SOPHOS

Security made simple.

# SAV Interface Developer Toolkit user manual

Product version: 4.9

Document date: April 2015

# Contents

# 1 Document revision history

This table describes the changes to the SAV Interface Developer Toolkit user manual.

| Revision date | Summary of changes |
|---|---|
| January 9, 2015 | ▪ Updated information about installing Sophos Anti-Virus and SAVI. See About SAVI (page 7).<br>▪ Updated the description of the return values for the ReadStream function of the ISaviStream callback interface. See ReadStream (page 110). |
| April 9, 2015 | ▪ Updated for version 4.9. |

# 2 About SAVI

This section introduces SAVI and describes

- the SAVI programming paradigm
- SAVI programming methods
- how to use SAVI
- the SAVI threading model
- Macintosh file handling
- SAVI version 3 changes
- SAVI version 4 changes

**Note:**

Sophos Anti-Virus must be installed before third-party applications can call the SAVI interface. To install Sophos Anti-Virus, see the Sophos Anti-Virus installation documentation for your platform (available from www.sophos.com/en-us/support/documentation).

The Sophos Anti-Virus Interface (SAVI) is also available in a standalone SAVI package, which you will need to install before third-party applications can call the SAVI interface.

## 2.1 What is SAVI?

SAVI is an application programming interface (API) that enables software developers to integrate Sophos Anti-Virus with their applications.

## 2.2 The SAVI programming paradigm

The SAVI interface conforms to Microsoft's Component Object Model (COM) specification. This means that SAVI clients on Windows operating systems can access it using standard COM functions such as CoCreateInstance().

SAVI can be programmed without COM on any SAVI-supported platform. However, on platforms that support it, it is simpler to use COM.

## 2.3 SAVI programming methods

### 2.3.1 Sophos standard data types

To allow cross-platform development Sophos defines standard data types very strictly. Refer to the header file s_types.h for a full description.

### 2.3.2 String manipulation

All string information (including paths) passed to and from SAVI is defined in terms of pointers to OLE strings (LPOLESTR). On platforms where these types do not exist they are redefined to platform-specific standard types.

On Windows platforms, OLE strings consist of 16-bit Unicode characters. You must therefore build SAVI client applications with Unicode support.

When writing cross-platform code use the macro SOPHOS_COMSTR to translate strings to the appropriate data type for functions taking LPOLESTR.

Functions that write information into buffers supplied by the client usually enable the client to ask for the amount of buffer space required.

## 2.4 How to use SAVI

SAVI consists of a set of interfaces and enumerators that provide access to various objects used internally by SAVI. For a full listing, see SAVI Interfaces (page 31).

The interfaces are retrieved

- by making a call to the SAVI library entry point function, or
- by allowing COM to supply them automatically (only when accessing SAVI through a COM subsystem, e.g. on Win32 platforms), or
- from the SAVI interface itself.

### 2.4.1 Using C++ syntax or C syntax

The interfaces can be used with C++ syntax or C syntax. The C++ syntax is generally simpler, but is currently supported only on Windows platforms.

There are various differences between C and C++ syntax. In C syntax

- an additional parameter, a pointer to the interface structure, has to be supplied as the first parameter.
- interface member functions are accessed via the structure's pVtbl member.
- interface IDs (IIDs) cannot use the C++ reference syntax and so must be passed using explicit 'address of' syntax e.g. &SOPHOS_IID_SAVI3.

If you are using C++ you should include the isavi3.h header file in your client source. This contains a declaration for each SAVI interface in the form of a C++ abstract base class. Interface member functions should be called using the usual C++ syntax, for example

```
HRESULT hr = pSAVI->Initialise();
```

If you are using C you should include the csavi3c.h header file. This contains a declaration for each SAVI interface in the form of structures of function pointers. Interface member functions are then called using C pointer syntax, for example

```
HRESULT hr = pSAVI->pVtbl->Initialise(pSAVI);
```

### 2.4.2 Calling SAVI from other languages

On Microsoft Windows operating systems SAVI is implemented as a COM object which conforms fully with Microsoft's COM specification. However, this does not guarantee that SAVI objects can be called from all languages which nominally offer support for COM. Microsoft Visual Basic, for example, might link to a COM object using the "Automation" approach. This is implemented in COM objects by offering an interface called IDispatch. SAVI does not currently implement this interface.

Other languages which require Automation support in order to use COM objects will also be unable to use SAVI directly.

### 2.4.3 Initialising GUIDs

Applications which use SAVI tell the SAVI library (either directly or via the COM subsystem on Microsoft Windows) which kind of object they want by passing a pointer to a Globally Unique ID (GUID). A GUID is a 16-byte data structure representing a 128-bit number which is guaranteed to be unique.

GUIDs are used in COM to represent object class IDs and interface IDs. Pointers to class IDs and interface IDs are defined as types REFCLSID and REFIID.

In order for the SAVI client application to generate a pointer to a GUID it is necessary for a GUID data structure to be declared and initialised with values in the client code. This need only be done in one client source module: other modules in the client application can just declare an external pointer to this GUID structure.

The way that this is achieved in practice is that the GUIDs are defined in a header (SAVI GUIDs are defined in swiid.h). The GUIDS are declared in the header using a special macro, and the macro can have two forms. In one form it actually assigns the GUID symbol to some storage initialised with the 128-bit value. In the other form it just declares the symbol to be an external.

If building a SAVI client for a platform other than Microsoft Windows, the client code switches between these two forms by defining the symbol INITGUID before including swiid.h in just one of the source modules.

Under Microsoft Windows the situation is a little more complex in that the macros are defined in Microsoft-supplied headers, there are various methods for switching between the two forms, and these methods may vary depending on the compiler / SDK version. In some versions, defining INITGUID as described above will work. In some more recent versions, instead of defining INITGUID, it is necessary to include the Windows SDK headers objbase.h and initguid.h before including swiid.h, e.g. :

```
#include <objbase.h>  /* Windows platform SDK header. */
#include <initguid.h> /* Windows platform SDK header. */
#include “isavi3.h”  /* SAVI header - also includes swiid.h. */
```

### 2.4.4 Using enumerators

Much of SAVI's functionality is accessed through a special type of interface called an enumerator, which, when queried, returns a sequence of other interfaces.

The user asks the SAVI interface for a particular enumerator interface (e.g. ISweepResults) then iterates across it to query each underlying interface in turn. When an individual interface has been retrieved, the information stored can be accessed.

Typical use of an enumerator is to call Reset() initially, and then call Next() to retrieve one item at a time until SAVI returns SOPHOS_S_FALSE (i.e. function call unsuccessful).

The use of enumerators is shown in the examples in Using SAVI (page 22).

### 2.4.5 Testing results of SAVI function calls

All interface functions return values which indicate whether the function succeeded.

The simplest method for testing for success or failure is to use the SOPHOS_SUCCEEDED() or SOPHOS_FAILED() macros defined in the SAVI header files.

For example, to test whether SAVI was correctly initialised, use this code:

```
HRESULT hr = pSAVI->Initialise();
if (SOPHOS_FAILED(hr))
  puts("SAVI could not be initialised");
```

If more information than a simple success/failure test is required, the return value can be tested for the predefined values listed in the *SAV Interface Developer Toolkit supplement*. For example, calls to threat detection functions return a success code if a scan completes, whether a threat is found or not.

SAVI client applications should be designed to behave gracefully if they encounter new return codes not listed in the header file used at the time the application was compiled.

## 2.5 SAVI threading model

The Win32 implementation of SAVI includes multi-threaded support which makes it possible to call SAVI in a free-threaded fashion. However, developers writing non-Win32 or cross-platform code must use the apartment-threaded model, where the client should create and initialise a separate SAVI object for each thread. This is also recommended for Win32 platforms.

**Note:** The code samples supplied with the SAVI Developer Toolkit implement SAVI clients using the apartment-threaded model.

The underlying scanning engine relies on threat descriptions to detect threats. These threat descriptions are read in from a disk file, decompressed, indexed and stored in a set of tables. The procedure can be very demanding on system resources and so multiple SAVI objects created by a single client process share the threat data of the first SAVI object. The data stays in memory as long as the client process owns one or more SAVI objects. This is on a per-process basis – a separate client process will always have its own copy of the threat data.

Where a client application is actively creating and destroying multiple SAVI objects during the course of its execution, it will run much more efficiently if it is designed to keep at least one SAVI object alive throughout execution of the program. This object acts as an anchor that keeps threat data in memory.

**Note:** It is essential that the first SAVI object be allowed to finish loading the threat data before any other SAVI objects are given access to it. All operating systems currently supported by SAVI,

except for FreeBSD6, have thread protection to prevent problems in this area. If your application uses SAVI on a FreeBSD6 system, it must take responsibility for this. The safest approach is to create a SAVI object and establish threat data before creating any additional SAVI objects. The client must then ensure thread safety during any attempts to reload the threat data. These measures could include avoiding threat data reload altogether, but if a reload is necessary, activity should be halted on all other SAVI objects and threads until the reload is complete.

Multi-threaded SAVI client applications must ensure that, for each SAVI object, there is never more than one thread executing a SAVI interface function. This can be achieved by, for example, creating a separate SAVI object for each thread in an application.

Versions 3 and above of the SAVI library have been enhanced to allow the threat data to be reloaded by a running SAVI object. If this feature is used, multithreaded SAVI client applications must ensure that all scanning activity on all SAVI objects is halted until the reload is complete.

## 2.6 Macintosh file handling

On Apple Macintosh systems running versions of the Macintosh operating system earlier than Mac OS X, individual files are stored in two parts. These parts are known as the resource fork and the data fork. Both are considered to be part of the file, but the resource fork is used to save Macintosh resources such as executable code, menu definitions and icons. The data fork was primarily intended for user-supplied data, but may also contain executable code.

Although every pre-Mac OS X Macintosh file has resource and data forks, one or both can be empty, depending on the kind of file.

Support for this way of splitting a file's contents is not available on all operating/file systems. For example, the NTFS partitions available on Windows NT can save both forks, although the resource fork will not normally be visible if the file is listed from a Windows client.

However, many other file systems, including the FAT system used on Windows 95/98/Me and those common on Unix systems, cannot store both forks under one file name. These systems use various schemes to save the two forks as separate files. Some schemes, including encoding schemes like AppleSingle or BinHex, involve extracting both forks and saving them separately in a kind of archive file. Since email messages must be able to pass through many different kinds of system, this approach is used to send Macintosh files as email attachments.

From a threat-scanning perspective, native, executable Macintosh threats will generally be found in the resource fork. However, macro viruses, such as those targeting Microsoft Word files, will be located in the data fork. Non-Macintosh executable viruses only ever infect the data fork.

The SAVI NamespaceSupport setting allows a SAVI client to control which forks are scanned. However, this is only relevant on systems which support dual fork file storage. For these systems, the SAVI client can be configured to scan only for Macintosh executable (resource fork) threats by setting the option to SOPHOS_MAC_FILES. Alternatively, setting the option to SOPHOS_DOS_AND_MAC_FILES configures SAVI to scan both forks.

On all other systems, the SOPHOS_DOS_FILES setting will cause the single (data) fork to be scanned for all threat types.

**Note:** If SAVI scans a file created using one of the encoding schemes for separate resource and data forks, both forks are scanned, even if the file system does not natively support dual forks. See the list of engine configuration options in the SAV Interface Developer Toolkit supplement for details of the schemes supported by SAVI.

## 2.7 SAVI version 3 changes

This section outlines the main differences between SAVI2 and SAVI3. In particular it contains information for developers familiar with SAVI2 who would like to find out what's new in SAVI3.

Firstly it is important to draw attention to the distinction between the SAVI library (dll on Win32), a SAVI object, and the interfaces the object supports. A SAVI3 library will always create SAVI3 objects, just as the SAVI2 library always creates SAVI2 objects. This is true regardless of which interface is requested.

A SAVI3 object supports both the ISavi2 and ISavi3 interfaces, but the old SAVI2 object does not support the ISavi3 interface. ISavi3 is an extension of the ISavi2 interface. It contains the same functions, plus seven new ones that have been added to provide greater control and functionality.

The ISavi2 interface is supported for backwards compatibility in accordance with the principles of COM. When writing a SAVI3 client application, full functionality is obtained by utilising the ISavi3 interface and there is nothing to be gained by using ISavi2.

**Note:** To check if your library is SAVI2 or SAVI3, create a SAVI object and request an ISavi3 interface (see Initializing SAVI (page 14)). If this returns SOPHOS_E_NOINTERFACE, you have a SAVI2 library that does not support SAVI3.

### 2.7.1 Additional scanning functionality

SAVI3 includes three new scan/disinfect function pairs that enable SAVI3 to scan and disinfect file handles, memory buffers and client-implemented data streams. These functions are described in Using SAVI (page 22) and ISavi3 (page 60).

### 2.7.2 Threat data loading

SAVI3 also gives the client greater control of the underlying threat data, which is used to detect threats. The ISavi3 interface includes LoadVirusData() which can be called to release any existing data then reload. This enables threat data to be reloaded manually without shutting down or restarting either the client application or the SAVI object it owns.

SAVI3 also allows the location of threat identity files (IDEs) and the name and location of the threat data file to be modified. This is done via the configuration interface (see Configuring SAVI (page 16)). SAVI2 loads the threat data during initialisation, but as this can be a resource-consuming exercise, the load has been delayed for SAVI3 in order to give the client a chance to modify data locations or names. The data is now loaded either when it is required (i.e. when a Sweep...() function or GetVirusEngineVersion() is called) or when LoadVirusData() is called.

Following threat data load, if one of the threat data location configuration settings is subsequently changed then threat data will be loaded again from the new location the next time it is required.

If the new functionality is not required, a SAVI3 client can behave like a SAVI2 client as the threat data will be loaded automatically from the default location when needed (i.e. when a Sweep...() function or GetVirusEngineVersion is called). The only exception to this rule is if multiple SAVI objects are created by the same client process. In this case, LoadVirusData() should be called on the first SAVI object before creating any subsequent ones. This will avoid any confusion

between multiple SAVI objects that share the threat data and possible threading risks on some platforms (see Enumerator interfaces (page 71)).

**Note:** Note that any delay apparent while loading the threat data will now occur at the start of the first Sweep...() or GetVirusEngineVersion() call or during LoadVirusData(). Initialisation itself is now likely to be significantly faster than for SAVI2.

## 2.8 SAVI version 4 changes

### 2.8.1 New callback interfaces

Version 4 of the SAVI library introduces two new callback interfaces IChangeNotify and ISeverityNotify.

The first of these is used to allow a SAVI object to inform its client when a change has been made to threat data and/or the underlying threat engine. This change may have resulted from a call to LoadVirusData() made to this or another SAVI object in the same process. This will result in just the threat data changing. Win32 implementations have the ability to update both threat data and scanning engine without the SAVI client process being otherwise aware that this has happened (this is known as a “hot update”). In this situation a call is made to the IChangeNotify interface with details of both components (engine and data) having changed.

The ISeverityNotify callback interface is provided to give SAVI client code more fine-grained error reporting (i.e. there may be several errors generated internally but only one error code can be returned to the client from the SAVI interface function). The callback also provides additional information about the severity of the error. This may help client code take a decision on how to proceed from the error.

### 2.8.2 Automatic scan abort

The new "auto-stop" feature gives protection against some forms of malicious file which are designed to disrupt the action of AV scanners. These files (sometimes referred to as "zip bombs") usually take the form of otherwise innocent-looking archives which, when unpacked in order to scan, turn out to require enormous amounts of time, disk space or memory.

Detection of these files may be enabled by setting the EnableAutoStop option to "1". If a file of this sort is encountered then the scan is immediately aborted and the scan function returns the new error code SOPHOS_SAVI_ERROR_SCAN_ABORTED.

**Note:** Please note that detection of these files uses a set of heuristic tests and, as a result, it is possible to get the occasional false positive (i.e. scan is aborted on a genuine, non-malicious archive file).

# 3 Initializing SAVI

All access to Sophos Anti-Virus functionality is through the interfaces described in the isavi3.h (C++) or csavi3c.h (C) header files.

To gain access to these interfaces you must first create a SAVI interface object. The process of creation is simplified by using CoCreateInstance() on Win32 platforms that support COM. If you are not using COM, load the SAVI library and then call the DllGetClassObject() method to create a class factory object. Once the factory has been created it can be used to manufacture an instance of the SAVI interface.

Then initialize the scanning engine by calling Initialise() or InitialiseWithMoniker().

- Initializing the interface using COM and C++ (page 14) demonstrates how to do this using C++.
- Initializing the interface using C (page 15) demonstrates how to do this using C.

## 3.1 Initializing the interface using COM and C++

```
/ *
 * Initialise COM (Win32 only).
 * /
CoInitialize(NULL);
/ *
 * Create an instance of the SAVI interface.
 * /
ISavi3*  pSAVI;
HRESULT hr = CoCreateInstance( SOPHOS_CLASSID_SAVI,
                               NULL, CLSCTX_ALL,
                               SOPHOS_IID_SAVI3,
                               (void**)&pSAVI );
if( SOPHOS_SUCCEEDED (hr) ){
 LPCOLESTR ClientName = L”SAVI Demo”;
 / *
   * Initialise the SAVI interface
   * /
 hr = pSAVI->InitialiseWithMoniker(ClientName);
}
else if( pSAVI ){
 / *
   * Initialisation failed so clean up.
   * /
  pSAVI->Release();
  pSAVI = NULL;
}
```

## 3.2 Initializing the interface using C

```
CISavi3*              pSAVI;
CISweepClassFactory2* pFactory;
HRESULT               hr;
const OLECHAR*        ClientName =
                          SOPHOS_COMSTR(“SAVIDemo”);
/ *
  * Load the SAVI DLL and then request a class factory
  * interface.
  * /
hr = DllGetClassObject((REFCLSID)&SOPHOS_CLASSID_SAVI,
                  (REFIID)&SOPHOS_IID_CLASSFACTORY2,
                  (void**)&pFactory);
if( SOPHOS_SUCCEEDED (hr) ){
 / *
   * Ask the class factory for a CSAVI3 interface.
   * /
 hr = pFactory->pVtbl->CreateInstance(pFactory, NULL,
                                  &SOPHOS_IID_SAVI3,
                                  (void**)&pSAVI);
 / *
   * Drop the factory immediately, we don’t need it
   * again in this example.
   * /
 pFactory->pVtbl->Release(pFactory);
 / *
   * Was the CSAVI3 interface returned?
   * /
 if( SOPHOS_SUCCEEDED (hr) ){
   / *
     * Ask SAVI to initialise itself.
     * /
    hr = pSAVI->pVtbl->InitialiseWithMoniker(pSAVI,
                                          ClientName);
   / *
     * If the initialisation failed, release the
     * SAVI interface and set the pointer to NULL.
     * /
    if( SOPHOS_FAILED(hr) ) {
      printf(“ERROR: Initialise [%ld].”, (long)hr);
      pSAVI->pVtbl->Release(pSAVI);
      pSAVI = NULL;
   }
  }
 }
```

# 4 Configuring SAVI

This section describes how to configure SAVI scanning activity. For the full list of configuration options, see the *SAV Interface Developer Toolkit supplement*.

**Note:** The examples in this section are in C++ syntax and assume the code is compiled for Unicode on a Win32 platform. For the differences between C++ and C syntax, see Using C++ syntax or C syntax (page 8).

The values of SAVI configuration settings are passed across the SAVI interface as strings. For numeric settings the values are encoded as decimal strings. For more information about SAVI and strings see Sophos standard data types (page 7).

## 4.1 The configuration interface

By default SAVI will perform threat scanning in a mode recommended by Sophos. The configuration interface enables you to alter the behaviour of SAVI during the scanning process.

The configuration values are accessed via the SAVI interface. However, to get information about available configuration options, you must obtain an enumerator interface.

## 4.2 Retrieving configuration options

First acquire an enumerator interface from SAVI. Then use the enumerator interface to access each element in turn. Finally release the enumerator interface.

```
IEnumEngineConfig* pConfigEnum;
IEngineConfig*     pConfig;
HRESULT hr =  pSAVI->GetConfigEnumerator(
              SOPHOS_IID_ENUM_ENGINECONFIG,
              (void**)&pConfigEnum );
 / *
    * Reset the enumerator to the start of the list.
    * /
 HRESULT hr = pConfigEnum->Reset();
 / *
    * Loop through the enumerator.
    */
 while( pConfigEnum->Next( 1, (void**)&pConfig,
         &pcFetched ) == SOPHOS_S_OK ){
 / *
    * Output some information about this
    * configuration object.
    * /
 / *
    * Release this configuration interface.
    * /
 pConfig->Release();
}
```

## 4.3 Using a configuration option

Once an interface to a particular configuration value has been obtained it can be used to retrieve various important pieces of information about it.

With this information, you can get and set its value, as shown in the next two sections. To get a configuration object's name, use the Config interface and call GetName(). The type is retrieved in a similar way.

```
LPOLESTR name     =  NULL;
U32     nameLength =  0;
U32     arraySize =  0;
HRESULT  hr;
 / *
    * First get the size of the array that holds the name.
    * /
 hr = pConfig->GetName(0, NULL, &nameLength);
 if( SOPHOS_SUCCEEDED (hr) ){
 / *
   * Set aside memory for the name.
   */
  name = new OLECHAR[nameLength];
  arraySize = nameLength;
  nameLength = 0;
 / *
   * Call again to retrieve the name.
   */
 hr = pConfig->GetName( arraySize, name,
                        &nameLength );
   if( SOPHOS_SUCCEEDED (hr) ){
    / *
      * Now get the type of the config.
      * /
    U32 type;
    hr = pConfig->GetType(&type);
    if( SOPHOS_SUCCEEDED (hr) ){
     / *
       * Do what you want with the name and type.
       * /
     }
   }
   / *
     * Free up the allocated array.
     * /
   delete [] name;
}
```

## 4.4 Getting a configuration value

Using the configuration name and type, you can determine the current state of that setting. In this example GetConfigValue() is used twice; once to give the size of the value, and once to retrieve the value.

```
U32     size      = 0;
U32     arraySize = size;
LPOLESTR value    = NULL;
/ *
  * First call to get the buffer size required to hold
  * the data.
  * /
 hr = pSAVI->GetConfigValue( name,
                             type,
                             0,
                             NULL,
                             &size );
if( SOPHOS_SUCCEEDED (hr) ){
 / *
    * Allocate space for the value.
    * /
 arraySize = size;
 value = new OLECHAR[arraySize];
 size = 0;
 / *
    * Now call to retrieve the data.
    * /
 hr = pSAVI->GetConfigValue(  name,
                              type,
                              arraySize,
                              value,
                              &size );
 if( SOPHOS_SUCCEEDED (hr) ){
   / *
     * Do what you want with the value.
     * /
 }
 / *
    * Tidy up allocated memory
    * /
 delete [] value;
}
```

## 4.5 Setting a configuration value

For example, to switch on full scanning do the following.

```
hr = pSAVI->SetConfigValue(  SOPHOS_DO_FULL_SWEEP,
                               SOPHOS_TYPE_U32,
                               SOPHOS_COMSTR ("1") );
```

## 4.6 Default configurations

The first time a program uses the SAVI interface it begins with a series of default configuration values.

The default values are listed in the *SAV Interface Developer Toolkit supplement*.

# 5 SAVI callbacks

This section describes the use of callbacks to monitor threat scanning.

## 5.1 Using callbacks

SAVI client applications can monitor the progress of scanning within large archives. To do this, the application must register a callback object with SAVI. The callback can be used to gain information about objects scanned, threats found and errors as they occur. This enables the client application to control the flow of execution as Sophos Anti-Virus runs.

## 5.2 Creating callback objects

It is up to the SAVI clients to implement notification callback objects. These must support functions such as

- OnVirusFound( ... )
- OnErrorFound( ... )
- OnFileFound( ... )

as well as the standard COM interface members

- QueryInterface()
- AddRef()
- Release().

For further information see the SaviDemo application and descriptions of ISweepNotify (page 99) and ISweepNotify2 (page 104).

## 5.3 Registering callbacks

Once the notification object has been created the client application must register it with SAVI.

For example, using C++ syntax

```
CISweepNotify* pNotify = NULL;
HRESULT        hr;
/ *
  * Attempt to create a new notification interface.
  * /
 pNotify = new CISweepNotify();
 if( pNotify ){
 / *
    * Register the notification interface with SAVI.
    * /
```

```
 hr = pSAVI->RegisterNotification(
                SOPHOS_IID_SWEEPNOTIFY,
                pNotify,
                NULL );
 / *
    * Find out if SAVI accepted the notification
    * interface.
    * /
 if( SOPHOS_SUCCEEDED (hr) ){
   / *
     * Callback in place.
     * /
 }
 else {
   / *
    * Tidy up callback object.
     * /
  delete pNotify;
 }
}
```

# 6 Using SAVI

This section describes

- how to obtain scanning engine version information (Obtaining version information (page 22))
- how to scan for threats (Scanning for threats (page 22))
- how to handle the results of a scan (Handling the results of a scan (page 25)).

It assumes that you have already initialised SAVI (Initializing SAVI (page 14)).

**Note:** The examples in this chapter are in C++ syntax and assume the code is compiled for Unicode on a Win32 platform. For the differences between C++ and C syntax, see Using C++ syntax or C syntax (page 8).

## 6.1 Obtaining version information

You can use the SAVI interface to retrieve information about the scanning engine by calling GetVirusEngineVersion(). This will return the version number and date of the scanning engine, together with the number of threats that can be detected and an enumerator of the additional threat identity files that are in use.

**Note:** For more information about obtaining version information, see GetVirusEngineVersion (page 42) and the SAVI Developer Toolkit code samples.

## 6.2 Scanning for threats

This section describes how to use SAVI to scan different types of data for threats. For specific information about the functions mentioned in this section, see the reference section of this User Manual.

It describes

- scanning files
- scanning disk sectors
- scanning memory
- advanced scanning methods.

### 6.2.1 Scanning files

SAVI can scan files directly via a filename or alternatively via an open file handle.

**Available functions**

SweepFile() - ISavi2 and ISavi3

SweepHandle() - ISavi3

**Scanning a file via its filename**

To scan a file via its filename, pass the filename to SweepFile(). Filenames can be absolute or relative to the current directory.

```
IEnumSweepResults* pEnumResults;
HRESULT hr;
hr = pSAVI->SweepFile( SOPHOS_COMSTR("c:\\eg.exe"),
                       SOPHOS_IID_ENUM_SWEEPRESULTS,
                       (void**)&pEnumResults );
```

**Scanning a file via an open file handle**

To scan a file via an open file handle, use SweepHandle(). Use SOPHOS_FD (see savitype.h) for the file handle, which must be opened in an appropriate manner for the platform. Once opened it can be passed directly to SweepHandle().

```
IEnumSweepResults* pEnumResults;
HRESULT hr;
SOPHOS_FD fileHandle = SOPHOS_FD_NULL;
/ *
  * Open file handle here with appropriate system call
  * and then pass it to SweepHandle. e.g. for windows
  * platforms
  * /
fileHandle = CreateFileW( L"c:\\example.exe",
                    GENERIC_READ|GENERIC_WRITE,
                    FILE_SHARE_READ|FILE_SHARE_WRITE,
                    NULL, OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL, NULL );
If (fileHandle != SOPHOS_FD_NULL){
 hr = pSAVI->SweepHandle( L"handle name string",
                       fileHandle,
                       SOPHOS_IID_ENUM_SWEEPRESULTS,
                       (void**)&pEnumResults );
```

### 6.2.2 Scanning disk sectors

SAVI can scan disk sectors via physical disk, cylinder and head numbers, or alternatively via a logical sector number.

**Available functions**

SweepPhysicalSector() - ISavi2 and ISavi3

SweepLogicalSector() - ISavi2 and ISavi3

**Note:** These functions are not available for all platforms. Calls to unsupported functions will return SOPHOS_SAVI_ERROR_NOT_SUPPORTED.

**Scanning a physical disk sector**

To scan a physical disk sector, use SweepPhysicalSector(). The following example scans the boot sector of the first physical hard drive.

```
IEnumSweepResults* pEnumResults;
HRESULT hr;
hr = pSAVI->SweepPhysicalSector(
                        L"\\\\.\\PHYSICALDRIVE0",
                        0, 0, 0,
                        SOPHOS_IID_ENUM_SWEEPRESULTS,
                        (void**)&pEnumResults
);
```

**Scanning a logical disk sector**

To scan a logical disk sector, use SweepLogicalSector(). The following example scans the boot sector of drive A, usually the floppy drive.

```
IEnumSweepResults* pEnumResults;
HRESULT hr;
hr = pSAVI->SweepLogicalSector(
                       L"\\\\.\\a:",
                       0, 0,
                       SOPHOS_IID_ENUM_SWEEPRESULTS,
                       (void**)&pEnumResults
);
```

### 6.2.3 Scanning memory

**Available functions**

SweepMemory() - ISavi2 and ISavi3

SweepBuffer() - ISavi3

**Scanning memory**

SweepMemory() enables SAVI to scan memory blocks occupied by running processes.

**Note:** This function is not available for all platforms. Calls to unsupported functions will return SOPHOS_SAVI_ERROR_NOT_SUPPORTED.

```
IEnumSweepResults* pEnumResults = NULL;
HRESULT hr;
hr = pSAVI->SweepMemory( SOPHOS_IID_ENUM_SWEEPRESULTS,
                         (void**)&pEnumResults );
```

**Scanning a buffer**

To scan data that has first been loaded into memory (e.g. a memory-mapped file or disk sector), use SweepBuffer().

```
IEnumSweepResults* pEnumResults = NULL;
HRESULT hr;
hr = pSAVI->SweepBuffer( L"buffer name string",
                         buffLen,
```

```
                        pBuffer,
                        SOPHOS_IID_ENUM_SWEEPRESULTS,
                        (void**)&pEnumResults );
```

### 6.2.4 Advanced scanning

SAVI also provides a way to sweep data in more complex forms. This can be used when the nature of data to be scanned makes the other scanning options unsuitable (e.g. when reading data from a network socket interface).

**Available functions**

SweepStream() - ISavi3

To exploit this functionality, implement an object that offers the ISaviStream interface (see ISweepDiskChange (page 108)) and pass an interface pointer into the SweepStream() method.

```
IEnumSweepResults* pEnumResults = NULL;
HRESULT hr;
ISaviStream* pStream = new myStreamObject(…);
hr = pSAVI->SweepStream( L"stream name string",
                        SOPHOS_IID_SAVISTREAM,
                        (void*)pStream,
                        SOPHOS_IID_ENUM_SWEEPRESULTS,
                        (void**)&pEnumResults );
```

## 6.3 Handling the results of a scan

### 6.3.1 Testing return codes

Scanning an item returns an HRESULT which should be tested to determine the result of a scan. The client can either use the pre-determined macros SOPHOS_SUCCEEDED() and SOPHOS_FAILED(), or test the return value explicitly as follows.

```
switch( hr ){
case SOPHOS_S_OK:
 / *
   * Scan success - no threats encountered.
   * /
 break;
case SOPHOS_SAVI_INFO_THREATPRESENT:
 / *
   * See below for ways to extract more information
   * about the threat or threats discovered.
   * /
 break;
default:
 / *
   * Some other error.
   * /
```

```
 break;
}
```

### 6.3.2 Obtaining threat information

Calls to scan or disinfect objects take a pointer to an IEnumSweepResults enumerator interface. When the function returns you can iterate through this IEnumSweepResults enumeration to retrieve information about any threats found.

First ensure that processing starts at the begining of the enumeration. To do this call the Reset() function of the IEnumSweepResults interface returned by the scan.

```
HRESULT hr = pEnumResults->Reset();
```

Once the enumerator has been reset you can begin to iterate. To iterate over the set, use the Next() function of the IEnumSweepResults interface. This returns a new interface to an individual SweepResults object which you can query for further information.

```
ISweepResults* pResults;
while (pEnumResults->Next( 1, (void**)&pResults, NULL ) ==
SOPHOS_S_OK){
 / *
   * Put code relating to specific result here.
   * /
}
```

Once a specific SweepResults interface has been retrieved, use the ISweepResults functions to get further information. For example, you can obtain the name of the threats, as well as whether it can be disinfected.

```
U32      BufLen = 0;
U32      Len    = 0;
LPOLESTR pBuf   = NULL;
/*
 * First find the length of the buffer required.
 */
pResults->GetVirusName( BufLen, pBuf, &Len );
/*
 * Allocate memory.
 */
BufLen = Len;
pBuf = new OLECHAR[Len];
Len = 0;
/*
 * Get the name of the threat.
 */
pResults->GetVirusName( BufLen, pBuf, &Len );
wprintf( L"Virus name: %s\n", pBuf );
/*
 * Free memory.
 */
delete [] pBuf;
pBuf = NULL;
/*
```

```
 * Now tell user if it can be disinfected.
 */
S32 IsDisinfectable;
pResults->IsDisinfectable( &IsDisinfectable );
wprintf ( L"Is threat disinfectable?: %s\n",
        ( isDisinfectable? L"Yes": L"No") );
```

After processing, each SweepResult object must be released.

```
/*
 * Release this results interface ready for the next
 * time around.
 */
pResults->Release();
pResults = NULL;
```

Likewise the IEnumSweepResults interface must be released once it has been finished with. This is always the case, even if it has not been used to obtain threat information.

```
pEnumResults->Release();
pEnumResults = NULL;
```

### 6.3.3 Disinfection

If a threat has been detected, there are two ways to approach disinfection.

Individual SweepResults objects describing a specific disinfectable threat can be passed into Disinfect() to attempt to remove the instance of the threat.

```
/*
 * Perform a sweep and iterate over the results
 * object as shown in the previous section.
 */
/*
 * Check to see if each object is disinfectable.
 */
U32 isDisinfectable = 0;
pResults->IsDisinfectable( &isDisinfectable );
if( isDisinfectable ){
 / *
   * Pass the object back to the SAVI interface for
   * disinfection.
   * /
 HRESULT hr = pSAVI->Disinfect(
                    SOPHOS_IID_SWEEPRESULTS,
                    pResults );
 if( SOPHOS_SUCCEEDED (hr) ){
  / *
    * Check the results object to ensure disinfection
    * was complete and repeat process if required.
    * /
 }
}
```

```
/ *
  * Once all results objects are finished with,
  * release the pointer to the results interface.
  * /
```

**Note:** Do not assume that because Disinfect() succeeded the object is completely free of threats. Disinfect() only attempts to disinfect a specific instance of the threat found. Multiple infections within objects are not dealt with. If a SAVI client calls Disinfect() the client must re-scan the object to ensure it is free of threats (see Disinfect (page 53)).

As an alternative to disinfecting specific threats, each of the Sweep...() functions in the SAVI interface has a corresponding Disinfect....() function (the exception is SweepMemory() for which disinfection is not offered due to the risks and complications of attempting disinfection of code segment memory). These methods can be used to attempt to remove all disinfectable threats from the object.

```
IEnumSweepResults* pEnumResults;
IEnumSweepResults* pEnumDisinfResults;
HRESULT hr;
hr = pSAVI->SweepFile( L"c:\\example.exe",
                       SOPHOS_IID_ENUM_SWEEPRESULTS,
                       (void**)&pEnumResults );
if (hr == SOPHOS_SAVI_INFO_THREATPRESENT){
 / *
   * Enumerate through results to extract threats
   * details and check any threats found are
   * disinfectable (see note below).
   */
  hr = pSAVI->DisinfectFile(L"c:\\example.exe",
                       SOPHOS_IID_ENUM_SWEEPRESULTS,
                       (void**)& pEnumDisinfResults);
  if (hr == SOPHOS_SAVI_INFO_THREATPRESENT){
    /*
     * Disinfection unavailable: file still infected.
     */
  }
  else if ( SOPHOS_SUCCEEDED (hr) ){
    /*
     * Disinfection was successful.
     */
  }
  else {
    /*
     * Error encountered during file disinfection.
     */
  }
  pEnumDisinfResults->Release();
  pEnumDisinfResults = NULL;
}
pEnumResults->Release();
pEnumResults = NULL;
```

**Note:** It is still advisable to use the results object to obtain threat information as previously described, and confirm that any threats found are disinfectable.

# 7 Terminating the interface

The following code shows how to unload SAVI when the program has finished with it. The version using COM must also uninitialize the COM library when it completes, while the non-COM version, which dynamically loaded the SAVI DLL (see Initializing SAVI (page 14)), may need to free it.

The code given here is in C++ syntax.

```
/ *
  * Finish with the SAVI interface and release the
  * reference to it
  * /
pSAVI->Terminate();
pSAVI->Release();
/ *
  * Finish with COM:
  * /
CoUninitialize();
```

# 8 Updating SAVI

You must regularly update SAVI to ensure it can detect all the latest threats.

Sophos provides the following updates.

- Full updates of the scanning engine.
- Threat identity files (IDEs), issued whenever a new threat is discovered.

If you are using a version of SAVI that does not support the ISavi3 interface, you must terminate and restart any running SAVI interface objects in order for the updated threat data to be read. Versions of SAVI that support ISavi3 can use LoadVirusData() to discard the old threat data definitions and re-read updated information.

Note that Win32 versions of SAVI implement the Sophos Scanning Engine as a separate library which can be updated independently. SAVI clients will automatically back off during updating to enable the scanning engine to be replaced. There is normally no need to stop SAVI clients for the update to succeed. Once the Sophos Anti-Virus update has completed successfully, SAVI clients will begin to service requests using the new scanning engine.

Following such an update of the scanning engine (on Windows only), calls to SAVI interface functions may return SOPHOS_SAVI_ERROR_MUST_REINIT. This happens when the new version of the Engine is incompatible with the currently loaded version of the SAVI library. In this situation it is necessary for the old SAVI library to be explicitly unloaded and the new one loaded in its place. There is sample source code in the SAVI Developer Toolkit ‘C++ Demo’ application which illustrates how this may be done.

# 9 SAVI Interfaces

SAVI consists of the set of interfaces declared within isavi3.h (C++) or csavi3c.h (C). These interfaces provide access to various objects used internally by SAVI. The header files isavi2.h and csavi2c.h are also provided for backward compatibility.

The SAVI interfaces are as follows:

| Interface name | Interface identifier | Description |
|---|---|---|
| ISavi2 | SOPHOS_IID_SAVI2 | The basic interface for programming scans and disinfections. |
| ISavi3 | SOPHOS_IID_SAVI3 | An extension of ISavi2 offering a wider range of scan and disinfect functions. |
| IIDEDetails | SOPHOS_IID_IDEDETAILS | The interface which describes an individual threat identity file (IDE). |
| IEnumIDEDetails | SOPHOS_IID_ENUM_ IDEDETAILS | Used to enumerate a list of IDEs. |
| ISweepResults | SOPHOS_IID_ SWEEPRESULTS | The results interface that describes an individual threat found by a scan. |
| ISweepError | SOPHOS_IID_ SWEEPERROR | The error interface that describes an error encountered when attempting to scan a file. |
| IEnumSweepResults | SOPHOS_IID_ ENUM_SWEEPRESULTS | Used to enumerate a list of scan results. |
| IEngineConfig | SOPHOS_IID_ ENGINECONFIG | The interface that describes an individual SAVI configuration option. |
| IEnumEngineConfig | SOPHOS_IID_ ENUM_ENGINECONFIG | Used to enumerate a list of SAVI configuration options. |
| IVersionChecksum | SOPHOS_IID_CHECKSUM | The interface which describes the checksum of one of the SAVI components. |

| Interface name | Interface identifier | Description |
|---|---|---|
| IEnumVersionChecksum | SOPHOS_IID_ENUM_ CHECKSUM | This is used to enumerate a list of IVersionChecksum objects. |
| IChangeNotify | SOPHOS_IID_ CHANGENOTIFY | This interface may be implemented by SAVI clients wishing to receive notification of changes to SAVI components. |
| ISeverityNotify | SOPHOS_IID_ SEVERITYNOTIFY | This interface may be implemented by SAVI clients requiring more detailed error reporting. |
| ISweepNotify | SOPHOS_IID_ SWEEPNOTIFY | The basic notification interface that can be implemented by SAVI clients that wish to receive notification callbacks from SAVI. |
| ISweepNotify2 | SOPHOS_IID_ SWEEPNOTIFY2 | An extension of ISweepNotify enabling greater control of the scanning process. |
| ISweepDiskChange | SOPHOS_IID_ DISKCHANGE | This interface must be implemented by SAVI clients that wish to receive notifications when SAV requires part of the virus data. |
| ISaviStream | SOPHOS_IID_SAVISTREAM | This interface must be implemented by SAVI clients using SweepStream() or DisinfectStream() in ISavi3. |
| IClassFactory | SOPHOS_IID_ CLASSFACTORY2 | Used to create an instance of SAVI. |

# 10 Return values

All interface functions return values that indicate whether the function succeeded.

In this User Manual, the most common return values are listed with each function description. For a definition of any other return value, refer to the SAV Interface Developer Toolkit supplement, which is included in the SAVI Developer Toolkit and is also available on the OEM Integration Resources webpage of the Sophos website.

# 11 SAVI configuration options

SAVI configuration options configure SAVI and the underlying threat scanning engine. They have unique name strings, and are passed to the APIs as LPCOLESTR. This data type maps to a pointer to either a Unicode/wide character string or an ASCII string, depending on the platform. Symbols defining some of the option names are published in savitype.h (under "Configuration option names"). However, this list is not comprehensive.

For a complete list of the configuration options supported by an installed version of SAVI, clients can call the ISavi2 or ISavi3 function GetConfigEnumerator() (GetConfigEnumerator (page 57)) . The returned interface can be used to list the names and types of all available configuration options.

The list of configuration options grows as more features are added and new file types are supported. They are published in the *SAV Interface Developer Toolkit supplement*, which is included in the SAVI Developer Toolkit and is also available on the OEM Integration Resources webpage of the Sophos website.

To return configuration options to their default values, use the SetConfigDefaults() function (SetConfigDefaults (page 54)) of the ISavi2 or ISavi3 interfaces.

To change or read the values of individual configuration options use the SetConfigValue() (SetConfigValue (page 55)) and GetConfigValue() (GetConfigValue (page 56)) functions of the ISavi2 or ISavi3 interfaces.

From SAVI3 onwards, group configuration options are also available. They enable the SAVI client to set the value of a group of related options in a single operation. Group configuration options are explained further in the *SAV Interface Developer Toolkit supplement*.

# 12 SAVI server entry point

There is one entry point to the SAVI server, named DllGetClassObject(), whose prototype follows. SAVI clients not using COM use this function to retrieve an instance of the SAVI class factory.

```
HRESULT   DllGetClassObject(
  REFCLSID  rclsid,
  REFIID  riid,
  LPVOID*  ppv
);
```

**Parameters**

| | |
|---|---|
| rclsid | A constant that identifies the type of class factory to create. This must be SOPHOS_CLASSID_SAVI. |
| riid | A constant that identifies the type of interface to be requested from the object created. This must be SOPHOS_IID_CLASSFACTORY2. |
| ppv | A pointer to a location to which DllGetClassObject() copies a pointer to the requested interface. If DllGetClassObject() fails, NULL is copied. |

## Return Values

If the object was created and the interface is supported, the return value is SOPHOS_S_OK.

## Remarks

DllGetClassObject() adds a reference to the interface returned through the ppv parameter. The client must be sure to release that reference when it has finished with the interface, by calling the interface's Release() function.

If you are using SAVI on a Windows platform, a simpler method of obtaining a SAVI interface pointer is by calling the Windows CoCreateInstance API, thus:

```
ISavi3*  pSAVI;HRESULT hr = CoCreateInstance( &SOPHOS_CLASSID_SAVI,
NULL, CLSCTX_ALL, &SOPHOS_IID_SAVI3, (void**)&pSAVI );
```

Although this is the recommended method of obtaining a SAVI interface on Windows platforms, you may call DllGetClassObject() directly. This method can be used to create truly cross-platform SAVI client source code.

# 13 IUnknown

The member functions of IUnknown in COM are common to all the SAVI interfaces listed in this document.

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. |
| AddRef | Add a reference to the interface, preventing its destruction. |
| Release | Release a reference. |

## 13.1 QueryInterface

C++ syntax:

```
HRESULT  SOPHOS_STDCALL QueryInterface(
  REFIID   IID,
  void**    ppObject
);
```

C syntax:

```
HRESULT  SOPHOS_STDCALL QueryInterface(
  void*    object,
  REFIID   IID,
  void**   ppObject
);
```

**Description**

Asks an object for a pointer to a specific interface. If the object supports the interface, QueryInterface() returns a pointer to the interface. If the object does not support the interface, QueryInterface() returns an error.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A poitner to the SAVI interface structure. |
| IID | One of the predefined interface identifiers contained in swiid.h. |

| | |
|---|---|
| ppObject | A pointer to a location to which QueryInterface() will copy a pointer to the interface requested. If QueryInterface() fails, NULL will be copied. |

### Return values

If the interface is supported, the return value is SOPHOS_S_OK. If the interface is not supported, the return value is SOPHOS_E_NOINTERFACE.

### Remarks

None.

## 13.2 AddRef

C++ syntax:

```
SOPHOS_ULONG SOPHOS_STDCALL AddRef();
```

C syntax:

```
SOPHOS_ULONG SOPHOS_STDCALL AddRef( void* object );
```

**Description**

Called by a client of an interface to add a reference to the underlying object. The underlying object will continue to exist as long as its reference count is greater than zero.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |

### Return values

If AddRef() succeeds, the return value is an integer in the range 1 to n, the value of the new reference count. Note that in multithreading situations this return value is not reliable and should therefore not generally be used.

### Remarks

A client may add a reference to an object to extend its life for as long as the object is required by a client. SAVI internally calls the AddRef() function before returning any interface pointers requested by the client, so generally clients do not need to call AddRef().

Every client call to AddRef() must be balanced by a subsequent call to Release().

## 13.3 Release

C++ syntax:

```
SOPHOS_ULONG SOPHOS_STDCALL Release();
```

C syntax:

```
SOPHOS_ULONG SOPHOS_STDCALL Release( void* object );
```

**Description**

Called by a client of an interface to release a reference to the underlying object. The underlying object will continue to exist as long as its reference count is greater than zero.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |

### Return values

If Release() succeeds, the return value is an integer in the range 0 to n, the value of the new reference count. Note that in multithreading situations this return value is not reliable and should therefore not generally be used.

### Remarks

A SAVI client makes a call to Release() when it has finished using an interface or an object. It must make a call to Release() to balance every call to AddRef(). SAVI internally calls AddRef() on every interface returned to client code, which is then responsible for calling Release(). An interface must not be used after Release() has been called. If the client fails to call Release(), the system is unable to recover memory used by the object.

### Example

```
IEnumSweepResults* pEnumResults;

HRESULT hr = pSAVI->SweepFile(L"a:\\v.com",
SOPHOS_IID_ENUM_SWEEPRESULTS, (void **) &pEnumResults);
if (SUCCEEDED(hr))
{  / *
     * Use pEnumResults here.
     * /
    pEnumResults->Release();
   / *
     * Finish with the list of infections.
     * /
}
```

# 14 ISavi2

ISavi2 is an interface to the main SAVI object used for performing threat scans and attempting disinfections. This object is created using an operating system-specific method (see SAVI server entry point (page 35)). More than one SAVI object may be created, but each instance must be initialised by InitialiseWithMoniker(), using a different moniker.

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| Initialise | Initialize the SAVI object. |
| InitialiseWithMoniker | Initialize the SAVI object with a client name. |
| Terminate | Finish with the SAVI object. |
| GetVirusEngineVersion | Retrieve information about the underlying virus scanner. |
| SweepFile | Scan a single file for viruses. |
| DisinfectFile | Try to completely disinfect a file. |
| SweepLogicalSector | Scan a logical disk sector for viruses. |
| SweepPhysicalSector | Scan a physical disk sector for viruses. |
| DisinfectLogicalSector | Try to completely disinfect a logical disk sector. |
| DisinfectPhysicalSector | Try to completely disinfect a physical disk sector. |
| SweepMemory | Scan memory for viruses. |
| Disinfect | Disinfect a file or sector from a specific virus infection. Use DisinfectFile() or DisinfectSector() if you require complete disinfection. |
| SetConfigDefaults | Reset all configuration options to their default values. |

| | |
|---|---|
| SetConfigValue | Set the value of a specific configuration option. |
| GetConfigValue | Get the value of a specific configuration option. |
| GetConfigEnumerator | Create an object that can be used to enumerate the names of configuration options. |
| RegisterNotification | A SAVI client may supply a notification interface to this function if it wishes to be notified of events during a scan. |

## 14.1 Initialise

C++ syntax:

```
HRESULT SOPHOS_STDCALL Initialise();
```

C syntax:

```
HRESULT SOPHOS_STDCALL Initialise( void* object );
```

**Description**

Initializes the SAVI interface ready for use. Either Initialise() or InitialiseWithMoniker() must be called before any other interface functions are called.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |

### Return values

If Initialise() succeeds, the return value is SOPHOS_S_OK or SOPHOS_SAVI_ERROR_OLD_VIRUS_DATA.

### Remarks

On Windows platforms it is strongly recommended that InitialiseWithMoniker() is called instead of Initialise(). The client must call Terminate() when it has finished using the interface.

## 14.2 InitialiseWithMoniker

C++ syntax:

```
HRESULT  SOPHOS_STDCALL InitialiseWithMoniker(LPCOLESTR
pApplicationMoniker
);
```

C syntax:

```
HRESULT  SOPHOS_STDCALL InitialiseWithMoniker(
  void*     object,
  LPCOLESTR pApplicationMoniker
);
```

**Description**

Initializes the SAVI interface ready for use. Either Initialise() or InitialiseWithMoniker() must be called before any other interface functions are called.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pApplicationMoniker | A short text string supplied by the client to uniquely identify this instance of the SAVI interface. In the case of Windows platforms, this text will appear in the Sophos Anti-Virus window. |

### Return values

If InitialiseWithMoniker() succeeds, the return value is SOPHOS_S_OK or SOPHOS_SAVI_ERROR_OLD_VIRUS_DATA.

### Remarks

The client must call Terminate() when it has finished using the interface.

On Windows the length of the moniker string is limited to 200 characters. If the passed string exceeds this length then it will be truncated. If a zero-length string is passed, then the function will return SOPHOS_E_INVALIDARG.

## 14.3 Terminate

C++ syntax:

```
HRESULT SOPHOS_STDCALL Terminate();
```

C syntax:

```
HRESULT SOPHOS_STDCALL Terminate( void* object );
```

**Description**

Terminates the SAVI interface after use.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |

### Return values

If Terminate() succeeded, the returned value is SOPHOS_S_OK.

### Remarks

None.

## 14.4 GetVirusEngineVersion

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetVirusEngineVersion(
 U32*        pVersion,
 LPOLESTR    pVersionString,
 U32         StringLength,
 SYSTEMTIME* pVdataDate,
 U32*        pNumberOfDetectableViruses,
 U32*        pVersionEx,
 REFIID      DetailsIID,
 void**      ppDetailsList
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetVirusEngineVersion(
 void*       object,
 U32*        pVersion,
 LPOLESTR    pVersionString,
 U32         StringLength,
 SYSTEMTIME* pVdataDate,
 U32*        pNumberOfDetectableViruses,
 U32*        pVersionEx,
 REFIID      DetailsIID,
 void**      ppDetailsList
);
```

**Description**

Gets information related to the scanning engine used for scanning.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pVersion | A pointer to a location for the scanning engine version number. This U32 is made up of 2 U16s in the form MajorVersion\|MinorVersion. To extract the relevant information use the following:<br>`U16 MajorVersion = *pVersion>>16;` |

| | |
|---|---|
| | `U16 MinorVersion = *pVersion&0x0000FFFF`<br>This can be NULL if not required. |
| pVersionString | A pointer to a buffer for a string containing the version number of the main threat data. This can be NULL if not required. |
| StringLength | The size of the buffer allocated for pVersionString in characters (not bytes). |
| pVdataDate | A pointer to a location for the date of the main threat data. This can be NULL if not required. |
| pNumberOfDetectableViruses | A pointer to a location for the number of detectable threats. This can be NULL if not required. |
| pVersionEx | A pointer to a location for extended scanning engine version information. This U32 is made up of two U16s in the same way as for pVersion above, except that the MSW is the 'patch' level and the LSW an (optional) revision. NB either of these words may be set to a value of 0xffff to indicate that this version number is undefined. Can bel NULL if not required. |
| DetailsIID | Identifies the type of details to be returned. This can be one of: SOPHOS_IID_ENUM_IDEDETAILS, SOPHOS_IID_ENUM_CHECKSUM or SOPHOS_IID_DATA_VERSION_NOLOAD. |
| ppDetailsLists | A pointer to a pointer to the interface object created by this function. The object can be used by the client to enumerate objects of the class appropriate to the DetailsIID parameter. If there are none, an enumerator containing zero items is returned. When the client has finished with the enumerator object, it must call Release() on the object. This parameter can be NULL if it is not required. It *must* be NULL if DetailsIID is SOPHOS_IID_DATA_VERSION_NOLOAD. |

## Return values

If GetVirusEngineVersion() succeeds the return value is SOPHOS_S_OK or SOPHOS_SAVI_ERROR_PARTIAL_INFORMATION.

## Remarks

If DetailsIID is SOPHOS_IID_ENUM_IDEDETAILS then information on loaded threat data files is returned in an IEnumIDEDetails interface.

If DetailsIID is SOPHOS_IID_ENUM_CHECKSUM then checksums of SAVI binary components are returned in an IEnumVersionChecksum interface.

If it is wished to obtain just scanning engine and threat data version information without the overhead of actually loading the threat data then pass DetailsIID as SOPHOS_IID_DATA_VERSION_NOLOAD and ppDetailsList and pNumberOfDetectableViruses as NULL. SAVI will return threat data and Engine version information via the pVersion, pVersionEx, pVersionString and pVdataDate parameters. NB this functionality is only applicable if called via ISavi3, as ISavi2 loads threat data during initialisation.

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

If the date of the main threat data is requested it is returned as a SYSTEMTIME structure. The date and time encoded in this structure is an absolute UTC time and is not adjusted for any local timezone.

## 14.5 SweepFile

C++ syntax:

```
HRESULT SOPHOS_STDCALL SweepFile(
 LPCOLESTR  pFileName,
 REFIID     ResultsIID,
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SweepFile(
 void*      object,
 LPCOLESTR  pFileName,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Scans a file for threats.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pFileName | The name fo the file to be scanned. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResuts | A pointer to a location to which SweepFile() will copy a pointer to the results interface requested. This parameter may be supplied as NULL if no results are required. |

### Return values

If SweepFile() succeeds and no threats are found, the return value is SOPHOS_S_OK. If a threat is found, the return value is SOPHOS_SAVI_INFO_THREATPRESENT.

Note that there are two success return values so it is a good idea to use the SOPHOS_SUCCEEDED() macro to check for success.

### Remarks

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

The client may use the results object to enumerate threats found in the file scanned. If no threats are found, the results object returned contains zero entries. The individual results objects can be passed directly to Disinfect().

If the file scanned is a compound file such as a Zip archive more than one threat may be found. In this case, the results object contains one entry for each threat found.

A confusing situation can occur when scanning compound files. If one part of a compound file contains a threat, and an error occurred scanning a different part of the compound file, SweepFile() returns SOPHOS_SAVI_INFO_THREATPRESENT. There is no indication that Sophos Anti-Virus failed to scan the whole compound file. Therefore, if a compound file contains a threat, treat the entire contents of the compound file as suspect.

On Windows versions of SAVI, scanning functions which take a file name string as a parameter now open the literal file name. This bypasses any automatic Windows file name modification (e.g. for file names ending with a space character).

## 14.6 DisinfectFile

C++ syntax:

```
HRESULT SOPHOS_STDCALL DisinfectFile(
 LPCOLESTR  pFileName,
 REFIID     ResultsIID,
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL DisinfectFile(
 void*      object,
 LPCOLESTR  pFileName,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Tries to disinfect a file completely, then scans it in the same way as SweepFile().

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pFileName | The name of the file to be disinfected. |
| ResultsIID | The type of results required. At present this parameter is SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to the location to which DisinfectFile() copies a pointer to the results interface requested. This parameter may be supplied as NULL if no results are required. |

### Return values

If DisinfectFile() succeeds the return value is SOPHOS_S_OK. If any threats are still present, the return value is SOPHOS_SAVI_INFO_THREATPRESENT.

### Remarks

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface. This function only returns SOPHOS_S_OK if the file is completely free of threats after disinfection.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

## 14.7 SweepLogicalSector

C++ syntax:

```
HRESULT SOPHOS_STDCALL SweepLogicalSector(
 LPCOLESTR  pDriveName,
 U32        Reserved,
 U32        SectorNumber,
 REFIID     ResultsIID,
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SweepLogicalSector(
 void*      object,
 LPCOLESTR  pDriveName,
```

```
 U32        Reserved,
 U32        SectorNumber,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Scans a logical sector on a disk.

**Note:** Logical sector scanning is not available on all platforms.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pDriveName | The name of the disk drive to scan, in the form \\.\A: where A is the drive letter. |
| Reserved | This value is reserved. It must be 0. |
| SectorNumber | The sector number you wish to scan (zero based). |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to a location to which SweepLogicalSector() will copy a pointer to the results interface requested. This parameter may be supplied as NULL if no results are required. |

## Return values

If SweepLogicalSector() succeeds and no threats are found, the return value is SOPHOS_S_OK. If a threat is found, the return value is SOPHOS_SAVI_INFO_THREATPRESENT.

Note that there are two success return values so it is a good idea to use SOPHOS_SUCCEEDED() to check for success.

## Remarks

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

The client may use the results object to enumerate threats found in the sector scanned. If no threats are found, the results object returned contains zero entries. Individual results objects can be passed directly to Disinfect().

## 14.8 SweepPhysicalSector

C++ syntax:

```
HRESULT SOPHOS_STDCALL SweepPhysicalSector(
 LPCOLESTR  pDriveName,
 U32        Head,
 U32        Cylinder,
 U32        Sector,
 REFIID     ResultsIID,
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SweepPhysicalSector(
 void*      object,
 LPCOLESTR  pDriveName,
 U32        Head,
 U32        Cylinder,
 U32        Sector,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Scans a physical sector on a disk.

**Note:** Physical sector scanning is not available on all platforms.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pDriveName | The name of the disk drive to scan. Two formats are supported: \\.\A: where A is the drive letter or \\.\PHYSICALDRIVEx, where x is the physical drive number (0 for the first fixed disk). |
| Head | The zero-based index to the physical disk head. |
| Cylinder | The zero-based index to the physical disk cylinder. |
| Sector | The one-based index to the physical disk sector. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |

| | |
|---|---|
| ppResults | A pointer to a location to which SweepPhysicalSector() copies a pointer to the results interface requested. If SweepPhysicalSector() fails, NULL is copied. This parameter may be supplied as NULL if no results are required. |

### Return values

If SweepPhysicalSector() succeeds and no threats are found, the return value is SOPHOS_S_OK. If a threat is found, the return value is SOPHOS_SAVI_INFO_THREATPRESENT.

Note that there are two success return values so it is a good idea to use SOPHOS_SUCCEEDED() to check for success.

### Remarks

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

The client may use the results object to enumerate threats found in the sector scanned. If no threats are found, the results object returned contains zero entries. The individual results objects can be passed directly to Disinfect().

## 14.9 DisinfectLogicalSector

C++ syntax:

```
HRESULT SOPHOS_STDCALL DisinfectLogicalSector(
 LPCOLESTR  pDriveName,
 U32        Reserved,
 U32        SectorNumber,
 REFIID     ResultsIID,
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL DisinfectLogicalSector(
 void*      object,
 LPCOLESTR  pDriveName,
 U32        Reserved,
 U32        SectorNumber,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Tries to completely disinfect a logical sector on a disk believed to be infected by a threat, then scan it in exactly the same way as SweepLogicalSector().

**Note:** Logical sector disinfection is not available on all platforms.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pDriveName | The name of the disk drive to scan, in the form \\.\A: where A is the drive letter. |
| Reserved | This value is reserved. It must be 0. |
| SectorNumber | The sector number you wish to scan (zero based). |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to a location to which DisinfectLogicalSector() copies a pointer to the results interface required. This parameter may be supplied as NULL if results are not required. |

### Return values

If DisinfectLogicalSector() succeeds the return value is SOPHOS_S_OK. Do not assume that DisinfectLogicalSector() succeeded, check the return value.

### Remarks

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

Generally it is easier to use Disinfect() than DisinfectLogicalSector() if SweepLogicalSector() has been called and a sweep results object is available.

## 14.10 DisinfectPhysicalSector

C++ syntax:

```
HRESULT SOPHOS_STDCALL DisinfectPhysicalSector(
 LPCOLESTR  pDriveName,
 U32        Head,
 U32        Cylinder,
```

```
 U32        Sector,
 REFIID     ResultsIID,
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL DisinfectPhysicalSector(
 void*      object,
 LPCOLESTR  pDriveName,
 U32        Head,
 U32        Cylinder,
 U32        Sector,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Tries to completely disinfect a physical sector on a disk believed to be infected by a threat, then scan it in exactly the same way as SweepPhysicalSector().

**Note:** Physical sector disinfection is not available on all platforms.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pDriveName | The name of the disk drive to scan. Two formats are supported: \\.\A: where A is the drive letter or \\.\PHYSICALDRIVEx, where x is the physical drive number (0 for the first fixed disk). |
| Head | The zero-based index to the physical disk head. |
| Cylinder | The zero-based index to the physical disk cylinder. |
| Sector | The one-based index to the physical disk sector. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to a location to which DisinfectPhysicalSector() copies a pointer to the results interface requested. This parameter may be supplied as NULL if results are not required. |

### Return values

If DisinfectPhysicalSector() succeeds the return value is SOPHOS_S_OK. Do not assume that DisinfectPhysicalSector() succeeded, check the return value.

### Remarks

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

Generally it is easier to use Disinfect() than DisinfectPhysicalSector() if SweepPhysicalSector() has been called and a sweep results object is available.

## 14.11 SweepMemory

C++ syntax:

```
HRESULT SOPHOS_STDCALL SweepMemory(
 REFIID    ResultsIID,
 void**    ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SweepMemory(
 void*     object,
 REFIID    ResultsIID,
 void**    ppResults
);
```

**Description**

Scans memory for threats. The memory blocks scanned are those containing the executable segments of all currently running processes. To scan a block of memory created or owned by the SAVI client use SweepBuffer() (see SweepBuffer (page 62)).

**Note:** Memory scanning is not available on all platforms.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to a location to which SweepMemory() copies a pointer to the results interface requested. |

| | |
|---|---|
| | This parameter may be supplied as NULL if no results are required. |

### Return values

SOPHOS_S_OK. If a threat is found, the return value is SOPHOS_SAVI_INFO_THREATPRESENT. Versions of SAVI in which this function is not supported return SOPHOS_SAVI_ERROR_NOT_SUPPORTED.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

Note that there are two success return values so it is a good idea to use SOPHOS_SUCCEEDED() to check for success.

### Remarks

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

The client may use the results object to enumerate any threats found. If no threats are found, the results object returned contains zero entries.

This function could take several minutes to complete, depending on factors such as the number of processes in memory.

## 14.12 Disinfect

C++ syntax:

```
HRESULT SOPHOS_STDCALL Disinfect(
 REFIID    ToDisinfectIID,
 void*     pToDisinfect
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL Disinfect(
 void*     object,
 REFIID    ToDisinfectIID,
 void*     pToDisinfect
);
```

**Description**

Tries to remove a specific threat from a specific item. The item and threat to disinfect are identified by the pToDisinfect object that was previously created by one of the Sweep...() functions.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| ToDisinfectIID | The type of results to disinfect. At present this parameter must be SOPHOS_IID_SWEEPRESULTS. |
| pToDisinfect | A pointer to the sweep results object that identifies the item to disinfect. |

### Return values

If Disinfect() succeeds the return value is SOPHOS_S_OK. Do not assume that Disinfect() succeeded (see below).

### Remarks

Disinfect() differs from other Disinfect...() functions in that after eliminating one threat, it does not perform another scan to check for further threats. Therefore, Disinfect() may return SOPHOS_S_OK after a successful disinfection when the object actually contains more threats.

- If complete disinfection of an object is required, call the relevant Disinfect...() function.
- If you use Disinfect(), perform another scan after a successful disinfection to ensure the object is completely fee of threats.

## 14.13 SetConfigDefaults

C++ syntax:

```
HRESULT SOPHOS_STDCALL SetConfigDefaults();
```

C syntax:

```
HRESULT SOPHOS_STDCALL SetConfigDefaults(
 void*     object
);
```

**Description**

Resets all configuration options to their default values. For a list of the defaults for all configuration options, see the SAV Interface Developer Toolkit supplement.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |

### Return values

If SetConfigDefaults() succeeds the return value is SOPHOS_S_OK.

### Remarks

This can cause a virus data reload if global virus data related config items have been changed from default values

## 14.14 SetConfigValue

C++ syntax:

```
HRESULT SOPHOS_STDCALL SetConfigValue(
 LPCOLESTR  pValueName,
 U32        Type,
 LPCOLESTR  pData
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SetConfigValue(
 void*      object,
 LPCOLESTR  pValueName,
 U32        Type,
 LPCOLESTR  pData
);
```

**Description**

Sets the value of a configuration option.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pValueName | The name of the configuration option the value of which is to be changed. |
| Type | The type of configuration option. |
| pData | The value to set, supplied as a text string. |

### Return values

If SetConfigValue() succeeds the return value is SOPHOS_S_OK.

### Remarks

The name and type of the configuration option must correspond to an item in the *SAV Interface Developer Toolkit supplement*, which is included in the SAVI Developer Toolkit and is also available on the OEM Integration Resources webpage of the Sophos website.

## 14.15 GetConfigValue

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetConfigValue(
 LPCOLESTR  pValueName,
 U32        Type,
 U32        MaxSize,
 LPOLESTR   pData,
 U32*       pSize
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetConfigValue(
 void*      object,
 LPCOLESTR  pValueName,
 U32        Type,
 U32        MaxSize,
 LPOLESTR   pData,
 U32*       pSize
);
```

**Description**

Reads the value of a configuration option. See Getting a configuration value (page 18).

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pValueName | The configuration option name being read. |
| Type | The type of configuration option. |
| MaxSize | The size of the buffer supplied in characters (not bytes). |
| pData | The value read, copied into this buffer as a text string. If you supply NULL for this parameter, the size of the required buffer is returned in pSize. |
| pSize | This value receives the size of the config value in characters (including terminator). |

### Return values

If GetConfigValue() succeeds the return value is SOPHOS_S_OK or SOPHOS_SAVI_INFO_OPT_GRP_INVAL_RTN. If the client supplied a buffer that was not big enough, SOPHOS_SAVI_ERROR_BUFFER_TOO_SMALL is returned.

### Remarks

The name and type of the configuration option must correspond to an item in the *SAV Interface Developer Toolkit supplement*, which is included in the SAVI Developer Toolkit and is also available on the OEM Integration Resources webpage of the Sophos website.

## 14.16 GetConfigEnumerator

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetConfigEnumerator(
 REFIID    ConfigIID,
 void**    ppConfigs
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetConfigEnumerator(
 void*     object,
 REFIID    ConfigIID,
 void**    ppConfigs
);
```

**Description**

Gets a pointer to an enumerator interface that can be used to list the configuration options supported by SAVI. See Retrieving configuration options (page 16).

Parameters

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| ConfigIID | The type of configuration object required. At present this parameter must be SOPHOS_IID_ENUM_ENGINECONFIG. |
| ppConfigs | A pointer to a location to which SAVI copies a pointer to the config enumerator interface requested. If GetConfigEnumerator() fails, NULL is copied. |

### Return values

If GetConfigEnumerator() succeeds the return value is SOPHOS_S_OK.

### Remarks

The configuration option enumerator can only be used to list the options available. To modify or read the values call the SetConfigValue() (page 55) and GetConfigValue() (page 56) functions in the ISavi2 interface.

## 14.17 RegisterNotification

C++ syntax:

```
HRESULT SOPHOS_STDCALL RegisterNotification(
 REFIID    NotifyIID,
 void*     pCallbackInterface,
 void*     Token
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL RegisterNotification(
 void*     object,
 REFIID    NotifyIID,
 void*     pCallbackInterface,
 void*     Token
);
```

**Description**

Supplies the SAVI interface with a notification interface that SAVI can use to call back when specific events occur during scanning or while reading the threat data.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the interface structure. |
| NotifyIID | The type of notification object supplied. At present this parameter must be SOPHOS_IID_SWEEPNOTIFY or SOPHOS_IID_SWEEPNOTIFY2 or SOPHOS_IID_SWEEPDISKCHANGE or SOPHOS_IID_CHANGENOTIFY or SOPHOS_IID_SEVERITYNOTIFY. |
| pCallbackInterface | A pointer to the callback interface supplied to SAVI. To unregister a previously registered sweep notification, supply NULL. |
| Token | An optional pointer to client data. This pointer is supplied to functions in the notification interface. It is not used internally by SAVI at all. If it is not used, supply NULL. |

### Return values

If RegisterNotification() succeeds the return value is SOPHOS_S_OK.

### Remarks

At present the only types of notification interface supported are ISweepNotify (page 99), ISweepNotify2 (page 104), ISweepDiskChange (page 108), IChangeNotify (page 119) and

ISeverityNotify (page 121). Attempts to register other types will fail. The SAVI interface immediately calls AddRef() on the interface supplied, and subsequently calls Release() when the interface is no longer required. The SAVI client must be prepared for the life of the notification object to be at least as long as the life of the ISavi2 interface with which it is registered.

**Note:** If a SOPHOS_IID_SWEEPDISKCHANGE notification is required, it must be registered with RegisterNotification() *before* threat data is loaded (see Threat data loading (page 12)).

# 15 ISavi3

ISavi3 is a development of the ISavi2 interface, described in ISavi2 (page 39). It combines the member functions of ISavi2 with some additional functions. General comments about ISavi2 also refer to ISavi3.

The member functions are listed in the following table. For a description of the duplicated member functions, see ISavi2 (page 39).

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| Initialise | Initialise the SAVI object. See Initialise (page 40). |
| InitialiseWithMoniker | Initialise the SAVI object with a client name. See InitialiseWithMoniker (page 40). |
| Terminate | Finish with the SAVI object. See Terminate (page 41). |
| GetVirusEngineVersion | Retrieve information about the underlying virus scanner. See GetVirusEngineVersion (page 42). |
| LoadVirusData | Load virus descriptions into the virus scanner. |
| SweepFile | Scan a single file for viruses. See SweepFile (page 44). |
| DisinfectFile | Try to completely disinfect a file. See DisinfectFile (page 45). |
| SweepLogicalSector | Scan a logical disk sector for viruses. See SweepLogicalSector (page 46). |
| SweepPhysicalSector | Scan a physical disk sector for viruses. See SweepPhysicalSector (page 48). |
| DisinfectLogicalSector | Try to completely disinfect a logical disk sector. See DisinfectLogicalSector (page 49). |
| DisinfectPhysicalSector | Try to completely disinfect a physical disk sector. See DisinfectPhysicalSector (page 50). |
| SweepMemory | Scan memory for viruses. See SweepMemory (page 52). |

| | |
|---|---|
| SweepBuffer | Scan a passed buffer for viruses. |
| SweepHandle | Scan a passed open file handle for viruses. |
| SweepStream | Scan for viruses using a passed stream interface object. |
| DisinfectBuffer | Try to completely disinfect a passed buffer. |
| DisinfectHandle | Try to completely disinfect a passed open file handle. |
| DisinfectStream | Try to completely disinfect a passed stream interface object. |
| Disinfect | Disinfect a file or sector from a specific virus infection. Use DisinfectFile() or DisinfectSector() if you require complete disinfection. See Disinfect (page 53). |
| SetConfigDetails | Reset all configuration options to their default values. See SetConfigDetails (page 54). |
| SetConfigValue | Set the value of a specific configuration option. See SetConfigValue (page 55). |
| GetConfigValue | Get the value of a specific configuration option. See GetConfigValue (page 56). |
| GetConfigEnumerator | Create an object that can be used to enumerate the names of configuration options. See GetConfigEnumerator (page 57). |
| RegisterNotification | A SAVI client may supply a notification interface to this function if it wishes to be notified of events during a scan. See RegisterNotification (page 58). |

## 15.1 LoadVirusData

C++ syntax:

```
HRESULT SOPHOS_STDCALL LoadVirusData();
```

C syntax:

```
HRESULT SOPHOS_STDCALL LoadVirusData( void* object );
```

**Description**

Causes SAVI to load (or reload) the underlying scanning engine with threat description data.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |

### Return values

If the threat data is located and loaded successfully, SOPHOS_S_OK is returned. If the threat descriptions are detected as being out of date, SOPHOS_SAVI_ERROR_OLD_VIRUS_DATA is returned.

### Remarks

If the scanner has already read threat descriptions, it drops them before re-reading commences. SAVI needs to have threat descriptions loaded before it can complete many of its member functions such as Sweep...(), Disinfect...() and GetThreatEngineVersion(). If one of these functions is called before threat descriptions have been loaded, SAVI makes an internal call to this function before proceeding.

This function may be called at any time, but can take a significant time to complete, during which any scanning activity by SAVI clients sharing the data is suspended. It is advisable to call this function only when threat data has changed, or as part of an initialisation sequence.

## 15.2 SweepBuffer

C++ syntax:

```
HRESULT SOPHOS_STDCALL SweepBuffer(
 LPCOLESTR  pBuffName,
 U32        buffSize,
 U08*       pBuff,
 REFIID     ResultsIID,
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SweepBuffer(
 void*      object,
 LPCOLESTR  pBuffName,
 U32        buffSize,
 U08*       pBuff,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Scans the passed memory buffer for threats.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |

| | |
|---|---|
| pBuffName | A name assigned by the SAVI client to the buffer. This is used for identification purposes and is not used internally by SAVI. |
| buffSize | The size of the buffer in bytes. |
| pBuff | A pointer to the buffer. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to the location where the function copies a pointer to the requested results interface. The parameter may be supplied as NULL if no results are required. |

### Return values

If no threats are found in the buffer, SOPHOS_S_OK is returned. If a threat is found SOPHOS_SAVI_INFO_THREATPRESENT is returned.

### Remarks

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

The client may use the results object to enumerate threats found in the buffer scanned. If no threats are found, the results object contains zero entries. The individual results object can be passed directly to Disinfect().

SweepBuffer() fails if the passed buffer is in an address space that is inaccessible from that occupied by SAVI.

## 15.3 SweepHandle

C++ syntax:

```
HRESULT SOPHOS_STDCALL SweepHandle(
 LPCOLESTR  pHandleName,
 SOPHOS_FD  fileHandle,
 REFIID     ResultsIID,
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SweepHandle(
 void**     object,
```

```
 LPCOLESTR  pHandleName,
 SOPHOS_FD  fileHandle,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Scans the file corresponding to the passed handle for threats.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pHandleName | A name assigned by the SAVI client to the file being scanned. This is used for identification purposes and is not used internally by SAVI. |
| fileHandle | The handle returned from opening the file. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to the location where the function copies a pointer to the requested results interface. The parameter may be supplied as NULL if no results are required. |

## Return values

If no threats are found in the buffer, SOPHOS_S_OK is returned. If a threat is found SOPHOS_SAVI_INFO_THREATPRESENT is returned.

## Remarks

The data type represented by SOPHOS_FD varies between platforms (see the header file savitype.h). It is essential that the file to be scanned is opened using a platform-specific function that returns a handle of this data type.

Files to be scanned must have the binary access flag set where appropriate to the platform. In addition, file handles to be disinfected must be created with write access.

SweepHandle() fails if the SAVI process does not have sufficient read/write access permissions on file handles created by the SAVI client.

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

The client may use the results object to enumerate threats found in the buffer scanned. If no threats are found, the results object contains zero entries. The individual results objects can be passed directly to Disinfect().

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

## 15.4 SweepStream

C++ syntax:

```
HRESULT SOPHOS_STDCALL SweepStream(
 LPCOLESTR   pStreamName,
 REFIID      StreamIID,
 void*       pStream,
 REFIID      ResultsIID,
 void**      ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SweepStream(
 void*       object,
 LPCOLESTR   pStreamName,
 REFIID      StreamIID,
 void*       pStream,
 REFIID      ResultsIID,
 void**      ppResults
);
```

**Description**

Scans for threats using the passed stream interface functions.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pStreamName | A name assigned by the SAVI client to the stream. This is used mainly for reporting purposes. |
| StreamIID | The type of stream interface implemented by the client. At present this must be either SOPHOS_IID_SAVISTREAM or SOPHOS_IID_SAVISTREAM2. |
| pStream | A pointer to the ISaviStream object. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to the location where the function copies a pointer to the requested results interface. The parameter may be supplied as NULL if no results are required. |

### Return values

If no threats are found in the stream, SOPHOS_S_OK is returned. If a threat is found SOPHOS_SAVI_INFO_THREATPRESENT is returned.

### Remarks

The passed ISaviStream pointer points to an interface structure that enables the threat scanner to make calls to client-supplied code in order to access the data to be scanned. This enables the client to implement more complex data access methods than allowed by the other Sweep...() methods. For information about the ISaviStream and ISaviStream2 interfaces, see ISaviStream (page 110) and ISaviStream2 (page 114).

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

The client may use the results object to enumerate threats found in the scanned stream. If no threats are found, the results object contains zero entries. The individual results objects can be passed directly to Disinfect().

If threats are discovered, SAVI calls AddRef() on the ISaviStream interface for each threat. This ensures the ISaviStream object persists in case Disinfect() is called on the individual results object. These references are released automatically when the client releases the results interface.

SweepStream() fails if the passed stream interface is in an address space that is inaccessible from the SAVI process.

## 15.5 DisinfectBuffer

C++ syntax:

```
HRESULT SOPHOS_STDCALL DisinfectBuffer(
 LPCOLESTR  pBuffName,
 U32        buffSize,
 U08*       pBuff,
 REFIID     ResultsIID,
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL DisinfectBuffer(
 void*      object
 LPCOLESTR  pBuffName,
 U32        buffSize,
 U08*       pBuff,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Tries to disinfect a file completely, and then scan it in the same way as SweepBuffer().

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pBuffName | A name assigned by the SAVI client to the buffer. This is used for identification purposes and is not used internally by SAVI. |
| buffSize | The size of the buffer in bytes. |
| pBuff | A pointer to the buffer. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to the location where the function copies a pointer to the requested results interface. The parameter may be supplied as NULL if no results are required. |

### Return values

If DisinfectBuffer() succeeds, SOPHOS_S_OK is returned. If any threats are still present, SOPHOS_SAVI_INFO_THREATPRESENT is returned.

### Remarks

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

This function only returns SOPHOS_S_OK if the buffer is completely free of threats after the disinfection attempt.

DisinfectBuffer() fails if the passed buffer is in an address space inaccessible from that occupied by SAVI.

## 15.6 DisinfectHandle

C++ syntax:

```
HRESULT SOPHOS_STDCALL DisinfectHandle(
 LPCOLESTR  pHandleName,
 SOPHOS_FD  fileHandle,
 REFIID     ResultsIID,
```

```
 void**     ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL DisinfectHandle(
 void**     object,
 LPCOLESTR  pHandleName,
 SOPHOS_FD  fileHandle,
 REFIID     ResultsIID,
 void**     ppResults
);
```

**Description**

Attempts complete disinfection of the file corresponding to the passed handle, and then scans it in exactly the same way as SweepHandle().

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pHandleName | A name assigned by the SAVI client to the file being scanned. This is used for file identification purposes and is not used internally by SAVI. |
| fileHandle | The handle returned from opening the file in read/write mode. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to the location where the function copies a pointer to the requested results interface. The parameter may be supplied as NULL if no results are required. |

## Return values

If DisinfectHandle() succeeds, SOPHOS_S_OK is returned. If any threats are still present, SOPHOS_SAV_INFO_THREATPRESENT is returned.

## Remarks

The data type represented by SOPHOS_FD varies between platforms (see the header file savitype.h). It is essential that the file to be scanned is opened using a platform-specific function that will return a handle of this data type.

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

Files to be disinfected must be opened in read/write mode, with the binary access flag set where appropriate to the platform. In addition, file handles to be disinfected must be created with write access.

DisinfectHandle() fails if the SAVI process does not have sufficient read/write access permissions on file handles created by the SAVI client.

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

This function only returns SOPHOS_S_OK if the file is completely free of threats after the disinfection attempt.

## 15.7 DisinfectStream

C++ syntax:

```
HRESULT SOPHOS_STDCALL DisinfectStream(
 LPCOLESTR   pStreamName,
 REFIID      StreamIID,
 void*       pStream,
 REFIID      ResultsIID,
 void**      ppResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL DisinfectStream(
 void*       object,
 LPCOLESTR   pStreamName,
 REFIID      StreamIID,
 void*       pStream,
 REFIID      ResultsIID,
 void**      ppResults
);
```

**Description**

Attempts complete disinfection of the object encapsulated by the passed ISaviStream interface, then scans it in exactly the same way as SweepStream().

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pStreamName | A name assigned by the SAVI client to the stream. This is used mainly for reporting purposes. |
| StreamIID | The type of stream interface implemented by the client. At present this must be either SOPHOS_IID_SAVISTREAM or SOPHOS_IID_SAVISTREAM2. |

| | |
|---|---|
| pStream | A pointer to the implemented stream interface. |
| ResultsIID | The type of results required. At present this parameter must be SOPHOS_IID_ENUM_SWEEPRESULTS. |
| ppResults | A pointer to the location where the function copies a pointer to the requested results interface. The parameter may be supplied as NULL if no results are required. |

## Return values

If DisinfectStream() succeeds, the return value is SOPHOS_S_OK. If any threats are still present, SOPHOS_SAVI_INFO_THREATPRESENT is returned.

## Remarks

The passed ISaviStream pointer points to an interface structure that enables the threat scanner to make calls to client-supplied code in order to access the data to be scanned. This enables the client to implement more complex data-access methods than allowed via other Sweep...() functions. For information about the ISaviStream and ISaviStream2 interfaces, see ISaviStream (page 110) and ISaviStream2 (page 114) .

A results interface may be returned even if the function returns a failure code. Always check the return value of ppResults because failure to release objects returned may cause memory leaks.

If a results interface is returned, SAVI internally calls AddRef() for the interface before it is returned to the client. Therefore the client must call Release() when it has finished using the interface.

The client may use the results object to enumerate threats found in the scanned stream. If no threats are found, the results object contains zero entries. The individual results objects can be passed directly to Disinfect().

If threats remain in the stream, SAVI calls AddRef() on the ISaviStream interface for each threat. This ensures the ISaviStream object persists in case Disinfect() is called on the individual results object. These references are released automatically when the client releases the results interface.

DisinfectStream() fails if the passed stream interface is in an address space that is inaccessible from the SAVI process.

# 16 Enumerator interfaces

The functions of an enumerator interface allow SAVI client code to work through a list of objects of a particular type, for example the descriptions of the IDE files currently loaded. There are a number of enumerator interfaces in SAVI. Each one has the same functions and parameters, but the type of object enumerated is different for each.

Enumerator interfaces may be returned by a number of different SAVI interface functions.

SAVI currently offers the following enumerator interfaces, which are returned by the functions listed.

| | |
|---|---|
| IEnumIDEDetails | Returns a list of IIDEDetails objects, each of which describes a virus data file used by the current SAVI object. Returned by GetVirusEngineVersion(). |
| IEnumSweepResults | Returns a list of ISweepResults. These describe the results obtained from scanning a file, disk sector, etc. Returned by SweepFile(), SweepLogicalSector(), etc. |
| IEnumEngineConfig | Returns a list of IEngineConfig objects, each of which describes one of the configuration settings available for a SAVI object. Returned by GetConfigEnumerator(). |
| IEnumVersionChecksum | Returns a list of IVersionChecksum objects, which describe the checksum of a SAVI component. Depending on platform, IVersionChecksum objects may be returned for the SAVI library code and/or the virus data. Returned by GetVirusEngineVersion(). |
| IEnumIdentityInfo | Returns a list of IIdentityInfo objects, each of which describes an identity in the virus database. Returned by the GetMatchingIdentities() and GetAllIdentities() methods of IQueryLoadedProtection. |

The number of functions of each enumerator interface are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| Next | Request the next item from the list. |
| Skip | Skip over the next items from the list. |

| | |
|---|---|
| Reset | Reset the enumerator to the start of the list. |
| Clone | Create a copy of the enumerator in its current state. |

## 16.1 Next

C++ syntax:

```
HRESULT  SOPHOS_STDCALL Next(
 SOPHOS_ULONG   cElement,
 void*          pElement[],
 SOPHOS_ULONG*  pcFetched
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL Next(
 void*          object,
 SOPHOS_ULONG   cElement,
 void*          pElement[],
 SOPHOS_ULONG*  pcfetched
);
```

**Description**

Copies the next elements from the enumerator into a buffer supplied by the client.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the enumerator interface structure. |
| cElement | The number of elements requested, typically one. |
| pElement | A pointer to a buffer allocated by the client that is big enough to accept cElement elements. |
| pcFetched | A pointer to a location that will receive the actual number of elements copied. NULL may be supplied if this value is not required. |

### Return values

If Next() succeeds, the return value is SOPHOS_S_OK. If Next() retrieves fewer elements than requested, it returns SOPHOS_S_FALSE.

### Remarks

Next calls AddRef() on each interface pointer that it returns. Therefore the client must call Release() when it has finished using each interface.

## 16.2 Skip

C++ syntax:

```
HRESULT SOPHOS_STDCALL Skip(
 SOPHOS_ULONG   cElement
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL Skip(
 void*          object,
 SOPHOS_ULONG   cElement
);
```

**Description**

Skips the next elements in the enumerator.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the enumerator interface structure. |
| cElement | The number of elements to skip. |

### Return values

If Skip() succeeds, the return value is SOPHOS_S_OK. If Skip() skips fewer elements than were requested, it returns SOPHOS_S_FALSE.

### Remarks

None.

## 16.3 Reset

C++ syntax:

```
HRESULT SOPHOS_STDCALL Reset();
```

C syntax:

```
HRESULT SOPHOS_STDCALL Reset( void* object );
```

**Description**

Resets the enumerator so that the next call to Next() retrieves the first item in the enumerator.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the enumerator interface structure. |

### Return values

If Reset() succeeds, the return value is SOPHOS_S_OK.

### Remarks

None.

## 16.4 Clone

C++ syntax:

```
HRESULT  SOPHOS_STDCALL Clone(
 void**   ppEnum
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL Clone(
 void*    object,
 void**   ppEnum
);
```

**Description**

Creates a copy of the enumerator in its current state.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the enumerator interface structure. |
| ppEnum | A pointer to a location to which Clone() copies a pointer to the newly created copy of the enumerator object. |

### Return values

If Clone() succeeds, the return value is SOPHOS_S_OK.

## Remarks

Clone() calls AddRef() on the new enumerator object before returning it to the client. Therefore the client must call Release() when it has finished using the object. The new enumerator is an exact copy of the enumerator that cloned it, including its internal state.

# 17 IIDEDetails

IIDEDetails is the interface to an object that represents an update file (IDE, UPD or VDL) in use by the underlying scanning engine. The interface can be used to retrieve information about the update file, such as its name. An enumerator of IIDEDetails is created by the GetThreatEngineVersion() function in the ISavi2 and ISavi3 interfaces (see GetVirusEngineVersion (page 42)).

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| GetName | Get the name of the update file. |
| GetType | Get the type of the update file. |
| GetState | Get the state of the update file. |
| GetDate | Get the date of the update file. |

## 17.1 GetName

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetName(
 U32       ArraySize,
 LPOLESTR  pIDEName,
 U32*      pIDENameLength
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetName(
 void*      object,
 U32        ArraySize,
 LPOLESTR   pIDEName,
 U32*       pIDENameLength
);
```

**Description**

Gets the name of the update file.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| ArraySize | The size of the buffer supplied by the client, in characters (not bytes). |
| pIDEName | A pointer to a buffer supplied by the client, to which is copied the name of the update file. This parameter may be supplied as NULL, in which case ArraySize is ignored. |
| pIDENameLength | The length of the update file name (including terminator) is copied to this location in characters (not bytes). This parameter may be supplied as NULL. |

### Return values

If GetName() succeeds, the return value is SOPHOS_S_OK. If the buffer supplied by the client is not big enough, SOPHOS_SAVI_ERROR_BUFFER_TOO_SMALL is returned.

### Remarks

If the size of the name string is required in advance then call the function with a NULL LPOLESTR and the function will write the required buffer size (characters including terminator) to the name length pointer.

## 17.2 GetType

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetType(
 U32*    pType
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetType(
 void*    object,
 U32*     pType
);
```

**Description**

Gets the type of the update file.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pType | Receives the type of the update file. |

### Return values

If GetType() succeeds, the return value is SOPHOS_S_OK.

### Remarks

The possible types of update file are as follows:

| | |
|---|---|
| SOPHOS_TYPE_IDE | The file is an IDE file. |
| SOPHOS_TYPE_UPD | The file is a UPD file. |
| SOPHOS_TYPE_VDL | The file is a VDL file. |
| SOPHOS_TYPE_MAIN_THREAT_DATA | The file is the main threat data file. |
| SOPHOS_TYPE_UNKNOWN | The file type is unknown. |

## 17.3 GetState

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetState(
 U32*    pState
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetState(
 void*    object,
 U32*     pState
);
```

**Description**

Gets the state of the update file.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |

| | |
|---|---|
| pState | Receives the state of the update file. |

### Return values

If GetState() succeeds the returned value is SOPHOS_S_OK.

### Remarks

The possible states of an update file are as follows:

| | |
|---|---|
| SOPHOS_IDE_VDL_SUCCESS | The update file is in use. |
| SOPHOS_IDE_VDL_FAILED | A problem occurred loading the update file - it is not in use. |
| SOPHOS_IDE_VDL_INVALID_VERSION | The data format is an unrecognized version - the file is not in use. |
| SOPHOS_IDE_VDL_OLD_WARNING | The update file is in use, but more than 90 days old. |

## 17.4 GetDate

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetDate(
 SYSTEMTIME* pDate
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetDate(
 void*        object,
 SYSTEMTIME*  pDate
);
```

**Description**

Gets the file system time stamp of the update file.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pDate | Receives the file system time stamp of the update file. |

### Return values

If GetDate() succeeds, the return value is SOPHOS_S_OK.

### Remarks

None.

# 18 ISweepResults

ISweepResults is the interface to an object that represents a threat found by a scanning function. The interface can be used to retrieve information about the threat, such as its name. An enumerator of ISweepResults is created by any scan or disinfection function in the ISavi2 or ISavi3 interfaces. An ISweepResults interface pointer is passed as a parameter to Disinfect() to identify the threat to be disinfected.

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| IsDisinfectable | Indicate whether the infected object can be disinfected. |
| GetVirusType | Indicate the type of the virus infection. |
| GetVirusName | Get the name of the virus. |
| GetLocationInformation | Get the location of the virus infection. |

## 18.1 IsDisinfectable

C++ syntax:

```
HRESULT SOPHOS_STDCALL IsDisinfectable(
 U32*     pIsDisinfectable
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL IsDisinfectable(
 void*    object,
 U32*     pIsDisinfectable
);
```

**Description**

Call this function to find out whether a threat found can be disinfected.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pIsDisinfectable | Receives the value of the IsDisinfectable() flag. This value is either SOPHOS_SAVI_NOT_DISINFECTABLE or SOPHOS_SAVI_DISINFECTABLE. |

### Return values

If IsDisinfectable() succeeds, the returned value is SOPHOS_S_OK.

### Remarks

None.

## 18.2 GetThreatType

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetThreatType(
 U32*     pThreatType
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetThreatType(
 void*    object,
 U32*     pThreatType
);
```

**Description**

Call this function to discover how the threat was identified.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pThreatType | Receives the type of threat. |

### Return values

If GetThreatType() succeeds, the return value is SOPHOS_S_OK.

### Remarks

The threat type is not generally of interest to a SAVI client. However, it takes the following values:

| | |
|---|---|
| SOPHOS_NO_THREAT | No threat found. |
| SOPHOS_THREAT | Threat found. |
| SOPHOS_THREAT_IDENTITY | Threat identity found. |
| SOPHOS_THREAT_PATTERN | Threat pattern found. |
| SOPHOS_THREAT_MACINTOSH | Macintosh threat found. |

## 18.3 GetThreatName

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetThreatName(
 U32        ArraySize,
 LPOLESTR   pName,
 U32*       pNameLength
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetThreatName(
 void*      object,
 U32        ArraySize,
 LPOLESTR   pName,
 U32*       pNameLength
);
```

**Description**

Gets the name of the threat.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| ArraySize | The size of the buffer supplied by the client, in characters (not bytes). |
| pName | A pointer to a buffer supplied by the client, to which is copied the name of the threat. This parameter may be supplied as NULL, in which case ArraySize is ignored. |

| | |
|---|---|
| pNameLength | The length of the threat name (including terminator) in characters (not bytes) is copied to this location. This parameter may be supplied as NULL. |

### Return values

If GetThreatName() succeeds, the return value is SOPHOS_S_OK. If the buffer supplied by the client is not big enough, SOPHOS_SAVI_ERROR_BUFFER_TOO_SMALL is returned.

### Remarks

None.

## 18.4 GetLocationInformation

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetLocationInformation(
 U32        ArraySize,
 LPOLESTR   pName,
 U32*       pNameLength
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetLocationInformation(
 void*      object,
 U32        ArraySize,
 LPOLESTR   pName,
 U32*       pNameLength
);
```

**Description**

Gets the location of the threat.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| ArraySize | The size of the buffer supplied by the client, in characters (not bytes). |
| pName | A pointer to a buffer supplied by the client, to which is copied the name of the location where the threat was found. This parameter may be supplied as NULL, in which case ArraySize is ignored. |

| | |
|---|---|
| pNameLength | The length of the threat name (including terminator) in characters (not bytes) is copied to this location. This parameter may be supplied as NULL. |

## Return values

If GetLocationInformation() succeeds, the return value is SOPHOS_S_OK. If the buffer supplied by the client is not big enough, SOPHOS_SAVI_ERROR_BUFFER_TOO_SMALL is returned.

## Remarks

None.

# 19 ISweepError

ISweepError is the interface to an object that holds information about an error. It can be used from within the SAVI OnErrorFound() callback function to retrieve detailed information about the error encountered.

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| GetLocationInformation | Get the location of the error. |
| GetErrorCode | Get the type of error that occurred. |

## 19.1 GetLocationInformation

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetLocationInformation(
 U32        ArraySize,
 LPOLESTR   pLocation,
 U32*       pLocationNameLength
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetLocationInformation(
 void*      object,
 U32        ArraySize,
 LPOLESTR   pLocation,
 U32*       pLocationNameLength
);
```

**Description**

Gets the location of the error.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| ArraySize | The total size of the buffer supplied to receive the name, in characters. |
| pLocation | A pointer to the buffer that will receive the location text. You may supply NULL for this parameter, for example, to discover the required buffer size. |
| pLocationNameLength | This location receives the total length of the error location (including terminator) in characters. |

### Return values

If GetLocationInformation() succeeds, the return value is SOPHOS_S_OK. If the client supplies a buffer that is not big enough, SOPHOS_SAVI_ERROR_BUFFER_TOO_SMALL is returned.

### Remarks

None.

## 19.2 GetErrorCode

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetErrorCode(
 HRESULT  *  ErrorCode
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetErrorCode(
 void*     object,
 HRESULT*  ErrorCode
);
```

**Description**

Gets the type of error that occurred.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| ErrorCode | This location receives the return value (error code). At present the following values are possible: |

| | |
|---|---|
| SOPHOS_SAVI_ERROR_FILE_ENCRYPTED | The file is encrypted. |
| SOPHOS_SAVI_ERROR_CORRUPT | The file is corrupt. |
| SOPHOS_SAVI_ERROR_NOT_SUPPORTED | The file format is not supported. |
| SOPHOS_SAVI_ERROR_COULD_NOT_OPEN | The file cannot be opened. |

**Note:** See the *SAV Interface Developer Toolkit supplement* for information about other error code values.

## Return values

If GetErrorCode() succeeds the return value is SOPHOS_S_OK. Otherwise, one of the return values listed above may be returned.

## Remarks

None.

## Example

```
HRESULT SOPHOS_STDCALL CSaviNotify::OnErrorFound(
                                    void*   Token,
                                    REFIID ErrorIID,
                                    void*  pError )
{
   try{
  HRESULT       hr          = SOPHOS_S_OK;
  HRESULT       errorCode   = SOPHOS_S_OK;
  ISweepError*  INTerror    = (ISweepError*)pError;
  OLECHAR*      location    = NULL;
  U32           locLength   = 0;
  if( (!pError) || (ErrorIID !=SOPHOS_IID_SWEEPERROR)) {
  return SOPHOS_E_INVALIDARG;
        }
  / *
    * Find out how long the location string is.
    * /
  hr = INTerror->GetLocationInformation( 0, NULL,
                  &locLength );
  if( FAILED(hr) ) {
    hr = SOPHOS_E_UNEXPECTED;
  }
  else {
    / *
      * Make memory to store the location string.
      * /
     location = new OLECHAR[locLength];
```

```
    / *
      * Now get the location string.
      * /
    hr = INTerror->GetLocationInformation( locLength,
          location, NULL );
    if( FAILED(hr) ) {
     hr = E_UNEXPECTED;
    }
    else {
     / *
       * Now get the error code.
       * /
     hr = INTerror->GetErrorCode( &errorCode );
     if( FAILED(hr) ) {
      hr = E_UNEXPECTED;
     }
     else {
      / *
        * Now output the information
        * /
      wprintf(L"——>Error 0x%08lx in %s\n",
              errorCode, location);
      hr = SOPHOS_SAVI_CBCK_CONTINUE_NEXT;
     }
    }
    delete [] location;
  }
  return hr;
  }
}
```

# 20 IEngineConfig

IEngineConfig is the interface to an object that represents a single configuration option of the underlying threat engine. The interface can be used to retrieve information about the option, such as its name.

An enumerator of IEngineConfig is created by GetConfigEnumerator() in the ISavi2 or ISavi3 interfaces. Although the IEngineConfig interface provides name and type of configuration options, it cannot be used to read or modify values. The ISavi2 and ISavi3 interfaces provide those functions.

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| GetName | Get the name of the configuration option. |
| GetType | Get the type of the configuration option. |

## 20.1 GetName

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetName(
 U32        ArraySize,
 LPOLESTR   pName,
 U32*       pNameLength
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetName(
 void*      object,
 U32        ArraySize,
 LPOLESTR   pName,
 U32*       pNameLength
);
```

**Description**

Gets the name of the configuration option.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| ArraySize | The size of the buffer supplied by the client, in characters (not bytes). |
| pName | A pointer to a buffer supplied by the client to which will be copied the name of the configuration option. This parameter may be supplied as NULL, in which case ArraySize is ignored. |
| pNameLength | The length of the configuration option name (including terminator) is copied to this location in characters (not bytes). This parameter may be supplied as NULL. |

### Return values

If GetName() succeeds, the return value is SOPHOS_S_OK. If the buffer supplied by the client is not big enough, SOPHOS_SAVI_ERROR_BUFFER_TOO_SMALL is returned.

### Remarks

If the size of the name string is required in advance then call the function with a NULL LPOLESTR and the function will write the required buffer size (characters, including terminator) to the name length pointer.

## 20.2 GetType

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetType(
 U32*     pType
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetType(
 void*    object,
 U32*     pType
);
```

**Description**

Gets the type of a particular configuration option, one of the SOPHOS_TYPE_... constants, listed in the SAV Interface Developer Toolkit supplement.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pType | Receives the type of the configuration option. |

## Return values

If GetType() succeeds, the returned value is SOPHOS_S_OK.

## Remarks

None.

# 21 IVersionChecksum

IVersionChecksum is the interface to an object that represents the checksum of one of the underlying components of the running SAVI implementation. This allows SAVI clients to check whether or not the fundamental scanning capability has been updated.

Depending on platform, IVersionChecksum objects may be returned for the SAVI library code and/or the threat data. The interface will return both the checksum and which the component the checksum relates to.

An enumerator of IVersionChecksum is created by passing SOPHOS_IID_ENUM_CHECKSUM as the DetailsIID parameter to the GetThreatEngineVersion() function in the ISavi2 and ISavi3 interfaces (see GetVirusEngineVersion (page 42)).

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| GetType | Get the type of the checksum. |
| Get Value | Get the value of the checksum. |

## 21.1 GetType

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetType (
  U32*      pType
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetType (
  void*     object,
  U32*      pType
);
```

**Description**

Gets the type of the checksum – i.e. what SAVI component is represented by the checksum. This may be either SOPHOS_TYPE_VDATA (threat data) or SOPHOS_TYPE_BINARY (SAVI library).

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pType | Receives the type of the checksum. |

### Return values

If GetType() succeeds, the returned value is SOPHOS_S_OK.

### Remarks

None.

## 21.2 GetValue

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetValue (
    U32*      pValue
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetValue (
  void*     object,
  U32*      pValue
);
```

**Description**

Gets the checksum value itself.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pValue | Receives the value of the checksum. |

### Return values

If GetValue() succeeds, the returned value is SOPHOS_S_OK.

## Remarks

None.

# 22 IClassFactory

ClassFactory is the interface used for creating instances of the SAVI object. An instance of IClassFactory is obtained by calling DllGetClassObject() (see SAVI server entry point (page 35)). See also the Win32 function CoGetClassObject().

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| CreateInstance | Creates an instance of a specified object. |
| LockServer | Locks the SAVI server in memory. |

## 22.1 CreateInstance

C++ syntax:

```
HRESULT SOPHOS_STDCALL CreateInstance(
 IUnknown*   pUnkOuter,
 REFIID      riid,
 void**      ppObject
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL CreateInstance(
 void*      object,
 IUnknown*  pUnkOuter,
 REFIID     riid,
 void**     ppObject
);
```

**Description**

Creates an instance of a SAVI object and requests it for an interface whose type is supplied in the riid parameter. At present the only type supported is SOPHOS_IID_SAVI.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| pUnkOuter | Not used, only present for compatibility with COM. This parameter must always be supplied as NULL. |
| riid | Identifiees the type of interface requested. |
| ppObject | A pointer to a location to which CreateInstance() will copy a pointer to the interface requested. If CreateInstance() fails, NULL will be copied. |

### Return values

If CreateInstance() succeeds, the return value is SOPHOS_S_OK.

### Remarks

None.

## 22.2 LockServer

C++ syntax:

```
HRESULT SOPHOS_STDCALL LockServer(
 SOPHOS_BOOL  Lock
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL LockServer(
 void*        object,
 SOPHOS_BOOL  Lock
);
```

**Description**

Locks the SAVI interface server in memory (Windows platforms only).

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| Lock | If this parameter is non-zero, the SAVI server is locked in memory. Otherwise, it is unlocked. |

## Return values

If LockServer() is supported on this platform and it succeeds, the return value is SOPHOS_S_OK.

## Remarks

On Windows platforms, this function can be used to lock the SAVI interface server in memory. This feature might be useful to keep the server loaded when no SAVI interfaces are currently in use, saving time the next time a SAVI interface is requested. Every call that locks the server must be balanced by a call that unlocks the server.

On non-Windows platforms this function has no effect.

# 23 Callback interfaces

The following group of interfaces are fundamentally different to the other SAVI interfaces in that they are implemented by the client and called by SAVI. These interfaces may be used by clients wishing to receive additional information from SAVI or notification about particular events.

A SAVI client wishing to receive these notifications can implement an object with the relevant interface as described in the following sections. The pointer to an instance of one of these objects is then passed to SAVI using RegisterNotification() in the SAVI interface (see RegisterNotification (page 58)). The client must implement QueryInterface(), AddRef() and Release() (see IUnknown (page 36)) as well as the notification functions themselves.

Many of the notification functions can return values to the SAVI object that indicate how SAVI should then proceed.

## 23.1 ISweepNotify

This callback interface provides information on the progress of a scan.

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| OnFileFound | This function is called by SAVI whenever it encounters a file or sub-file. |
| OnVirusFound | This function is called by SAVI whenever it has found a virus in an object. |
| OnErrorFound | This function is called by SAVI whenever it is unable to scan an object because of an error of some kind. |

### 23.1.1 OnFileFound

C++ syntax:

```
HRESULT SOPHOS_STDCALL OnFileFound(
 void*      Token,
 LPCOLESTR  pName
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL OnFileFound(
 void*      object,
 void*      Token,
 LPCOLESTR  pName
);
```

**Description**

This function is called by SAVI whenever it encounters a file or sub-file. For example, if SweepFile() has been called on a self-extracting Zip archive called test.exe which contains two files, file1.exe and file2.dat, OnFileFound() will be called three times, once for test.exe and once for each contained file. However, if SweepFile() has been called on a simple executable called prog.exe, OnFileFound() will be called once, for prog.exe.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the interface structure. |
| Token | This is the token value supplied to RegisterNotification() in the SAVI interface when the notification object is registered. Typically it points to some data within the SAVI client that the function needs to access. |
| pName | This is the name of the file or sub-file about to be scanned. |

## Return values

The implementation of this function must return one of the following values to determine how the scan will proceed:

| | |
|---|---|
| SOPHOS_SAVI_CBCK_CONTINUE_THIS | Go ahead and scan this sub-file. |
| SOPHOS_SAVI_CBCK_CONTINUE_NEXT | Skip this sub-file, go on to the next one. |
| SOPHOS_SAVI_CBCK_STOP | Perform no further scanning in this file. |
| SOPHOS_SAVI_CBCK_DEFAULT | Adopt default behavior for this callback. |

The SOPHOS_SAVI_CBCK_DEFAULT return value from OnFileFound() is equivalent to SOPHOS_SAVI_CBCK_CONTINUE_THIS.

If any value other than those in the above table is returned, SOPHOS_SAVI_ERROR_CALLBACK is ultimately returned to the client where it invoked the scan.

### Remarks

The implementation of this function must be thread-safe if the SAVI client is multithreading. For example, on Windows platforms all the functions in the interface should be protected with a critical section.

The implementation of this function may not internally call any SAVI interface functions.

## 23.1.2 OnThreatFound

C++ syntax:

```
HRESULT SOPHOS_STDCALL OnThreatFound(
 void*     token,
 REFIID    ResultsIID,
 void*     pResults
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL OnThreatFound(
 void*     object,
 void*     token,
 REFIID    ResultsIID,
 void*     pResults
);
```

**Description**

This function is called by SAVI whenever it has found a threat.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the interface structure. |
| Token | This is the token value supplied to RegisterNotification() in the SAVI interface when the notification object is registered. Typically it points to some data within the SAVI client that the function needs to access. |
| ResultsIID | This value indicates the type of object passed in the pResults parameter. At present it is SOPHOS_IID_SWEEPRESULTS. Other types may be supplied in the future. |
| pResults | This parameter points to an object that describes the threat found. At present it is always of type ISweepResults. Other types may be supplied in the future. |

### Return values

The implementation of this function must return one of the following values to determine how the scan will proceed:

| | |
|---|---|
| SOPHOS_SAVI_CBCK_CONTINUE_THIS | Continue searching this file or sub-file for further viruses. This feature is only useful for detecting multiple infections, not generally necessary. |
| SOPHOS_SAVI_CBCK_CONTINUE_NEXT | Go on to scan the next file or sub-file. |
| SOPHOS_SAVI_CBCK_STOP | Perform no further scanning within this file or sub-file. |
| SOPHOS_SAVI_CBCK_DEFAULT | Adopt default behavior for this callback. |

The SOPHOS_SAVI_CBCK_DEFAULT return value from OnThreatFound() is equivalent to SOPHOS_SAVI_CBCK_CONTINUE_NEXT.

If any value other than those in the above table is returned, SOPHOS_SAVI_ERROR_CALLBACK is ultimately returned to the client where it invoked the scan.

### Remarks

The implementation of this function must be thread-safe if the SAVI client is multithreading. For example, on Windows platforms all the functions in the interface should be protected with a critical section.

The implementation of this function may not internally call any SAVI interface functions.

## 23.1.3 OnErrorFound

C++ syntax:

```
HRESULT SOPHOS_STDCALL OnErrorFound(
 void*    Token,
 REFIID   ErrorIID,
 void*    pError
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL OnErrorFound(
 void*     object,
 void*     Token,
 REFIID    ErrorIID,
 void*     pError
);
```

**Description**

This function is called by SAVI whenever it is unable to scan a file or sub-file.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the interface structure. |
| Token | This is the token value that was supplied to RegisterNotification in the SAVI interface when the notification object was registered. Typically it points to some data within the SAVI client that the function needs to access. |
| ErrorIID | This value indicates the type of object passed in the pError parameter. At present it is SOPHOS_IID_SWEEPERROR. Other types may be supplied in the future. |
| pError | This parameter points to an object that describes the error found. At present it is always of type ISweepError, but other types may be supplied in the future. |

## Return values

The implementation of this function must return one of the following values to determine how the scan will proceed:

| | |
|---|---|
| SOPHOS_SAVI_CBCK_CONTINUE_THIS | Attempt to ignore the error and continue scanning the file or sub-file. |
| SOPHOS_SAVI_CBCK_CONTINUE_NEXT | Go on to scan the next file or sub-file. |
| SOPHOS_SAVI_CBCK_STOP | Perform no further scanning within this file or sub-file. |
| SOPHOS_SAVI_CBCK_DEFAULT | Adopt default behavior for this callback. |

The SOPHOS_SAVI_CBCK_DEFAULT return value from OnErrorFound() is equivalent to SOPHOS_SAVI_CBCK_CONTINUE_NEXT.

If any value other than those in the above table is returned, SOPHOS_SAVI_ERROR_CALLBACK is ultimately returned to the client where it invoked the scan.

## Remarks

The implementation of this function must be thread-safe if the SAVI client is multithreading. For example, on Windows platforms all the functions in the interface should be protected with a critical section.

The implementation of this function may not internally call any SAVI interface functions.

## 23.2 ISweepNotify2

ISweepNotify2 is a development of the ISweepNotify interface, described in ISweepNotify (page 99). It combines the member functions of ISweepNotify with two additional functions. General comments about ISweepNotify also refer to ISweepNotify2.

The member functions are listed in the following table. For a description of the duplicated member functions, see ISweepNotify (page 99).

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| OnFileFound | This function is called by SAVI whenever it encounters a file or sub-file. See OnFileFound (page 99). |
| OnVirusFound | This function is called by SAVI whenever it has found a virus in an object. See OnThreatFound (page 101). |
| OnErrorFound | This function is called by SAVI whenever it is unable to scan an object because of an error of some kind. See OnErrorFound (page 102). |
| OnClassification | This function is called by SAVI whenever it has successfully identified a file or subfile as a type recognized by SAVI. |
| OkToContinue | This function is called by SAVI whenever it is engaged in a potentially lengthy operation. |

### 23.2.1 OnClassification

C++ syntax:

```
HRESULT SOPHOS_STDCALL OnClassification(
 void*    Token,
 U32      Classifn
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL OnClassification(
 void*    object,
 void*    Token,
```

```
 U32      Classifn
);
```

**Description**

This function is called by SAVI whenever it has successfully identified a file or sub-file (e.g. a file contained within a Zip archive) as one of the types handled by the threat scanner.

**Note:** SAVI does not make calls to OnClassification() unless the StorageReport configuration option is set to 1.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| Token | The token value supplied to RegisterNotification() when this notification object was registered. Typically it points to some data within the SAVI client that the notification function may need to access. |
| Classifn | One of the file classification codes from the ID_..._STORAGE list defined in savitype.h. |

## Return values

The implementation of this function must return one of the values in the following table to determine how the scan will proceed:

| | |
|---|---|
| SOPHOS_SAVI_CBCK_CONTINUE_THIS | Go ahead and scan this sub-file. |
| SOPHOS_SAVI_CBCK_CONTINUE_NEXT | Skip this sub-file, go on to the next one. |
| SOPHOS_SAVI_CBCK_STOP | Perform no further scanning in this file. |
| SOPHOS_SAVI_CBCK_DEFAULT | Adopt default behavior for this callback (CONTINUE_THIS). |

The SOPHOS_SAVI_CBCK_DEFAULT return value from OnClassification() is equivalent to SOPHOS_SAVI_CBCK_CONTINUE_THIS.

If any value other than those in the above table is returned, SOPHOS_SAVI_ERROR_CALLBACK is ultimately returned to the client where it invoked the scan.

## Remarks

Classification codes passed to the SAVI client using this interface function may be one of the ID_..._STORAGE codes listed in savitype.h. Sophos is continually developing its products to

handle new and previously undetected file types. SAVI clients must therefore be written in such a way as to cope with new classification codes that may not yet be in the published header.

The classification codes refer to the file or sub-file most recently notified through the OnFileFound() function of this interface (see OnFileFound (page 99)).

Some file types may go through several stages of classification. For example, self-extracting executables will first be identified as executable, then as self-extracting types, and then as executable again as the underlying compressed executable file is processed. In this situation there will be several calls to OnClassification() following a single call to OnFileFound().

The implementation of this function must be thread-safe if the SAVI client is multithreading. For example, on Windows platforms all the functions in the interface should be protected with a critical section.

The implementation of this function may not internally call any SAVI interface functions.

### 23.2.2 OkToContinue

C++ syntax:

```
HRESULT SOPHOS_STDCALL OkToContinue(
 void*       Token,
 U16         Activity,
 U32         Extent,
 LPCOLESTR   pTarget
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL OkToContinue(
 void*      object,
 void*      Token,
 U16        Activity,
 U32        Extent,
 LPCOLESTR  pTarget
);
```

**Description**

This function is repeatedly called by SAVI during internal threat scan operations that could potentially use up large amounts of computer resources. Resources are assessed in terms of processor cycles, disk space or simply elapsed time. The SAVI client can decide whether or not to allow the scan to continue.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| Token | The token value supplied to RegisterNotification() when this notification object was registered. Typically it points to some data within the SAVI client that RegisterNotification() may need to access. |

| | |
|---|---|
| Activity | This parameter indicates which activity the threat scanner is currently performing, and is set to one of the SOPHOS_ACTIVITY_... codes defined in savitype.h. |
| Extent | This parameter indicates how much of the above activity has taken place so far. |
| Target | This parameter defines the name of the object upon which the Activity is being performed. |

## Return values

The implementation of this function must return one of the values in the following table to determine how the scan will proceed.

| | |
|---|---|
| SOPHOS_SAVI_CBCK_CONTINUE_THIS | Go ahead and scan this sub-file. |
| SOPHOS_SAVI_CBCK_CONTINUE_NEXT | Skip this sub-file, go on to the next one. |
| SOPHOS_SAVI_CBCK_STOP | Perform no further scanning in this file. |
| SOPHOS_SAVI_CBCK_DEFAULT | Adopt default behavior for this callback (CONTINUE_THIS). |

The SOPHOS_SAVI_CBCK_DEFAULT return value from OkToContinue() is equivalent to SOPHOS_SAVI_CBCK_CONTINUE_THIS.

If any value other than those in the above table is returned, SOPHOS_SAVI_ERROR_CALLBACK is ultimately returned to the client where it invoked the scan.

## Remarks

The client function may use this callback to interrupt processing if scanning a particular object seems to be consuming excessive resources. It is up to the client to decide, based on the Activity and Extent parameters, what action to take. This callback can be useful in terminating the processing of files that maliciously target threat scanners.

Activity and Extent parameters may be interpreted as follows:

| | |
|---|---|
| `SOPHOS_ACTVTY_CLASSIF` | This activity code indicates that the threat scanner is attempting to match the contents of the file with threat patterns. In this situation, the Extent parameter represents the number of thousands of loops around the central pattern-matching loop. |

| | |
|---|---|
| `SOPHOS_ACTVTY_DECOMPR` | The threat scanner is decompressing the contents of a compressed archive. The Extent parameter represents the number of kilobytes decompressed from the current compressed stream. |
| `SOPHOS_ACTVTY_NEXTFILE` | The threat scanner has found another subfile inside an archive-type parent. The Extent parameter represents the number of files encountered in the parent archive. It should be noted that if excessive subfiles are encountered within an archive parent, the only appropriate return value to abort processing is SOPHOS_SAVI_CBCK_STOP. |

The implementation of this function must be thread-safe if the SAVI client is multithreading. For example, on Windows platforms all the functions in the interface should be protected with a critical section.

The implementation of this function may not internally call any SAVI interface functions.

## 23.3 ISweepDiskChange

A SAVI client that wishes to receive notifications when SAVI requires part of the threat data must implement an object with this interface. This interface must be supplied to the RegisterNotification() function in the SAVI interface before either Initialise() or InitialiseWithMoniker() is called.

The notification functions return values that indicate to SAVI how to proceed.

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| OnDiskChange | This function is called by SAVI whenever a new section of VDL is required. Usually this is caused by the data being spread across more than one disk, but may be caused by the data being spread across more than one file in the same directory. |

### 23.3.1 OnDiskChange

C++ syntax:

```
HRESULT SOPHOS_STDCALL OnDiskChange(
 void*      token,
 LPCOLESTR  pFileName,
 U32        partNumber,
 U32        timesRound
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL OnDiskChange(
 void*      object,
 void*      token,
 LPCOLESTR  pFileName,
 U32        partNumber,
 U32        timesRound
);
```

**Description**

This function is invoked whenever a new section of VDL is required. Usually this is caused by the data being spread across more than one disk, but it may be caused by the data being spread across more than one file in the same directory. The implementation of this routine should prompt the user to provide the required part of the threat data (e.g. by changing disks).

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the interface structure. |
| Token | The user value supplied to RegisterNotification() in the SAVI interface. |
| pFileName | The filename of the next part of the VDL data that is being requested. |
| partNumber | The part number being requested. |
| timesRound | Counter to indicate the number of times this part has been requested (zero for the first time). This allows clients to display a different message to prompt the user on subsequent calls. |

#### Return values

If OnDiskChange() locates the requested threat data part file, SOPHOS_S_OK should be returned. SAVI may proceed with loading threat data.

If OnDiskChange() fails to locate the requested threat data part file, SOPHOS_S_FALSE should be returned. The loading of threat data should be aborted.

### Remarks

None.

## 23.4 ISaviStream

A SAVI client using the SweepStream() or DisinfectStream() functions of the ISavi3 interface must pass in a pointer to the ISaviStream interface.

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| ReadStream | Read data from the stream. |
| WriteStream | Write data to the stream. |
| SeekStream | Seek within the stream. |
| GetLength | Return the length of the stream. |

### 23.4.1 ReadStream

C++ syntax:

```
HRESULT SOPHOS_STDCALL ReadStream(
 void*  lpvBuffer,
 U32    uCount,
 U32*   bytesRead
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL ReadStream(
 void*  object,
 void*  lpvBuffer,
 U32    uCount,
 U32*   bytesRead
);
```

**Description**

This function is called when SAVI wishes to read data from the stream represented by the ISaviStream object.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the interface structure. |
| lpvBuffer | A pointer to a buffer (allocated by SAVI_ to which the data read must be copied. |
| uCount | The number of bytes to be read from the stream. |
| bytesRead | A pointer to a value that will hold the number of bytes successfully read from the stream (this may be less than or equal to the number requested). |

### Return values

The implementation of this function should return one of the following values:

- SOPHOS_S_OK if one or more bytes were returned.
- SOPHOS_SAVI_ERROR_ STREAM_ READ_FAIL if there was an error and no bytes were returned.

Attempting to read from a position past the end of the stream is an error.

### Remarks

The number of bytes to be read from the stream is represented by a U32 value because it is not logical to permit a request to read a negative number of bytes.

## 23.4.2 WriteStream

C++ syntax:

```
HRESULT SOPHOS_STDCALL WriteStream(
 const void* lpvBuffer,
 U32         uCount,
 U32*        bytesWritten
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL WriteStream(
 void*       object,
 const void* lpvBuffer,
 U32         uCount,
 U32*        bytesWritten
);
```

**Description**

This function is called when SAVI wishes to write data to the stream represented by the ISaviStream object.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the interface structure. |
| IpvBuffer | A pointer to a buffer (allocated by SAVI) containing the data to be written. |
| uCount | The number of bytes to be written (a maximum of 32K). |
| bytesWritten | A pointer to a value that holds the number of bytes successfully written to the stream (this may be less than or equal to the number requested). |

### Return values

The implementation of this function should return an appropriate value, likely to be SOPHOS_S_OK or SOPHOS_SAVI_ERROR_STREAM_WRITE_FAIL.

### Remarks

None.

## 23.4.3 SeekStream

C++ syntax:

```
HRESULT SOPHOS_STDCALL SeekStream(
 S32  lOffset,
 U32  uOrigin,
 U32* newPosition
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SeekStream(
 void* object,
 S32   lOffset,
 U32   uOrigin,
 U32*  newPosition
);
```

**Description**

This function is called when SAVI wishes to move the stream pointer of the ISaviStream object.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the interface structure. |
| lOffset | The number of bytes to move the pointer. |
| uOrigin | A value representing the origin of the pointer movement. |
| newPosition | A pointer to a value that will hold the new position of the stream pointer (relative to the start of the stream). |

### Return values

The implementation of this function should return an appropriate value, likely to be SOPHOS_S_OK or SOPHOS_SAVI_STREAM_SEEK_FAIL.

### Remarks

This function should move the stream pointer by the number of bytes specified in lOffset, relative to the position specified by uOrigin. uOrigin must be one of the following values (see savitype.h): SAVISTREAM_SEEK_SET (seek relative to start of stream), SAVISTREAM_SEEK_CUR (seek relative to current position) or SAVISTREAM_SEEK_END (seek relative to end of stream).

## 23.4.4 GetLength

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetLength(
 U32*  length
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetLength(
 void*  object,
 U32*   length
);
```

**Description**

This function is called when SAVI wishes to determine the length of the stream represented by the ISaviStream object.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the interface structure. |
| length | A pointer to a value that will hold the overall length of the stream (in bytes). |

### Return values

The implementation of this function should return an appropriate value, likely to be SOPHOS_S_OK or SOPHOS_SAVI_ERROR_STREAM_GETLENGTH_FAIL.

### Remarks

None.

## 23.5 ISaviStream2

A SAVi client using the SweepStream() or DisinfectStream() methods of the ISavi3 interface may pass a pointer to a stream object which implements either the ISaviStream or ISaviStream2 interface. Which of these interfaces is actually used is controlled by the StreamIID parameter of the method. If scanning using ISaviStream then pass SOPHOS_IID_SAVISTREAM. For ISaviStream2 pass SOPHOS_IID_SAVISTREAM2.

ISaviStream2 offers some advantages over ISaviStream. Firstly, it allows for stream sizes in excess of the 4 Gbyte limit imposed by the 32 bit ISaviStream interface. Second, it allows SAVi to request that the stream be truncated to a different length (usually zero). This offers more complete disinfection, as viral fragments at the end of a stream can be effectively eliminated.

A reference will be taken on the stream object as a result of SAVi calling QueryInterface() to request the passed interface.

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| ReadStream | Read data from the stream. |
| WriteStream | Write data to the stream. |
| SeekStream | Seek within the stream. |
| GetLength | Return the length of the stream. |
| TruncateStream | Set the length of the stream to the passed value. |

### 23.5.1 ReadStream

C++ syntax:

```
HRESULT SOPHOS_STDCALL ReadStream(
 void*   lpvBuffer,
 U32     count,
 U32*    bytesRead
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL ReadStream(
 void*   object,
 void*   lpvBuffer,
 U32     count,
 U32*    bytesRead
);
```

**Description**

This function is called when SAVi wishes to read data from the stream represented by the ISaviStream2 object.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| lpvBuffer | A pointer to a buffer (allocated by SAVI) to which the data read must be copied. |
| uCount | The number of bytes to be read from the stream. |
| bytesRead | A pointer to a value that will hold the number of bytes successfully read from the stream (this may be less than or equal to the number requested). |

#### Return values

The implementation of this function should return an appropriate value: SOPHOS_S_OK or SOPHOS_SAVI_ERROR_STREAM_READ_FAIL.

#### Remarks

In the situation where a read request simply cannot return the number of bytes requested, the function should still return SOPHOS_S_OK. The number of bytes actually read can be returned in *bytesRead. Even if this is zero, scanning may still continue. The …STREAM_READ_FAIL return code indicates a fatal error in the stream and will cause scanning to be terminated.

### 23.5.2 WriteStream

C++ syntax:

```
HRESULT SOPHOS_STDCALL WriteStream (
 const void* lpvBuffer,
 U32         uCount,
 U32*        bytesWritten
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL WriteStream (
 void*       object,
 const void* lpvBuffer,
 U32         uCount,
 U32*        bytesWritten
);
```

**Description**

This function is called when SAVI wishes to write data to the stream represented by the ISaviStream2 object.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| lpvBuffer | A pointer to a buffer (allocated by SAVI) containing the data to be written. |
| uCount | The number of bytes to be written. |
| bytesWritten | A pointer to a value that holds the number of bytes successfully written to the stream (this may be less than or equal to the number requested). |

#### Return values

The implementation of this function should return an appropriate value: SOPHOS_S_OK or SOPHOS_SAVI_ERROR_STREAM_WRITE_FAIL.

#### Remarks

In the situation where WriteStream cannot write the number of bytes requested, the function should still return SOPHOS_S_OK. The …STREAM_WRITE_FAIL return code indicates a fatal error in the stream and will cause scanning to be terminated.

### 23.5.3 SeekStream

C++ syntax:

```
HRESULT SOPHOS_STDCALL SeekStream (
 S32   lOffsetLo,
 S32   lOffsetHi,
 U32   uOrigin,
 U32*  newPosLo,
 U32*  newPosHi
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL SeekStream (
 void* object,
 S32   lOffsetLo,
 S32   lOffsetHi,
 U32   uOrigin,
 U32*  newPosLo,
 U32*  newPosHi
);
```

**Description**

This function is called when SAVI wishes to move the stream pointer of the ISaviStream2 object.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| lOffsetLo | The low 32 bits of the requested seek offset. |
| lOffsetHi | The high 32 bits of the requested seek offset. |
| uOrigin | A value representing the origin of the pointer movement. |
| newPosLo | The low 32 bits of the resultant stream pointer value. |
| newPosHi | The high 32 bits of the resultant stream pointer value. |

#### Return values

The implementation of this function should return an appropriate value: SOPHOS_S_OK or SOPHOS_SAVI_ERROR_STREAM_SEEK_FAIL.

### Remarks

Requests to seek beyond the 'end' of the stream may just indicate some corruption in the contents of the stream being scanned and should not result in an error return. The …STREAM_SEEK_FAIL return code indicates a fatal error in the stream and will cause scanning to be terminated.

The value passed in uOrigin will be one of SAVISTREAM_SEEK_SET, SAVISTREAM_SEEK_CUR or SAVISTREAM_SEEK_END, which follow the normal semantics of seek functions.

## 23.5.4 GetLength

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetLength (
 U32*   lengthLo,
 U32*   lengthHi
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetLength (
 void*  object
 U32*   lengthLo,
 U32*   lengthHi
);
```

**Description**

This function is called when SAVI wishes to determine the length of the stream represented by the ISaviStream2 object.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| lengthLo | The low 32 bits of the stream length. |
| lengthHi | The high 32 bits of the stream length. |

### Return value

The implementation of this function should return an appropriate value: SOPHOS_S_OK or SOPHOS_SAVI_ERROR_STREAM_GETLENGTH_FAIL.

### Remarks

The …STREAM_GETLENGTH_FAIL return code indicates a fatal error in the stream and will cause scanning to be terminated.

### 23.5.5 TruncateString

C++ syntax:

```
HRESULT SOPHOS_STDCALL TruncateStream (
 U32   lengthLo,
 U32   lengthHi
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL TruncateStream (
 void* object,
 U32   lengthLo,
 U32   lengthHi
);
```

**Description**

This function is called when SAVI wishes to modify the length of the stream represented by the ISaviStream2 object.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| lengthLo | The low 32 bits of the new stream length. |
| lengthHi | The high 32 bits of the new stream length. |

#### Return value

The implementation of this function should return an appropriate value: SOPHOS_S_OK or SOPHOS_SAVI_ERROR_STREAM_TRUNC_FAIL.

#### Remarks

This function will normally be (a) called as part of DisinfectStream() and (b) will usually just request that the stream length is truncated to zero.

The …STREAM_TRUNC_FAIL return code indicates a fatal error in the stream and will cause scanning to be terminated.

## 23.6 IChangeNotify

This callback interface notifies the client that a change to one or more SAVI components has occurred, either as a result of an update or through a SAVI object making a call to LoadThreatData().

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| OnChange | Invoked whenever virus data or a binary component has changed. |

### 23.6.1 OnChange

C++ syntax:

```
HRESULT SOPHOS_STDCALL OnChange (
 void*      Token
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL OnChange (
 void*     object,
 void*     Token
);
```

**Description**

Notifies client that a change to SAVI has taken place.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| Token | The user value supplied to RegisterNotification in the SAVI interface. |

#### Return values

Return value from client is ignored.

#### Remarks

None.

## 23.7 ISeverityNotify

This callback interface provides additional information about errors detected by SAVI. Not only does it call back to the client on intermediate errors but it also provides a severity rating for every error reported.

The member functions are as follows:

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| OnSevereError | Called whenever an error condition is detected by the SAVI object. |

### 23.7.1 OnSevereError

C++ syntax:

```
HRESULT SOPHOS_STDCALL OnSevereError (
 void*      Token,
 HRESULT    ErrorCode,
 U32        Severity
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL OnSevereError (
 void*      object,
 void*      Token,
 HRESULT    ErrorCode,
 U32        Severity
);
```

**Description**

Provides client with details of an error detected during a SAVI method call.

**Parameters**

| | |
|---|---|
| object (C syntax only) | A pointer to the SAVI interface structure. |
| Token | The user value supplied to RegisterNotification in the SAVI interface. |

| | |
|---|---|
| ErrorCode | The error that has occurred (see the *SAV Interface Developer Toolkit supplement* for full list of HRESULT error codes.) |
| Severity | The severity of the error (see remarks). |

## Return values

Return value from client is ignored.

## Remarks

The possible severity values passed in calls to OnSevereError are as follows:

| | |
|---|---|
| SOPHOS_ERROR_SEVERITY_SUCCESS | No error encountered. |
| SOPHOS_ERROR_SEVERITY_TRANSIENT | Single function failure - not likely to recur. |
| SOPHOS_ERROR_SEVERITY_UNKNOWN | Severity not determined. |
| SOPHOS_ERROR_SEVERITY_SUSPEND_ACTIVITY | Wait until threat data update is complete. |
| SOPHOS_ERROR_SEVERITY_REINIT_SAVI | Reinitialize SAVI interface before continuing. |
| SOPHOS_ERROR_SEVERITY_REINSTALL_SAV | Reinstall Sophos Anti-Virus. |
| SOPHOS_ERROR_SEVERITY_CRITICAL | Permanent, unrecoverable error. |

# 24 IQueryLoadedProtection

The IQueryLoadedProtection and associated interfaces provides a means of determining which identities are loaded in the threat database and some details about them.

The IQueryLoadedProtection interface can only be obtained by using QueryInterface on an existing SAVI interface. It cannot be explicitly created for example.

| | |
|---|---|
| QueryInterface | Request a particular interface from an object. See QueryInterface (page 36). |
| AddRef | Add a reference to the interface, preventing its destruction. See AddRef (page 37). |
| Release | Release a reference. See Release (page 38). |
| GetMatchingIdentities | Get a list of all identities matching a sub-string. |
| GetAllIdentities | Get a list of all identities. |
| GetSingleIdentity | Get the information for a single named identity. |

## 24.1 GetMatchingIdentities

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetMatchingIdentities(        LPCOLESTR
pIdentityName,
 REFIID     IdListIID
 void**     ppIdList,
 U32*       pMatchingIdCount
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetMatchingIdentities(
 void*        object,
 LPCOLESTR    pIdentityName,
 REFIID       IdListIID,
 void**       ppIdList,
 U32*         pMatchingIdCount
);
```

**Description**

Returns a list and the number of entries in the list of identities matching a given sub-string.

**Parameters**

| | |
|---|---|
| Object | (C syntax only) A pointer to the IQueryLoadedProtection interface structure. |
| pIdentityName | The sub-string against which the identity names are to be matched. The string is case insensitive. Wildcard matches are not supported. |
| ppIdListIID | The IID specifying the type of identity information object required. Must be SOPHOS_IID_ENUM_IDENTITY_INFO. |
| ppIdList | An address of a location to receive the address of the required enumerator. This may be NULL if the count is not required. |
| pMatchingIdCount | The address of a location to receive the number of matching identities. This may be NULL if the content is not required. |

### Remarks

One of ppIdLIst and pMatchingIdCount must be non-NULL.

The string compare is case insensitive and looks for a sub-string and compares the given string against the full name of the threat as returned by the GetName function of the IdentityInfo interface.

## 24.2 GetAllIdentities

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetAllIdentities(
 REFIID   IdListIID,
 void**   ppIdList,
 U32*     pMatchingIdCount
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetAllIdentities(
 void*     object,
 REFIID    IdListIID,
 void**    ppIdList,
 U32*      pMatchingIdCount
);
```

**Description**

Returns a list and the number of entries in the list of all identities.

**Parameters**

| | |
|---|---|
| Object | (C syntax only) A pointer to the IQueryLoadedProtection interface structure. |
| ppIdListIID | The IID specifying the type of identity information object required. Must be SOPHOS_IID_ENUM_IDENTITY_INFO. |
| ppIdList | An address of a location to receive the address of the required enumerator. This may be NULL if identity information is not required. |
| pMatchingIdCount | The address of a location to receive the number of matching identities. This may be NULL if the count is not required. |

### Remarks

One of ppIdLIst and pMatchingIdCount must be non-NULL.

## 24.3 GetSingleIdentity

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetSingleIdentity(
 LPCOLESTR    pIdentityName,
 REFIID       IdInfoIID,
 void**       ppIdInfo
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetSingleIdentity(
 void*        object,
 LPCOLESTR    pIdentityName,
 REFIID       IdInfoIID,
 void**       ppIdInfo
);
```

**Description**

Returns the identity information for the identity with the given name.

**Parameters**

| | |
|---|---|
| Object | (C syntax only) A pointer to the IQueryLoadedProtection interface structure. |
| pIdentityName | The name of the desired identity. The name is case insensitive. |
| ppIdInfoIID | The IID specifying the type of identity information object required. Must be SOPHOS_IID_IDENTITY_INFO. |

| | |
|---|---|
| ppIdInfo | An address of a location to receive the address of the required enumerator. This may be NULL if the count is not required. |
| pMatchingIdCount | An address of a location to receive the address of the required IdentityInfo. |

## Remarks

The string compare is case insensitive and compares the given string against the full name of the threat as returned by the GetName function of the IdentityInfo interface.

# 25 IIdentityInfo

IIdentityInfo is the interface to an object that represents some details of a threat database identity. The interface can be used to retrieve information about the identitity, such as its name.

An enumerator of IIdentityInfo is created by the GetMatchingIdentities() and GetAllIdentities() methods of IQueryLoadedProtection.

| | |
|---|---|
| QueryInterface | Standard |
| AddRef | Standard |
| Release | Standard |
| GetName | Gets the name of the threat |
| GetNameWithoutType | Gets the name of the threat but without the type qualifier |
| IsVariant | Is the threat a variant on a base threat |
| IsFamily | Is the identity for a family of threats |

## 25.1 GetName

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetName(
 U32         ArraySize,
 LPOLESTR    pName,
 U32*        pNameLength
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetName(
 void*       object,
 U32         ArraySize,
 LPOLESTR    pName,
 U32*        pNameLength
);
```

**Description**

GetName returns the name of the threat for which the identity is targeted, e.g. W32/Netsky-G.

**Parameters**

| | |
|---|---|
| Object | (C syntax only) A pointer to the IIdentityInfo interface structure. |
| ArraySize | The size in characters of the buffer specified by pName. |
| pName | The address of a location to receive the name of the threat. pName may be NULL if the name is not required. |
| pNameLength | The address of a location to receive the actual length of the name. pNameLength may be NULL if the length is not required. |

### Remarks

Either of pName and pNameLength may be NULL, but not both.

The lengths are in characters, not bytes, and must include the terminator character.

## 25.2 GetNameWithoutType

C++ syntax:

```
HRESULT SOPHOS_STDCALL GetNameWithoutType(
 U32         ArraySize,
 LPOLESTR    pName,
 U32*        pNameLength
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL GetNameWithoutType(
 void*       object,
 U32         ArraySize,
 LPOLESTR    pName,
 U32*        pNameLength
);
```

**Description**

GetNameWithoutType returns the name of the threat for which the identity is targeted but without the threat type qualifier, e.g. Netsky-G.

**Parameters**

| | |
|---|---|
| Object | (C syntax only) A pointer to the IIdentityInfo interface structure. |
| ArraySize | The size in characters of the buffer specified by pName. |

| | |
|---|---|
| pName | The address of a location to receive the name of the threat. pName may be NULL if the name is not required. |
| pNameLength | The address of a location to receive the actual length of the name. pNameLength may be NULL if the length is not required. |

### Remarks

Either of pName and pNameLength may be NULL, but not both.

The lengths are in characters, not bytes, and must include the terminator character.

## 25.3 IsVariant

C++ syntax:

```
HRESULT SOPHOS_STDCALL IsVariant(
 U32*   pVariant
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL IsVariant(
 void*    object,
 U32*     pVariant
);
```

**Description**

IsVariant returns whether the threat for which the identity is targeted is a variant of another threat.

**Parameters**

| | |
|---|---|
| Object | (C syntax only) A pointer to the IIdentityInfo interface structure. |
| pVariant | The address of a location to receive a boolean value which is non-zero if the threat is a variant. |

### Remarks

None.

## 25.4 IsFamily

C++ syntax:

```
HRESULT SOPHOS_STDCALL IsFamily(
 U32*    pFamily
);
```

C syntax:

```
HRESULT SOPHOS_STDCALL IsFamily(
 void*     object,
 U32*      pFamily
);
```

**Description**

IsFamily returns whether the identity is targeted at a family of threats.

**Parameters**

| | |
|---|---|
| Object | (C syntax only) A pointer to the IIdentityInfo interface structure. |
| pFamily | The address of a location to receive a boolean value which is non-zero if the identity is targeted at a family of threats. |

### Remarks

None.

# 26 Technical support

You can find technical support for Sophos products in any of these ways:

- Visit the SophosTalk community at community.sophos.com/ and search for other users who are experiencing the same problem.
- Visit the Sophos support knowledgebase at www.sophos.com/en-us/support.aspx.
- Download the product documentation at www.sophos.com/en-us/support/documentation.aspx.
- Open a ticket with our support team at https://secure2.sophos.com/support/contact-support/support-query.aspx.

# 27 Legal notices