
credentialsfile	The full path and file name of a file that contains the RMS credentials to use. This file must contain just the tenant ID, client ID, and client secret, in that order, separated by new lines.
inputfile	The full path and file name of the file to decrypt.
outputfile	The full path and file name to use for the decrypted file.

The following table describes the optional command-line arguments for the `panopticon_test` sample program.

Options for the `panopticon_test` sample program

Option	Description
-t *tempdir*	A temporary directory where Panopticon stores the temporary files that it generates. By default, it uses the system default temporary directory.

Chapter 3: Panopticon API Functions

This section describes the functions available in the Panopticon C API.

configureRMS()

This function provides a way to set the credentials required to access RMS protected files. After you set these credentials, the decryptFile() function is able to produce an unencrypted version of the RMS file.

> **CAUTION:** When Panopticon functions access the protected contents of RMS protected files, KeyView might place decrypted contents into the temporary directory. You can specify the temporary directory when you call init().

Syntax

```
KVPanopticonError configureRMS(
   KVPanopticonContext* const context,
   const KVRMSCredentials* const rmsCredentials
);
```

Arguments

`context`	A pointer to `KVPanopticonContext`, initialized by calling init().
`rmsCredentials`	A pointer to a `KVRMSCredentials` structure that contains the required credentials. You can store only one set of credentials at a time. You can call the function again with new credentials to override the existing configuration. Set this value to NULL to discard the existing credentials. Before you fill out the `KVRMSCredentials` structure, initialize the KVStructHead structure by using the macro `KVStructInit`.

Returns

If the function was successful, it returns `KVP_Success`. Otherwise, it returns a KVPanopticonError value describing the problem.

decryptFile()

This function removes the encryption on a protected file, giving access to the original, unencrypted file.

Syntax

```
KVPanopticonError decryptFile(
   KVPanopticonContext* const context,
   const char* const inputFilePath,
   const char* const outputFilePath
);
```

Arguments

`context`	A pointer to `KVPanopticonContext`, initialized by calling init().
`inputFilePath`	A null-terminated C string that contains the path of the file to decrypt.
`outputFilePath`	A null-terminated C string that contains the path of the output file to create. If a file already exists at this location, Panopticon overwrites it.

Returns

If the function was successful, it returns `KVP_Success`. Otherwise, it returns a KVPanopticonError value describing the problem.

Discussion

- To decrypt a protected file, Panopticon must make an HTTP request.
- By default, Panopticon uses the system proxy when it makes HTTP requests. You can also specify the proxy manually in the `cryptographyservices.cfg`. See Configure the Proxy for RMS, on page 10.
- This function returns an error if decryption is not supported for the `inputFile` you provide. You can obtain information about the level of support provided by using encryptionInfo().

encryptionInfo()

This function detects the type of encryption applied to a document, and information about the level to which Panopticon supports decryption. This information can be used to determine whether this file should be passed to decryptFile().

Syntax

```
KVPanopticonError encryptionInfo(
   KVPanopticonContext* const context,
   const char* const inputFilePath,
   KVPanopticonEncryptionInfo* const encryptionInfo /*out*/
);
```

Arguments

`context`	A pointer to `KVPanopticonContext`, initialized by calling init().
`inputFilePath`	A null-terminated C string that contains the path of the file to get encryption info for.
`encryptionInfo`	A pointer to a KVPanopticonEncryptionInfo. If the function completes successfully, it fills this structure out with the encryption information. You must initialize the KVStructHead structure by using the macro `KVStructInit`.

Returns

If the function was successful, it returns `KVP_Success`. Otherwise, it returns a KVPanopticonError value describing the problem.

Discussion

After you call this function, you must free the memory it allocates by using resetEncryptionInfo(), on page 19 before you call shutdown(), on page 21

init()

This function initializes a Panopticon session. If initialization is successful, the pointer that `context` points to is set to a valid context identifier. You must pass this context identifier as the first parameter to all other Panopticon functions.

Syntax

```
KVPanopticonError init(
   const char* const binDir,
   const char* const tempFolder,
   const char* const licenseOrganisation,
   const char* const licenseKey,
   KVPanopticonContext** const context /*out*/
);
```

Arguments

`binDir`	A null-terminated C string that contains the path of the directory where the Panopticon components are located.
`tempFolder`	(Optional) A null-terminated C string that contains the path of directory to use to store temporary files. Set this value to NULL to default to the system temporary directory.
`licenseOrganisation`	A pointer to a string that contains the organization name under which this installation of Panopticon is licensed. This value is the company name that appears at the top of the license key that Micro Focus provides. Add the text exactly as it appears in this file.
`licenseKey`	A pointer to a string that contains the license key for this installation of KeyView. This value is the appropriate license key provided by Micro Focus. The key is a string that contains 31 characters, for example 2TAD22D-2M6FV66-2KBF23S-2QEM5AB. Supply these characters exactly as they appear in the license key file, including the dashes. Do not include any leading or trailing spaces.
`context`	A valid pointer to a NULL pointer of type `KVPanopticonContext`. If initialization is successful, this target is set to a context-identifying value, which you must supply to subsequent Panopticon functions.

Returns

If the function was successful, it returns `KVP_Success`. Otherwise, it returns a KVPanopticonError value describing the problem.

KVPanopticonGetInterface()

This function is exported by the `panopticon` shared library. It supplies function pointers to the other Panopticon functions. When you call `KVPanopticonGetInterface()`, it assigns the function pointers to the structure pointed to by `panopticonInterface`.

Syntax

```
KVPanopticonError KVPanopticonGetInterface(KVPanopticonInterface* const
panopticonInterface);
```

Arguments

`panopticonInterface`	A pointer to the structure KVPanopticonInterface. You must initialize the KVStructHead structure by using the macro `KVStructInit`. This process sets the version number of the Panopticon API, and supports binary compatibility with future releases.

Returns

If the function was successful, it returns `KVP_Success`. Otherwise, it returns a KVPanopticonError value describing the problem.

Discussion

When you load this function from the Panopticon shared library, you can use the `typedef KV_PANOPTICON_GET_INTERFACE`, which is provided in `panopticon.h`.

resetEncryptionInfo()

This function frees the memory allocated by encryptionInfo(), on page 16. You can reuse a KVPanopticonEncryptionInfo, on page 23 structure after you have passed it to `resetEncryptionInfo()`.

Syntax

```
KVPanopticonError resetEncryptionInfo(
   KVPanopticonContext* const context,
   KVPanopticonEncryptionInfo* const encryptionInfo
);
```

Arguments

`context`	A pointer to `KVPanopticonContext`, initialized by calling init().
`encryptionInfo`	A pointer to a KVPanopticonEncryptionInfo. If the function completes successfully, it resets this structure, freeing all the memory allocated by encryptionInfo(), on page 16. You must initialize the KVStructHead structure by using the macro `KVStructInit`.

Returns

If the function was successful, it returns `KVP_Success`. Otherwise, it returns a KVPanopticonError value describing the problem.

shutdown()

This function terminates a Panopticon session that was initialized by init(), and frees allocated system resources. You must call this function when the Panopticon context is no longer required.

Syntax

```
KVPanopticonError shutdown(KVPanopticonContext** const context);
```

Arguments

`context`	A pointer to a pointer of type `KVPanopticonContext`, initialized by calling init(). This function sets the target to NULL to prevent accidental reuse of the context value that it contained, which must not be subsequently passed to any Panopticon function.

Returns

If the function was successful, it returns `KVP_Success`. Otherwise, it returns a KVPanopticonError value describing the problem.

Discussion

- The context value pointed to by context can be NULL.

Chapter 4: Panopticon Structures

This section describes the data structures used by the Panopticon C API.

KVPanopticonEncryptionInfo

This structure is filled out by encryptionInfo(), and provides information about what type of encryption the document has, the level of support Panopticon provides for that document, and (if applicable) a suggested name for the file after decryption.

```
typedef struct tag_KVPanopticonEncryptionInfo
{
   KVStructHeader;
   KVPanopticonEncryption encryptionType;
   KVPanopticonDecryptionSupport decryptionSupport;
   char* suggestedOutputName;
} KVPanopticonEncryptionInfo;
```

Member Descriptions

`KVStructHeader`	The KeyView version of the structure. See KVStructHead, on page 26.
`encryptionType`	A KVPanopticonEncryption value describing the type of encryption used to protect the document.
`decryptionSupport`	A KVPanopticonDecryptionSupport value describing the level of support Panopticon provides for this document.
`suggestedOutputName`	A suggested name for the file after it has been decrypted. This name is typically the same as the original file name, with the extension changed to the corrected extension for the decrypted file. In some cases, the suggested output name is identical to the original file name. When this occurs, ensure that you decrypt to a different path. If Panopticon does not support decrypting the file, or it could not unambiguously determine the decrypted file type, `suggestedOutputName` is NULL.

KVPanopticonInterface

This structure contains pointers to the Panopticon API functions. You can set the pointers by calling the KVPanopticonGetInterface() function.

```
typedef struct tag_KVPanopticonInterface
{
   KVStructHeader;
   KV_PANOPTICON_INIT init;
   KV_PANOPTICON_CONFIGURE_RMS configureRMS;
   KV_PANOPTICON_ENCRYPTION_INFO encryptionInfo;
   KV_PANOPTICON_DECRYPT_FILE decryptFile;
   KV_PANOPTICON_SHUTDOWN shutdown;
} KVPanopticonInterface;
```

Member Descriptions

`KVStructHeader` The KeyView version of the structure. See KVStructHead, on page 26.

The subsequent members of this structure are the pointers to the API functions. See Panopticon API Functions, on page 13.

KVRMSCredentials

This structure defines each element of the RMS credentials. This structure is defined in `kvdecryptionsettings.h`.

```
typedef struct _KVRMSCredentials
{
    KVStructHeader;

    const char* tenantID;
    const char* clientID;
    const char* clientSecret;
}
KVRMSCredentials;
```

Member Descriptions

`KVStructHeader`	The Panopticon version of the structure. See KVStructHead, on the next page.
`tenantID`	The tenant ID of the domain.
`clientID`	The client ID of the application.
`clientSecret`	The client secret for the application.

For Panopticon to access the protected contents of Microsoft Azure Rights Management System (RMS) protected files, your end-user application must be registered on the relevant Azure domain. For more information about how to register an app, refer to the Microsoft documentation: https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app.

After you register an application, you can find the client and tenant IDs in the Azure Portal, in the Overview section. You can find the client secret in the Certificates & Secrets section.

> **CAUTION:** This information is linked to the domain itself, rather than to a specific user. Providing this information allows Panopticon to access the contents of all files protected by this domain. Therefore you must handle these three pieces of information securely.

KVStructHead

This structure contains the current KeyView version number, and is the first member of other structures. It enables Micro Focus to modify the structures in future releases, but to maintain backward compatibility. Before you initialize a structure that contains the `KVStructHead` structure, use the macro `KVStructInit` to initialize `KVStructHead`. The structure and macro are defined in `kvstructhead.h`.

```
typedef struct _KVStructHead
{
   WORD        version;
   WORD        size;
   DWORD       reserved;
   void        *internal;
}
KVStructHeadRec, *KVStructHead;
```

Member Descriptions

`version`	The current KeyView version number. This is a symbolic constant (KeyviewVersion) defined in `kvstructhead.h`. This constant is updated for each KeyView release.
`size`	The size of the `KVStructHeadRec`.
`reserved`	Reserved for internal use.
`internal`	Reserved for internal use.

Example

```
KVPanopticonEncryptionInfo encryptionInfo;
KVStructInit(&encryptionInfo);
```

Chapter 5: Enumerated Types

This section provides information on some of the enumerated types used by the Panopticon C API.

Programming Guidelines

In future releases of KeyView Panopticon, some enumerated types might be expanded. For example, new encryption types might be added to KVPanopticonEncryption, or new error codes might be added to KVPanopticonError. When you use these expandable types, your code must ensure binary compatibility with future releases.

KVPanopticonDecryptionSupport

```
typedef enum tag_KVPanopticonDecryptionSupport
{
   DecryptionNotSupported,
   TextOnlyDecryption,
   FullDecryption
} KVPanopticonDecryptionSupport;
```

Enumerators

`DecryptionNotSupported`	Decryption is not supported
`TextOnlyDecryption`	Panopticon cannot decrypt the file to an unencrypted file. Instead it can decrypt any text content in the file and write it to a file of the same type as the original.
`FullDecryption`	Panopticon can decrypt the file.

KVPanopticonError

This enumerated type defines the type of error generated if Panopticon fails.

```
typedef enum tag_KVPanopticonError
{
   KVP_Success = 0,
   KVP_ERR_GeneralError = 1,
   KVP_ERR_MemoryError = 2,
   KVP_ERR_InvalidArguments = 3,
   KVP_ERR_StructureNotInitialised = 4,
   KVP_ERR_LicenseInvalid = 5,
   KVP_ERR_LicenseExpired = 6,
   KVP_ERR_DllNotFound = 7,
   KVP_ERR_DllLoadFailed = 8,
   KVP_ERR_TempFolderDoesNotExist = 9,
   KVP_ERR_CreateTempFileFailed = 10,
   KVP_ERR_InputFileNotFound = 11,
   KVP_ERR_FormatNotRecognised = 12,
   KVP_ERR_ParseEncryptedFileError = 13,
   KVP_ERR_WriteDecryptedFileError = 14,
   KVP_ERR_CannotCreateOutputFile = 15,
   KVP_ERR_DecryptionNotSupported = 16,
   KVP_ERR_InvalidConfig = 17,
   KVP_ERR_ConnectionFailure = 18,
   KVP_ERR_RMS_DecryptionFailed = 100,
   KVP_ERR_RMS_NotConfigured = 101,
   KVP_ERR_RMS_InvalidFileStructure = 102,
   KVP_ERR_RMS_MicrosoftServerError = 103

} KVPanopticonError;
```

Enumerators

`KVP_Success`	The function completed successfully.
`KVP_ERR_GeneralError`	General error.
`KVP_ERR_MemoryError`	A memory error occurred.
`KVP_ERR_InvalidArgument`	An argument to a Panopticon API function was invalid. For example, a required pointer was NULL.
`KVP_ERR_StructureNotInitialised`	A structure passed to a Panopticon API function was invalid. All structures containing a KVStructHead member must be initialized with `KVStructInit`.

`KVP_ERR_LicenseInvalid`	The license provided to init() was invalid.
`KVP_ERR_LicenseExpired`	The license provided to init() has expired.
`KVP_ERR_DllNotFound`	A DLL or shared library was not found.
`KVP_ERR_DllLoadFailed`	A DLL or shared library failed to load correctly.
`KVP_ERR_TempFolderDoesNotExist`	The specified temp folder does not exist.
`KVP_ERR_CreateTempFileFailed`	Panopticon was unable to create a temporary file in the temp folder.
`KVP_ERR_InputFileNotFound`	The specified input file was not found.
`KVP_ERR_FormatNotRecognised`	Panopticon did not recognize the file format of the specified input.
`KVP_ERR_ParseEncryptedFileError`	During text-only decryption, Panopticon was unable to obtain the text content from the file.
`KVP_ERR_WriteDecryptedFileError`	During text-only decryption, Panopticon was unable to write the decrypted content to a new file.
`KVP_ERR_CannotCreateOutputFile`	An output file could not be created at the specified location.
`KVP_ERR_DecryptionNotSupported`	Decryption of the specified input file is not supported.
`KVP_ERR_InvalidConfig`	The Panopticon configuration file is invalid.
`KVP_ERR_ConnectionFailure`	A required HTTP call was not successful.
`KVP_ERR_RMS_DecryptionFailed`	Decryption of the RMS encrypted file failed.
`KVP_ERR_RMS_NotConfigured`	decryptFile() was called on an RMS encrypted file, without credentials being supplied through configureRMS().
`KVP_ERR_RMS_InvalidFileStructure`	The structure of the input file was not valid.
`KVP_ERR_RMS_MicrosoftServerError`	Microsoft Server Error (Request returned HTTP 500).

KVPanopticonEncryption

This enumerated type defines the type of encryption used to protect the file.

```
typedef enum tag_KVPanopticonEncryption
{
   NoEncryptionDetected,
   OtherEncryption,
   RMSEncryption,
   SecloreEncryption
} KVPanopticonEncryption;
```

Enumerators

`NoEncryptionDetected`	No encryption was detected.
`OtherEncryption`	The file is encrypted.
`RMSEncryption`	The file is encrypted using RMS.
`SecloreEncryption`	The file is encrypted using Seclore encryption.
`SmartCipherEncryption`	The file is encrypted using SmartCipher encryption.

Send documentation feedback

If you have comments about this document, you can contact the documentation team by email. If an email client is configured on this system, click the link above and an email window opens with the following information in the subject line:

Feedback on Micro Focus IDOL Panopticon 12.10 Panopticon C Programming Guide

Add your feedback to the email and click **Send**.

If no email client is available, copy the information above to a new message in a web mail client, and send your feedback to swpdl.idoldocsfeedback@microfocus.com.

We appreciate your feedback!