SAVAPI

Documentation

AVIRA

Table of Contents

1. Introduction

The **S**ecure **A**nti**V**irus **A**pplication **P**rogramming **I**nterface (SAVAPI) from Avira provides an interface to detect malware and repair infected files. The cross-platform engine behind the interface is based on the technology of Avira Operations GmbH & Co. KG.

The SAVAPI interface is written in C and can be compiled by any common C/C++ compiler on many operating systems. Our goal is to give third-party developers an opportunity to easily add antivirus functionality to their products. Typical applications include:

- Email gateways
- Desktop and server solutions
- www/ ftp proxies
- Firewalls
- Backup applications

1.1 General features

Features	**Library**	**Service**	**Client Library (combo)**
Check Avira signature	yes	yes	yes
Communication over network sockets	no	yes	yes
Configure Avira Engine location	yes	yes	yes
Configure licence file location	yes	yes	yes
Configure logging options	yes (via API)	yes (via command line or configuration file)	yes (via API)
Configure network connection properties	no	yes (via command line or configuration file)	yes (via API)
Configure scan options	yes (via API)	yes (via command line or configuration file)	yes (via API)
Configure VDF location	yes	yes	yes

Features	Library	Service	Client Library (combo)
Repair files in memory	possible with user-implemented FOPS	no	no
Repair files on disk	yes	yes	yes
Scan files in memory*	yes	no	yes
Scan files on disk	yes	yes	yes
Scan mails and mailboxes	yes	yes	yes
Scan regular archives	yes (disabled by default)	yes (disabled by default)	yes (disabled by default)
Scan threads management	yes (must be implemented by the integrator)	limited (only the number of working threads can be configured)	yes (must be implemented by the integrator)
Report scan errors	yes (via API)	yes (via text-based protocol)	yes (via API)
Report scan information (warnings, info, progress)	yes	yes	yes
Report scan results	yes (via API)	yes (via text-based protocol)	yes (via API)
Supervisor	no	yes	yes (only for SAVAPI Service)
UNICODE support	yes	yes	yes
User-implemented FOPS	yes	no	no

* The structure used for scanning objects from memory uses the type: "`unsigned int`" for the size of the object in memory. This limits the maximum size for a scanned object to approximately 4GB (2^32) on most of the 64-bit systems.

1.2 System requirements

Supported OS and hardware platforms

The **32-bit SAVAPI version** runs on the following platforms:

- FreeBSD 6.2 x86 32-bit
- Linux x86 32-bit (starting with glibc 2.2)

- On request: Linux s390 32-bit (starting with glibc 2.2)
- Linux x86 64-bit (starting with glibc 2.2, if the 32-bit compatibility libs are installed on the system)
- Mac OS X (starting with Mac OS X Leopard)
- OpenBSD 3.9 x86 32-bit
- Solaris 9 sparc
- Solaris 10 sparc
- Windows, with the latest SP:
  - Windows XP
  - Windows Server 2003
  - Windows Server 2003 R2
  - Windows Vista
  - Windows 7
  - Windows Server 2008
  - Windows Server 2008 Core
  - Windows Server 2008 R2
  - Windows Server 2008 R2 Core

The **64-bit SAVAPI version** runs on:

- Linux x86 64-bit (starting with glibc 2.4)
- Mac OS X (starting with Mac OS X Leopard), if the hardware is 64-bit compatible
- Solaris 9 sparc
- Solaris 10 sparc
- Windows 64-bit, with the latest SP:
  - Windows XP
  - Windows Server 2003
  - Windows Server 2003 R2
  - Windows Vista
  - Windows 7
  - Windows Server 2008
  - Windows Server 2008 Core
  - Windows Server 2008 R2
  - Windows Server 2008 R2 Core

Minimum system requirements (for SAVAPI Service)

- 32-bit or 64-bit CPU, min. 1.6 GHz
- 512 MB RAM (exclusively for SAVAPI)
- 1GB HDD space (needed for unpacking archives)

Note
The system requirements depend on how the application is using SAVAPI. The suggested values are computed for the service default values. If the application that uses the SAVAPI Service modifies the default configuration, the requirements will be different.

On Windows systems, the Microsoft Visual C++ 2008 SP1 Redistributable Package for the system architecture you want to run should also be installed. Download links:

- for 32-bit systems:
  http://www.microsoft.com/downloads/en/details.aspx?familyid=A5C84275-3B97-4AB7-A40D-3802B2AF5FC2&displaylang=en
- for 64-bit systems:
  http://www.microsoft.com/downloads/en/details.aspx?FamilyID=BA9257CA-337F-4B40-8C14-157CFDFFEE4E

Limitations

- SAVAPI imposes a limit of 300 scan threads that can run in parallel. The best performance is generally reached when 1 or maximum 2 threads per processing unit (CPU core) are used. However, multiple threads per processing can be used in most of the situations, because the scan performance is mostly influenced not by the processing power but by the I/O operations.
  As there is no formula to calculate the optimal threads per processing unit number for all possible scenarios, SAVAPI integrators should try different setups and decide which one fits the environment better. The default values were chosen based on generic tests and they might not always be the best option.
- Based on stress tests under several versions of Windows, **we do not recommend creating more than 10 connections per second** in parallel to SAVAPI. This limitation has been observed while creating and destroying many connections (each connection created to scan a single file). Microsoft introduces a few new TCP/IP settings, starting with Windows XP SP2, in order to "reduce the threat" of worms spreading fast without control. The number of possible TCP connection attempts per second is limited to a certain amount.
  This behavior may deny the connection to the server for some scanning clients, causing instability in both client and server (because of the so-called "half connection" - **connected**, but **not accepted** by the service).
  For more details, please check: http://support.microsoft.com/kb/158474
- On UNIX systems, the opened files limit (`ulimit -n`) is very important for SAVAPI-based applications. The open files limit is shared between SAVAPI instances. If the limit is too low, it may happen that scan errors or failed connection errors are reported. The

higher the archives' recursion level, the more open files will be used during the archive processing. The recommended value for the open files limit is: *archive_max_rec * workers_number*.

Note
SAVAPI is not a real-time scanner; it does not monitor any file operations. The application implemented using SAVAPI can add real-time monitoring, if desired.
SAVAPI does not provide any anti-adware or anti-spyware features.
SAVAPI does not provide any Registry protection features.
SAVAPI is not a dedicated anti-rootkit product; it offers only basic anti-rootkit detection via VDF signatures.

Signed binaries

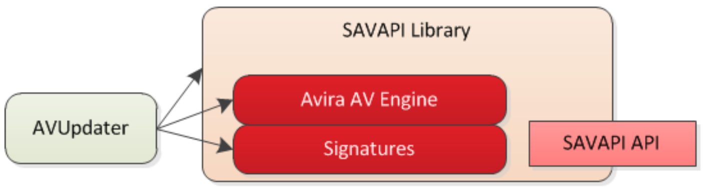
SAVAPI binaries are especially signed by Avira, with its own algorithm, in order to check the binaries' integrity (not with a certificate). If the DLL or EXE is changed, the product does not start anymore.

Because of this self-test, you cannot sign the binaries, nor change them in any way.

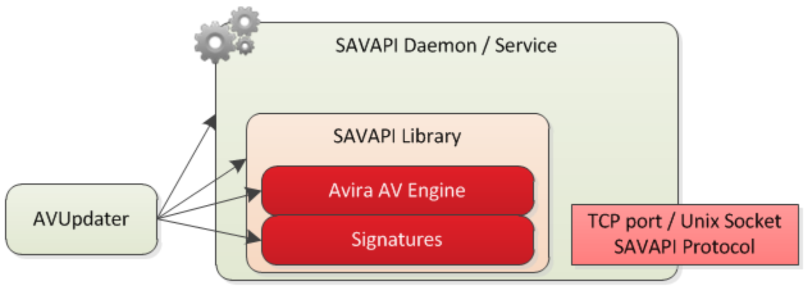
1.3 Integrating SAVAPI

SAVAPI provides more interfaces and modes of use to the customers. These modes allow the customers to integrate SAVAPI according to their needs. The figure below illustrates each of the three modes, highlighting possible ways to interact with SAVAPI.

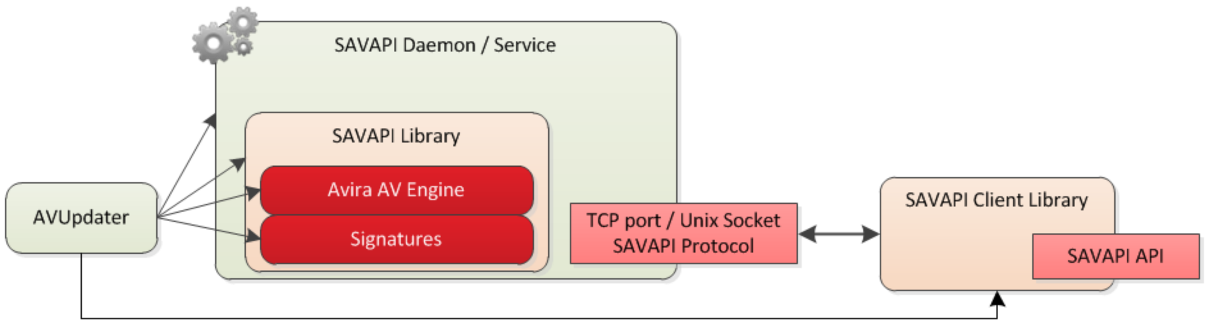
SAVAPI Library Mode:



SAVAPI Daemon / Service Mode:



SAVAPI Client Library + Daemon / Service Mode:



The following chapters present these modes, some recommended usage scenarios and advantages for every SAVAPI mode:

- Chapter 2. SAVAPI Library
- Chapter 3. SAVAPI Service
- Chapter 4. SAVAPI Client Library

2. SAVAPI Library

2.1 General description of SAVAPI Library

SAVAPI Library is a cross-platform library that allows the customer to initialize the scan engine, configure its options and scan a file. The functionality is exposed through an API written in C language.

The application based on SAVAPI Library must either load the library on runtime or link with the library at compile time; then call the library API to perform initialization, configuration, scanning, etc.

The application is also responsible for maintaining the scan jobs, creating threads to scan, allocating data, terminating the instances and uninitializing the library. If you want to have complete control over these resources, then you should opt for SAVAPI Library mode.

2.2 Integration of SAVAPI Library

The client-designed applications should link to the SAVAPI Library (either implicit or explicit) and use the SAVAPI Library API to access the SAVAPI technology. The SAVAPI Library API allows the user to configure the scan engine and other SAVAPI options, to process files and retrieve information about the processing status. For more details, please refer to the API documentation.

A very simple example of how one can use the SAVAPI Library is offered in the SAVAPI SDK kit. (You can refer to the SDK documentation under *doc/README.)*

2.3 Configuration of SAVAPI Library

The SAVAPI Library can be configured through the API. In order to set an option, the customer will just have to call the `SAVAPI3_set()` function with the appropriate parameters.

2.3.1 File Operation Structure (FOPS)

Avira Engine's FOPS is a mechanism which allows any SAVAPI client to implement a special method of scanning objects, other than files, on disk. Obviously, the name of the feature shows that its origin is file-related (File OPS), but it is possible to implement almost any access method, as long as the interface is correctly implemented.

If you want to process a different kind of data, other than files (an I/O stream for example), you can implement your own File Operation Structure (FOPS) and provide it to SAVAPI Library. FOPS is a collection of functions that give the SAVAPI Library access to the desired user data. The user would not be able to scan this data only with SAVAPI Service, nor with the SAVAPI Service + SAVAPI Client Library combo.

Real implementations of the FOPS include, but are not limited to:

- Implementing scanning on an encrypted file system – The application which uses the SAVAPI Library implements the logic for reading and writing in the encrypted file system, allowing the engine to read the files as if they were normally stored (without encryption).
- Implementing scanning on memory streams – The application is closely integrated with a server (email, file, http) which processes its data in memory, receiving the files as streams.

What is FOPS?

The FOPS structure is a collection of I/O routines which are usually used to access files on a normal disk, having a standard File Allocation Table. These routines are organized in an array of pointers to functions.

Note
The order in this array must be kept exactly as in the examples in the SDK.

Each function receives certain parameters, depending on the function type.

```
AVE_FOPS engine_fops = {
my_open,my_close,my_read,my_write,my_tell,my_seek,my_getfattr,my_s
etfattr,my_getfsize,my_unlink,my_rename,my_access,
my_malloc,my_free,my_gets,my_puts,my_getc,my_putc, my_ungetc,
my_flush, my_get_last_error };
```

Note
It is mandatory that all functions are defined. Do not simply add a NULL instead of a pointer. Doing so will make the engine abort its initialization.

The engine uses such functions to process any object, right after the `SAVAPI3_scan` function is called. Internally, there is such a structure already implemented, having an implementation similar to the functions in the two examples provided with the SDK (but not the same). The two examples are fully functional, but simplified in order to allow an easy understanding of the process.

Having such a structure in place and allowing the user to change it at any time, SAVAPI has to make sure to differentiate between "normal" engine operations (like opening the VDF files and creating or deleting temporary files) and the object scanning operations. The engine needs the normal FOPS for the first category because these files must always reside on disk (a RAM Disk is also considered a disk). The second category, representing the objects which must be scanned, have to be accessed using the user's FOPS.

How to use FOPS: the SAVAPI3_scan call

Once the `SAVAPI3_set_fops` is called with a pointer to the structure defined above, the engine will dynamically use one of the two FOPS structures, depending on the context.

This function requests the engine to scan an object.

If you scan a file on disk, the second parameter is the filename, so no further information is required.

If you scan an object or a stream, there are some things which have to be specially configured.

The entire idea is to provide to the `FOPS::open` function enough information to access the object referenced by the second parameter of the `SAVAPI3_scan` function (`const char* file_name`). Since this is a character buffer, you have to write somehow the address and the size of the memory area where the object to be scanned resides.

For example, you could use a string like this:

```
SAVAPI3_scan(&inst,"0xAddress,size,Filename.ext")
```

Where "`0xAddress`" represents the address of the pointer to the memory area where the object resides and the "`size`" is the size of the memory area. If you provide the "`Filename.ext`" parameter, the engine is going to be able to make some assumptions based on the file type and from case to case perform deeper analysis on the object.

This means that the open function which has the prototype

```
int fops_open (FOPS_HANDLE *fh, void *filename, int mode, void
*file_context, void *fops_context)
```

receives the following information:

```
FOPS_HANDLE *fh : undefined
```

You have to allocate this according to your needs. For example, having the value given above, you must save the address and the size. Later on, you have to convert this address to a valid pointer.

```
Struct MyFH
{
void* addr;
long size;
}
```

Do not forget to allocate your `FOPS Handle` structure:

```
MyFH* myfh = (MyFH)malloc(sizeof(MyFH));
… do processing here …
*fh = (void*)myfh;
```

1. First extract the "`address`" and the "`size`" from the "`filename`".
2. Convert the hexadecimal string into an `int`
   ```
   int32 address = convertHexStringToInt32( "0xaddress");
   ```

```
// you must write your own conversion function to a safe type
// (32 and 64 bits)
```

3. Assign to the file handle pointer the value

```
fh->addr = (char*)address;
fh->size = size;
```

> **Note**
> This is just a simplified example which is not safe and probably not cross-platform.

```
void *filename: "0xAddress,size,filename.ext"
```

This is the filename provided in the `SAVAPI3_scan` function. Extract the address and the size and use them to allocate the "`fh`"

```
int mode : 0 (OPEN_RO)
```

```
void *file_context: undefined or NULL – ignore this value
```

```
void *fops_context: the same value you defined when you called
SAVAPI3_set_fops
```

> **Note**
> The handle that was allocated in the `open` function must be released in the `close` function.
> It is possible that the `open` function will be called multiple times for the same file. The `close` function will also be called multiple times, corresponding to the number of `open` calls, for the same handle.
> In the function `getfsize` where you have to return the size, just return `fops->size`.

2.4 SAVAPI Library logging

2.4.1 Initialization

In order to receive the messages logged by the SAVAPI Library, the user needs to have a special callback function. The function needs to have the following format:

```
void(*SAVAPI3_LOG_CALLBACK) (SAVAPI3_LOG_LEVEL log_level, const
SAVAPI_TCHAR *message, void *user_data);
```

The function's parameters are described in the SAVAPI Library API.

Besides the function's header, there are no restrictions about the implementation.

The function must be registered in the SAVAPI Library with the following function:

```
int SAVAPI3_EXP SAVAPI3_set_log_callback(SAVAPI3_LOG_CALLBACK
log_fct, SAVAPI3_LOG_LEVEL min_level, void *user_data);
```

Note
This function can be called without initializing the library. The function's parameters are described in the SAVAPI Library interface section.

2.4.2 Configuration

The logging mechanism can be configured using the `SAVAPI3_set_log_callback` function. The `min_level` parameter will set the minimum log level of the messages received through the registered callback. All logs from higher levels will be logged. The `DEBUG` level has the highest verbosity and the `ERROR` level has the lowest verbosity.

The log-levels in decreasing order:

- `SAVAPI3_LOG_DEBUG` – this is the most verbose log level; messages on this level will contain low level information, such as:
  - details about the initialization and un-initialization of SAVAPI Library
  - details about the creation and release of instances
  - information about interface function behavior
- `SAVAPI3_LOG_INFO` – messages on this level will contain information about the general workflow of the library such as:
  - key stages of the initialization and un-initialization
  - key stages in the creation and release of library's instances
- `SAVAPI3_LOG_WARNING` – messages on this level will contain information about unexpected but recoverable errors that will not stop the execution of the library or of its functions. For example:
  - invalid product ID given in the initialization structure
  - VDF files are older than two weeks
- `SAVAPI3_LOG_ALERT` – messages on this level will contain information about alerts that have been raised during runtime. For example:
  - malware was found while scanning a file
- `SAVAPI3_LOG_ERROR` – this is the least verbose log level; messages on this level will contain information about unrecoverable errors that occur during runtime, in one of the library's functions. All error logs will contain a short description of the action that failed, followed by the error code, and the error's description.

If the user wants to stop the logging of the SAVAPI Library, the `SAVAPI3_set_log_callback` function must be called with `NULL` as the `log_fct` parameter. Also, if the log function or the minimum log-level needs to be modified, the same function can be called again with the new configuration.

In addition, the `user_data` parameter grants the user freedom to send any necessary information to the callback function. Usually this parameter contains a user-defined structure with multiple fields. For example:

- the log-file name or the log-file descriptor which can be used by the log callback to write the logs inside the respective file
- information about the running instance number: in a configuration that uses multiple scanning instance, the log messages can be divided between multiple files or have different formats for a better tracking of their thread source
- counters for the number of errors or alerts found in a running session
- counters for the number of infected files found

2.4.3 Logging guidelines

The logging mechanism allows users to receive feedback about the library's activity and performance. It should be started before the initialization of the SAVAPI Library, in order to log any possible errors.

The setting of the minimum log level determines the amount of log messages that will be received by the log callback. If the log level is set to the minimum level (`SAVAPI3_LOG_DEBUG`), the application's performance may be affected, because of the increased number of disk write-accesses that are performed during the writing of the log messages into the log files. This level should only be used to track and debug issues. For example: after receiving an unexpected error, the behavior can be reproduced on this level and the user may be able to track the source of the problem or give a more detailed feedback to the Support team.

In order not to affect the application's performance, the following log level configurations are recommended:

- `SAVAPI3_LOG_INFO` – should be the default setting. It offers all available information about the unexpected events (alerts, errors, warning) and keeps track of the key stages in the initialization workflow.
- `SAVAPI3_LOG_ERROR` – if the application's workflow is not important, or if the application is stable enough not to worry about unexpected events such as warnings and alerts, this level will guarantee minimum feedback and maximum performance.

2.5 Selective file repair

Here is how you can check if a file is repairable and start a repair attempt:

Note
Please note that files contained in archives or other containers cannot be repaired, unless the application itself unpacks the contents and provides the files to SAVAPI one by one. In this case it is the application's responsibility to pack the contents again.

2.5.1 Solution 1

1. Scan the file.

   In case of an infection, the callback for `SAVAPI3_CALLBACK_REPORT_FILE_STATUS` is triggered.

2. The application needs to check `SAVAPI3_FILE_STATUS_DATA` which contains information about the detected malware, in the `SAVAPI3_MALWARE_INFO` fields.

   In case the field `SAVAPI3_FILE_STATUS_DATA.malware_info.removable` has the value 1, then the file is repairable.

   `SAVAPI3_CALLBACK_REPORT_FILE_STATUS` is triggered again to signal that the scan has finished (with this call, the repairable field will be reset to 0).

3. At this point the application could ask the users if they want to clean the file.

4. In case the users decide to clean the file:

   - Set `SAVAPI3_OPTION_REPAIR` to `ON`,
   - Scan the file again.

   If the repair fails, the callback for `SAVAPI3_E_REPAIR_FAILED` is triggered.

   Otherwise the callback for `SAVAPI3_CALLBACK_REPORT_FILE_STATUS` is called, reporting a clean file.

2.5.2 Solution 2

1. Before scanning the file, set the option `SAVAPI3_OPTION_NOTIFY_REPAIR`.

2. Scan the file.

   In case the file is repairable the callback `SAVAPI3_CALLBACK_SCAN_DETAILS_REPORT` will be triggered, so you need to have it associated with a function already.

3. In the callback function, you need to check that the data type is `SAVAPI3_REPORT_REPAIRABLE`, so that you know the callback was called for a repairable file.

4. At this point the application could ask the users if they want to clean the file.

   The information about the file is contained in the data structure `SAVAPI3_repairable_data`, the fields: `SAVAPI3_malware_info` and `SAVAPI3_file_info`. So, at this stage the file has been scanned and you know it is repairable.

5. In case the users decide to attempt a repair:

   - Set `SAVAPI3_OPTION_REPAIR` to `ON`,

- Set `SAVAPI3_OPTION_NOTIFY_REPAIR` to `OFF`, so the callback will not be called again,
- Scan the file again.

If the repair fails, the callback `SAVAPI3_E_REPAIR_FAILED` is triggered.

If the repair succeeds, then `SAVAPI3_CALLBACK_REPORT_FILE_STATUS` is triggered, reporting a clean file.

3. SAVAPI Service

3.1 General description of SAVAPI Service

SAVAPI Service is an application based on SAVAPI Library. It offers all the features of the SAVAPI product, **except**:

- Implementing your own FOPS (possible with the SAVAPI Library)
- Scanning files mapped in memory (possible only if the combo SAVAPI Service + SAVAPI Client Library is used)

SAVAPI Service is a multi-thread application. When started, it creates a pool with worker threads and listens to client requests on a network socket (TCP/IP or UNIX local socket). When a client request arrives, it is associated to a worker thread that will process and answer all the client commands.

SAVAPI Service communicates with the client via a text-based protocol. Through this protocol, the client applications can

- configure the service,
- obtain information about the service status, configuration option values or licensing,
- send scan requests,
- stop the service.

The maximum command size allowed by the SAVAPI Service is set to:
`2 * PATH_MAX + 64`
This setting allows users to scan any possible file-path of the current file-system.

- `PATH_MAX` is the maximum length of a file-system path. It is a platform-dependent system variable on UNIX. On Windows, it is limited to `8192`.
- The maximum value allowed by SAVAPI includes the command's arguments, line-terminating characters (EOL) and also the string terminating NULL character:
  `<command> <arguments><EOL><\0>`

3.2 Integration of SAVAPI Service

There are two possibilities to use the SAVAPI Service.

1. **Use only the SAVAPI Service**

In this mode, the SAVAPI Service will be started and will listen on a specified interface (or the default interface, if none is specified).

The client application(s) will:

- connect to the network socket where SAVAPI Service is listening,
- perform the handshake with the service,

- set/ read options value,
- send scan requests,
- read scan answers.

The client application is responsible to implement all the communication-related workflow (create a socket, connect to the SAVAPI Service listening socket, read and write from the socket, react to socket I/O errors) and its own workflow (create threads for parallel requests, prioritize the scan requests, etc).

2. **Use the combo SAVAPI Service + SAVAPI Client Library**

The SAVAPI Client Library is using the SAVAPI SDK API, but instead of doing the work itself, it will just communicate the requests to a running SAVAPI Service. It provides an easier way to communicate with the SAVAPI Service. Instead of creating a socket and maintaining all the socket communication issues, the client has to take care only of the application workflow (prioritize scan requests, create threads/workers, etc).
See 4. SAVAPI Client Library.

3.3 Configuration of SAVAPI Service

There are more ways to configure the SAVAPI Service features and options.

- **Using command line parameters**
  The service accepts command line parameters at startup. A complete list with the accepted parameters, their functions and accepted values is detailed below.
  See 3.3.1 SAVAPI command line parameters.
- **Using configuration file options**
  Through a command line parameter, the service can receive a configuration file containing the options and their values. A complete list with the accepted options, their default values and their valid values is detailed below. See 3.3.2 SAVAPI configuration file options.
- **Using the SAVAPI protocol SET commands**
  SAVAPI Service uses a text-based protocol to communicate with the outside world. Through the protocol commands, the SAVAPI Service can be configured to answer to commands and requests. See 3.3.3 SAVAPI Service protocol.

3.3.1 SAVAPI command line parameters

Note
Some of the command line parameters are not available for all the supported platforms. For example, the parameters regarding the user and group are available only for UNIX-based operating systems, because in Windows the service runs in the background as a Service under the Local System Account user.

SAVAPI will consider all numeric values as decimal (base 10).

The command line parameter types include:

- Options – allow you to set some specific options at program startup
- Commands – allow you to send some specific commands to the service/daemon
- Information – allow you to retrieve various data regarding the service/daemon (help message, version information, etc)

Options

- **-N**

  Starts the SAVAPI Service in the foreground. SAVAPI will run as a process.

  This option is not activated by default. The service will start in the background.

- **--pool-scanners=<scanners number>**

  Starts the SAVAPI Service with the specified number of workers. Default: 24; Minimum: 1; Maximum: 300.

- **-C, --config=<configuration file location>**

  Specifies the location of the SAVAPI Service configuration file. No default value specified.

- **--temp=<scanning temporary files location>**

  Specifies the location of the scanning temporary files. The temporary files created while scanning and unpacking will be stored in this location. Default: */var/tmp* on UNIX,

  %temp% on Windows.

- **--key-dir=<key file location>**

  Specifies the location of the SAVAPI Service key files. The default value is the location of the SAVAPI Service binary.

- **--vdf-dir=<VDF files location>**

  Specifies the location of the VDF files.

  Default is the location of the SAVAPI Service binary.

- **--pid-dir=<PID file location>**

  Specifies the location of the PID files.

  The default location is: '*/var/tmp*'.

> **Note**
> This option is not available on Windows systems.

- **--socket-file=<socket file location>**

  Specifies the location and the name of the UNIX local socket.

Default: */var/tmp/.savapi3*

> **Note**
> This option is not available on Windows systems.

- **--socket-permissions=<str>**

  Specifies the UNIX local socket permissions.

  By default, the socket file is created with the same permissions as the running SAVAPI Service.

> **Note**
> This option is not available on Windows systems.
>
> The `user` and `group` options are obsolete and will be removed.

- **--log-file=<log file location>**

  Specifies the log file name and location. There is no default value.

- **--ldpath=<dependencies location>**

  Specifies the path to SAVAPI libraries. Default: location of the SAVAPI Service binary.

- **--install**

  On Windows, it registers the service in Service Control Manager (description, display name, binary image path, auto start).

> **Note**
> This option is available only on Windows systems.

- **--uninstall**

  On Windows, it removes the registration of the service from the Service Control Manager.

> **Note**
> This option is available only on Windows systems.

- **--tcp=[host:]port**

  Specifies the port (and optionally, the host) to which the SAVAPI TCP/IP listening socket bounds. If no host is provided, SAVAPI bounds to localhost.

  `host` can be either a host name or an IP address in any accepted format (dotted or not): decimal, hexadecimal, octal.

> **Note**
> SAVAPI supports only IPv4 addresses.

On Linux systems this forces SAVAPI to use TCP/IP sockets rather than UNIX local sockets.

On Windows systems this option is **mandatory**. The service will not start, if the interface is not specified, either through this option or through the configuration file.

- **--pool-connections=<pool connections number>**

  Defines the length of the pending connections queue. Default: 48; Minimum: number of `--pool-scanners`; Maximum: 900.

- **--ave-dir=<AVE files location>**

  Specifies the location of the engine files.

  The default is the location of SAVAPI Service binary.

- **--no-spv**

  SAVAPI starts without the supervisor process.

Note
This option is not available on Windows systems.

Commands

- **--stop**

  Stops SAVAPI. This will work only if you specify the running SAVAPI interface, either by command line options (`--tcp`) or by configuration file (`--config=<configuration_file>`).

Note
The `stop` command will not be executed if the user who sends the command does not have enough privileges (on Windows this means the user cannot send commands to services, on Linux it means that the user does not have enough privileges to connect to the SAVAPI Service).

- **--status**

  Retrieves the current status of SAVAPI. This will work only if you specify the running SAVAPI interface, either by command line options (`--tcp`) or by configuration file (`--config=<configuration_file>`).

Note
The `status` command will not be executed if the user who sends the command does not have enough privileges (on Windows this means the user cannot send commands to services, on Linux it means that the user does not have enough privileges to connect to the SAVAPI Service).

Information

- **-V, --version**

  Displays the version information.

```
# ./savapi --version
product build: Linux (i386, glibc 2.2)
product version:  3.3.0.3
engine version:   8.2.6.54
packlib version:  8.2.10.10
vdf version:      7.11.14.134
```

- **-h, --help, -?**

  Displays the help message.

3.3.2 SAVAPI configuration file options

All the lines starting with '#' or ' ' (space) in the configuration file are ignored, when options are parsed. The format of the configuration is as much as possible a 1:1 match with the SAVAPI TCP/IP protocol.

Accepted forms of writing the parameters in the configuration file are:

'`Attribute=value`'

'`Attribute value`'

Attributes are case insensitive. Values are case sensitive. Attributes are separated by CR.

The default values are provided for all the options, where such values exist.

The SAVAPI Service will not start, if invalid values are provided for any of the configuration file options. An error will be printed regarding the invalid value.

SAVAPI will consider all numeric values as decimal (base 10).

System hard-limit:

On UNIX, SAVAPI uses the system hard-limit (see `ulimit -Hn`) for the maximum number of opened files allowed by the process. This limit is shared between SAVAPI connections (the higher the archives' recursion level, the more files will be created). When using large values for the options `PoolScanners`, `PoolConnections` and `ArchiveMaxRec`, unexpected errors (scanning proc-errors, failed connections) may occur, if the system hard-limit is too small.

SAVAPI Service Running Options

- **CurrentWorkingDirectory**

  `CurrentWorkingDirectory </path/to/working/folder>`

  Sets the current working directory for SAVAPI. This eliminates the need to specify full paths in file names, when commands that accept file names as parameters are used.

  Available values: only absolute paths are accepted; relative paths are not accepted and the service exits with an error.

  Default value: the SAVAPI Service binary location.

- **TextMode**

  `TextMode <ASCII-PRINT | LOCALE | UTF-8>`

  Specifies the method used for character encoding. The SAVAPI protocol provides three modes for handling text. The modes are applied to both incoming and outgoing data.

  Available values:

  - `ASCII-PRINT`
    Only printable characters of the ASCII set are considered valid characters. These include the characters 9 and 32-126. All other character values are converted to '?' (63). This is a conservative setting for clients that want to keep communication restricted to 7-bit printable text.
  - `LOCALE`
    All characters will be handled "as-is". It is the responsibility of the client to ensure that the SAVAPI Service will be able to correctly interpret the text (for example, with matching locale settings between client, service and file system).

> **Note**
> 0-value characters will not be handled correctly in this mode. Encodings such as UTF-16LE or UTF-32LE are not supported by the SAVAPI TCP/IP protocol.

  - `UTF-8`
    All texts must be correctly encoded as UTF-8. Invalid encodings will result in error responses.

  Default value: `LOCALE`.

> **Note**
> If invalid text is received, the SAVAPI Service will output an error (whenever this is possible) or drop the connection.

- **PoolScanners**

  `PoolScanners <no_of_workers>`

  Specifies the number of SAVAPI workers. SAVAPI will start with the specified number of workers.

SAVAPI will not accept an infinite number of scanners. The maximum accepted number is 300.

Available values: 1 - 300

Default value: 24

- **User**

  `User <user>`

  Specifies the SAVAPI Service credentials.

  SAVAPI Service will start and will drop its credentials, as soon a possible, to the specified user.

  Available values: user name.

  Default value: there is no default value for this option. The service will run with the user and group it was started with (if nothing is specified in the configuration file).

  Note
  This option is available only on UNIX systems.

- **Group**

  `Group <group>`

  Specifies the SAVAPI Service credentials.

  SAVAPI Service will start and will drop its credentials, as soon a possible, to the specified group.

  Available values: group name.

  Default value: there is no default value for this option. The service will run with the user and group it was started with (if nothing is specified in the configuration file).

  Note
  This option is available only on UNIX systems.

- **PidDir**

  `PidDir </path/to/the/pid/dir>`

  Specifies the SAVAPI Service PID file location.

  Available values: only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

  Default value: `/var/tmp`.

  Note
  This option is available only on UNIX systems

- **KeyDir**

  `KeyDir </path/to/the/key/dir>`

  Specifies the location of the SAVAPI license key file(s).

  Available values: only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

  Default value: the location of the SAVAPI Service binary.

- **AveDir**

  `AveDir </path/to/the/ave/dir>`

  Specifies the location of the engine files.

  Available values: only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

  Default value: the location of the SAVAPI Service binary.

- **VdfDir**

  `VdfDir </path/to/the/vdf/dir>`

  Specifies the location of the VDF files.

  Available options: only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

  Default value: the location of the SAVAPI Service binary.

- **AttachToGuard**

  `AttachToGuard <0|1>`

  Windows: The SAVAPI Service will register with the Avira Realtime Protection (Guard) service, to prevent the Guard service from scanning the files that are scanned by the SAVAPI Service.

  UNIX: The SAVAPI daemon will register with the dazuko module, to allow the SAVAPI Service to scan files.

  Available values: 0 (disabled) or 1 (enabled).

  Default value: 1 (enabled).

- **SocketPermissions**

  `SocketPermissions <str>`

  Specifies the permissions for the UNIX local socket.

  The socket file is created with the same permissions as the running SAVAPI Service.

Note
The `user` and `group` options are obsolete and will be removed.

Connection Related Options

- **ListenAddress**

  Specifies the SAVAPI Service listening interface. The host can be either an IPv4 address or a hostname.

  Syntax for TCP connections:

  ```
  ListenAddress <inet:port[@host]>
  ```

  `inet` can be either a host name or an IP address, in any accepted format (dotted or not): decimal, hexadecimal, octal.

  Syntax for UNIX local sockets connections:

  ```
  ListenAddress <unix:/path/to/AF_UNIX/socket>
  ```

  Available values: listening interface location.

  Examples:

  Bind to TCP port 3333 on local host:
  `ListenAddress=inet:3333`

  Bind to TCP port 4444 on loop back interface:
  `ListenAddress=inet:4444@127.0.0.1`

  Bind to TCP port 5555 on public interface):
  `ListenAddress=inet:5555@testcomputer`

  Bind to UNIX local socket "`/var/tmp/.savapi3`":
  `ListenAddress=unix:/var/tmp/.savapi3`

  Default value: '`/var/tmp/.savapi3`' on UNIX. There is no default value on Windows.

- **ConnectionTimeout**

  ```
  ConnectionTimeout <value-in-sec>
  ```

  Specifies the timeout in seconds for the worker connection. A value of 0 means no timeout.

  Available values: 0 – INT32_MAX.

  Default value: 60.

  Available only in daemon/service mode.

Scan Related Options

- **ScanTemp**

  ```
  ScanTemp </path/to/temporary/folder>
  ```

  Sets the temporary folder used for scan related operations. SAVAPI may also use other temporary folders for non-scan operations.

  Available options: only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

Default value: the default temporary folder of the Operating System.

- **ArchiveScan**

  `ArchiveScan <value>`

  Activates archive detection and scanning.

  Available options: 0 (disabled) or 1 (enabled).

  Default value: 0.

- **ArchiveMaxSize**

  `ArchiveMaxSize <size>`

  Sets the maximum allowed size for any file within an archive, mailbox or email.

  Available values: file sizes up to `INT64_MAX` bytes.

  Default value: 1073741824

  A value of "0" means the maximum allowed value (`INT64_MAX` bytes).

  `<size>` can include K, M or G as a label. Otherwise the number is assumed to be in bytes. Examples: "100M" or "32K". (1K = 1024 bytes. 1M = 1024^2 bytes. 1G = 1024^3 bytes.)

  This setting is ignored if `ArchiveScan, MailboxScan` and `MimeScan` are all disabled.

- **ArchiveMaxRec**

  `ArchiveMaxRec <recursion-level>`

  Sets the maximum allowed recursion within an archive, mailbox or email.

  Available values: 0 – 1000.

  Default value: 200.

  This option is limited to 1000 recursion levels.

  A value of "0" means the maximum allowed value (1000 recursion levels).

  This setting is ignored if `ArchiveScan`, `MailboxScan` and `MimeScan` are all disabled.

- **ArchiveMaxRatio**

  `ArchiveMaxRatio <expansion-factor>`

  Sets the maximum allowed decompressing-ratio within an archive, mailbox or email.

  Available values: 0 – `INT32_MAX`.

  Default value: 150.

  A value of "0" means the maximum allowed value (`INT32_MAX`).

  This setting is ignored if `ArchiveScan`, `MailboxScan` and `MimeScan` are all disabled.

- **ArchiveMaxCount**

  `ArchiveMaxCount <count>`

  Sets the maximum allowed number of files within an archive, mailbox or email.

  Available values: 0 - `INT64_MAX`

  Default value: 0

  A value of "0" means the maximum allowed value (`INT64_MAX`).

  This setting is ignored if `ArchiveScan`, `MailboxScan` and `MimeScan` are all disabled.

- **MailboxScan**

  `MailboxScan <0|1>`

  Activates detection and scanning of mailboxes.

  Available values: 0 (disabled) or 1 (enabled)

  Default value: 0

- **HeurMacro**

  `HeurMacro <0|1>`

  Activates heuristic macro detection.

  Available options: 0 (disabled) or 1 (enabled)

  Default value: 1

- **HeurLevel**

  `HeurLevel <0-x>`

  Sets the heuristic level for the engine.

  Available values:

  0 - disable heuristic detection.

  1 - lazy heuristic detection. This is the lowest possible mode. Detection is not very good, but the false positives number will be low.

  2 - normal heuristic detection. This is the recommended heuristic detection.

  3 - high heuristic detection. This is the highest possible mode, but it will also increase the number of false positives.

  Default value: 2

- **DetectAdspy**

  `DetectAdspy <0|1>`

  Activates detection for software that displays advertising pop-ups or sends user-specific data to third parties without the user's consent and might therefore be unwanted.

  Available values: 0 (disabled) or 1 (enabled).

Default value: 1.

- **DetectAppl**

  `DetectAppl <0|1>`

  Activates detection for applications of uncertain origin or which might be hazardous to use.

  Available values: 0 (disabled) or 1 (enabled).

  Default value: 0.

- **DetectBdc**

  `DetectBdc <0|1>`

  Activates detection for Backdoor-Client programs. Such programs are used to spy out or change data on a computer.

  Available values: 0 (disabled) or 1 (enabled).

  Default value: 1.

- **DetectDial**

  `DetectDial <0|1>`

  Activates detection for Dial-Up programs, that charge for a connection fee.

  Available values: 0 (disabled) or 1 (enabled).

  Default value: 1.

- **DetectGame**

  `DetectGame <0|1>`

  Activates game detection.

  Available values: 0 (disabled) or 1 (enabled).

  Default value: 0

- **DetectHiddenExt**

  `DetectHiddenExt <0|1>`

  Activates the detection of hidden file extensions. The concerning file contains an executable, which is disguised by a harmless file extension.

  Available values: 0 (disabled) or 1 (enabled).

  Default value: 1.

- **DetectJoke**

  `DetectJoke <0|1>`

  Activates the detection of joke programs.

Available values: 0 (disabled) or 1 (enabled).

Default value: 0

- **DetectPck**

  `DetectPck <0|1>`

  Activates the detection of runtime compression tools.

  Available values: 0 (disabled) or 1 (enabled).

  Default value: 0.

- **DetectPhish**

  `DetectPhish <0|1>`

  Activates Phishing detection.

  Available values: 0 (disabled) or 1 (enabled).

  Default value: 1.

- **DetectSpr**

  `DetectSpr <0|1>`

  Activates the detection of programs that violate the private domain (Security Privacy Risk). This concerns software that may be able to compromise the security of your system, initiate unwanted program activities, damage your privacy or spy on your user behavior and could therefore be unwanted.

  Available values: 0 (disabled) or 1 (enabled).

  Default value: 0.

- **DetectAllTypes**

  `DetectAllTypes <0|1>`

  Activates/ deactivates all detection types: `DetectAdspy, DetectAppl, DetectBdc, DetectDial, DetectGame, DetectHiddenExt, DetectJoke, DetectPck, DetectPhish, DetectSpr`.

  Available values: 0 (all disabled) or 1 (all enabled).

  Default value: there is no default value for this option.

Note
If `DetectAllTypes` is active, it will override the settings for all the other `Detect`-options.

- **ScanTimeout**

  `ScanTimeout <time-in-seconds>`

  Sets the maximum number of seconds allowed to scan a file before aborting.

Available values: 0 - 86400

Default value: 0 (disabled).

- **Repair**

  `Repair <0|1>`

  Activates the repairing of infected files. The actual repairing occurs during the "`SCAN`" request.

  Available options: 0 (disabled) or 1 (enabled).

  Default value: 0.

- **SavapiNotifyRepair**

  `SavapiNotifyRepair <0|1>`

  Activates notification for a repairable infected file.

  Available options: 0 (disabled) or 1 (enabled).

  Default value: 0.

- **SavapiNotifyOffice**

  `SavapiNotifyOffice <0|1>`

  Activates the detection of Office documents.

  Available options: 0 (disabled) or 1 (enabled).

  Default value: 0.

- **SavapiNotifyOfficeMacro**

  `SavapiNotifyOfficeMacro <0|1>`

  Activates the detection of macros in Office documents.

  Available options: 0 (disabled) or 1 (enabled).

  Default value: 0

- **SavapiNotifyAlertURL**

  `SavapiNotifyAlertURL <0|1>`

  Activates the notification with the Avira virus description URL.

  Available options: 0 (disabled) or 1 (enabled).

  Default value: 0.

- **SavapiNotifyIframes**

  `SavapiNotifyIframes <0|1>`

  Activates the Iframes detection.

  Available options: 0 (disabled) or 1 (enabled).

Default value: 0.

- **AlertUrl**

  `AlertUrl <alert_url>`

  Specifies the Avira virus description URL.

  Available values: virus description URL. The URL must contain the string "`?q=%1`" for validation purposes. The keyword '`%1`' passes the virus name.

  Default value: `http://www.avira.com/en/threats?q=%1`

- **ScanMode**

  `ScanMode <SMART | ALL | EXTLIST>`

  Sets the scanning method.

  Available options:

  - `SMART`
    The files scanned for malware are chosen by SAVAPI. The choice is made based on the file's content. This is the recommended setting.
  - `ALL`
    All files are scanned for malware, no matter their content or extension.
  - `EXTLIST`
    Only files provided by the user, that mach a list of specific extensions, are scanned for malware content. Files inside archives are not filtered by extension.

  Default value: `SMART`

- **ReportEncryptedMime**

  `ReportEncryptedMime <0|1>`

  Activates reporting of encrypted mime emails.

  Available options: 0 (disabled) or 1 (enabled).

  Default value: 0

  Note
  This setting is ignored if `ArchiveScan`, `MailboxScan` and `MimeScan` are all disabled.

- **MimeScan**

  `MimeScan <0|1>`

  Activates detection and scanning of emails.

  Available values: 0 (disabled) or 1 (enabled)

  Default value: 1

Logging Related Options

- **LogFileName**

  `LogFileName </path/to/log/file>`

  Specifies the log file location and name. If this parameter is set, then file logging is activated.

  Available options: only absolute paths are accepted; relative paths will not be accepted and the service will exit with an error.

  Default value: none.

- **ReportLevel**

  `ReportLevel <0-3>`

  Specifies the log verbosity level.

  Available values:

  0 Log errors

  1 Log errors and alerts

  2 Log errors, alerts, warnings and info

  3 Log errors, alerts, warnings, info and debug messages

  Default value: 0

Note
"alerts" record information about potential malicious code.

- **LogRotate**

  `LogRotate <0|1>`

  Enables log rotation.

  Available options: 0 (disabled) or 1 (enabled).

  Default value: 0 (disabled) on UNIX; 1 (enabled) on Windows.

  Only 10 log files will be created during log rotation. Older log files will be deleted after the limit is reached and new log files will be created. See `LogFileSize` for more details.

- **LogFileSize**

  `LogFileSize <size>`

  Sets the maximum allowed size for the log file.

  Available values: file size up to `INT64_MAX` bytes.

  Default value: 2M.

  If log rotation is enabled (`LogRotate=1`), the log file is closed when the maximum size is reached; a new file is created and used for logging (i.e. the file "log" is renamed to "*log.001*", and the logging will continue to the new "*log*" file). If rotation is disabled (`LogRotate=0`), no action is performed when maximum size is reached, meaning that the logging will stop.

`Size` can include K, M, or G as a label. Otherwise the number is assumed to be in bytes. Examples: "100M" or "32K". (1K = 1024 bytes. 1M = 1024^2 bytes. 1G = 1024^3 bytes.)

- **LogTemplate**

  ```
  LogTemplate = ${DAY}/${MONTH}/${YEAR} ${HOUR}:${MINUTE}:${SECOND}
  ${HOST} ${PROGRAM}[${PID}]: ${SOURCE}: ${LEVEL}: ${MSG}
  ```

  Enables the templates for log messages.

  Available templates are:

  - `SOURCE`,
  - `LEVEL`,
  - `YEAR`,
  - `MONTH`,
  - `DAY`,
  - `HOUR`,
  - `MINUTE`,
  - `SECOND`,
  - `WEEKDAY`,
  - `TZOFFSET`,
  - `TZ`,
  - `HOST`,
  - `FULLHOST`,
  - `PROGRAM`,
  - `PID`,
  - `MSG`,
  - `WINDATE`,
  - `WINTIME`

  Default value: "`${DAY}/${MONTH}/${YEAR} ${HOUR}:${MINUTE}:${SECOND} ${HOST} ${PROGRAM}[${PID}]: ${SOURCE}: ${LEVEL}: ${MSG}`".

- **SyslogFacility**

  ```
  SyslogFacility <value>
  ```

  Specified the facility for the UNIX default system logger.

  Available values: desired system logger facility.

Default value: "`user`".

> **Note**
> This option is available only on UNIX systems.

- **DisableSystemLogger**

  ```
  DisableSystemLogger <0|1>
  ```

  Activates/ deactivates the default operating system logger.

  Available options: 0 (system logger enabled) or 1 (system logger disabled).

  Default value: 0 (system logger enabled).

3.3.3 SAVAPI Service protocol

Once a TCP/IP connection is established, the SAVAPI TCP/IP protocol can be used. The protocol specifies exchange of commands and information using synchronous communication in the form of messages. The messages consist in text lines (ASCII-PRINT, LOCALE, or UTF-8) followed by a new line.

Synchronous communication proceeds by sending a request and receiving a response. Responses include status codes to help classify the response. Some status codes imply that other responses are guaranteed to follow (thus informing the application that it must not yet send another request).

The general form of messages is as follows:

- request:
  `<command> <data>\n` (for UNIX)
  `<command> <data>\r\n` (for Windows)
- response:
  `<status-code> <data>\n` (for UNIX)
  `<status-code> <data>\r\n` (for Windows)

The SAVAPI Service initiates the communication by sending the following line:

```
100 SAVAPI:3.1
```

where '1' (in '3.1') is the minor version number of the SAVAPI protocol. The minor version number of the protocol is to be changed for extensions in the protocol.

Any 3.x version of the SAVAPI protocol must be 100% backwards compatible to a lesser 3.x version. If any incompatibilities should be introduced, the major version number ('3') of the protocol must be changed.

> **Note**
> '3.1' is the protocol version number, not a version number for the SAVAPI Service binaries themselves.

The initial SAVAPI "greeting" uses <10> as its new line.

Response Status Codes

The following table contains a list of status codes that will be returned after various requests.

Not all status codes apply to all requests. (Non-terminal responses imply that more responses are guaranteed to follow.)

Code	Definition	Type
100	Information	Typically as a response to a request
199	"Pong" response with optional ping-text	Terminal
200	File was not an archive, no alert found	Terminal
210	File was an archive, no alert found	Terminal
220	A connection timeout occurred	Terminal
310	Alert found	Non-terminal
319	Scan finished, alert found	Terminal
350	Error occurred	Terminal
401	Low-level alert information	Non-terminal
420	Repairable alert found (the alert itself will follow)	Non-terminal
421	Office document found	Non-terminal
422	Office document with macros found	Non-terminal
430	Alert URL	Non-terminal
440	IFRAME	Non-terminal
450	Plugin response	Non-terminal
499	Information	Non-terminal

Note
Code "*300: File was not an archive, alert found*" is obsolete.

The new engine can detect multiple malware in a regular file – non-archive file, without embedded objects (PDF, chm, etc.).

The only way to report multiple infections in a file is to list all non-terminal alerts and then to report the end of the scan. This is happening only if there is at least one malware detection (this means "alert"). If the file doesn't contain malware, the normal code `200` is returned.

Requests

The following sections describe the various requests that are allowed with the SAVAPI protocol. Any non-specified requests will result in an error response "`350 <error-message>`".

All request descriptions are followed by a list of possible status codes that can be used in response messages.

> **Note**
> All requests apply to the current session, with the exception of `RELOAD`. For more details, see the `RELOAD` command description).

SAVAPI will consider all numeric values as decimal (base 10).

SET (write only)

"`SET`" requests are available to configure SAVAPI. Usually a "`SET`" request also has a "`GET`" request counterpart to retrieve current settings. However, the following commands do not have a "`GET`" counterpart and are therefore labelled as "write only".

- **SET PRODUCT <id>**

  Set the key-id that is required by the application. SAVAPI will check if the key-id is within the license and that it is not expired. If it is available and is valid, the application is free to use SAVAPI. If not, this command and all `SCAN` requests will result in an error response, for example: "350 operation not allowed (license restriction)".

  (100, 350)

GET (read only)

"`GET`" requests are available to retrieve current SAVAPI settings. The response to a "`GET`" request is in the form of:

```
100 <key>:<value>
```

This makes it possible to combine multiple "`GET`" requests into a single request:

```
GET <key1> <key2> <key3>
<key1>:<value1> <key2>:<value2> <key3>:<value3>
```

> **Note**
> If values include ':' (colon) or ' ' (space) characters, the response could be ambiguous.

Usually a "`GET`" request also has a "`SET`" request counterpart to configure SAVAPI. However, the following commands do not have a "`SET`" counterpart and are therefore labelled as "read only".

- **GET SAVAPI**

  Retrieve SAVAPI protocol version number.
  (100, 350)

- **GET AVE**

  Retrieve engine version number.
  (100, 350)

- **GET VDF**

  Retrieve VDF set version number.
  (100, 350)

- **GET PID**

  Retrieve the process-id for the SAVAPI process that is currently handling the connection.
  (100, 350)

- **GET EXPIRE**

  Retrieve the expiration date of the SAVAPI license. The date is in the form of YYYYMMDD.
  (100, 350)

- **GET VDF_SIGCOUNT**

  Retrieve the number of signatures in the VDF set.
  (100, 350)

- **GET SELECTABLE_DETECT**

  Retrieve the various types that can be detected (and dynamically turned on/off). The types are returned as a comma separated list.
  (100, 350)

- **GET DESCR_DETECT_<type>**

  Retrieve the English description for the given type. `<type>` is one of the types returned by "`GET SELECTABLE_DETECT`".
  (100, 350)

GET/SET (read/write)

"`SET`" requests are available to configure SAVAPI. For the following requests, a "`GET`" counterpart is also available and these are therefore labelled as "read/write". The "`GET`" response will return the same data that is provided with the "`SET`" request (the representation of the data may be different. For example, a "`SET`" request with "10K" will lead to a "`GET`" response with "10240".)

> **Note**
> Almost all `SET` commands exist in the same format in the configuration file.
> Exception: `SET CONF`.

For more details, see 3.3.2 SAVAPI configuration file options.

- **CWD <directory>**

  Set the current working directory for SAVAPI. Thus you don't need to specify full paths in file names anymore.

  Default value: the SAVAPI Service binary location.
  (100, 350)

- **CONF <configuration-file>**

  Specify the configuration file used. The configuration file will be (re-)read as part of this request. Global SAVAPI configuration parameters will be ignored and only worker-specific parameters will be read from the original configuration file. There is no default value.
  (100, 350)

  The following parameters will be ignored when a `SET CONF` command is executed: `ListenAddress`, `AttachToGuard`, `LogRotate`, `LoadPlugins`, `SocketPermissions`, `User`, `Group`, `SyslogFacility`, `IncludeConfig`, `AveDir`, `KeyDir`, `VdfDir`, `PidDir`, `LogFileName`, `LogTemplate`, `LogFileSize`, `ReportLevel`, `PoolScanners`, `PoolConnections`.

- **ARCHIVE_SCAN <0|1>**

  Activate archive detection and scanning.

  Default value: `0`.
  (100, 350)

- **ARCHIVE_MAX_SIZE <size>**

  Set the maximum allowed size for any file within an archive, mailbox and email. A value of "0" means the maximum allowed value. A size can include K, M or G as a label. Otherwise the number is assumed to be in bytes. Examples: "100M" or "32K". (1K = 1024 bytes. 1M = 1024^2 bytes. 1G = 1024^3 bytes.)
  This setting is ignored, if `ARCHIVE_SCAN`, `MAILBOX_SCAN` and `MIME_SCAN` are all disabled.
  Default value: `1073741824`.
  (100, 350)

- **ARCHIVE_MAX_REC <recursion-level>**

  Set the maximum allowed recursion within an archive, mailbox and email. This option is limited to 1000 recursion levels. A value of "0" means the maximum allowed value (1000 recursion levels). This setting is ignored, if `ARCHIVE_SCAN`, `MAILBOX_SCAN` and

`MIME_SCAN` are all disabled.
Default value: `200`.
(100, 350)

- **ARCHIVE_MAX_RATIO <expansion-factor>**

  Set the maximum allowed decompressing-ratio within an archive, mailbox and email. A value of "0" means the maximum allowed value. This setting is ignored, if `ARCHIVE_SCAN`, `MAILBOX_SCAN` and `MIME_SCAN` are all disabled.
  Default value: `150`.
  (100, 350)

- **ARCHIVE_MAX_COUNT <count>**

  Set the maximum allowed number of files within an archive, mailbox and email. A value of "0" means the maximum allowed value. This setting is ignored, if `ARCHIVE_SCAN`, `MAILBOX_SCAN` and `MIME_SCAN` are all disabled.
  Default value: `9223372036854775807`.
  (100, 350)

- **MAILBOX_SCAN <0|1>**

  Activate detection and scanning of mailboxes.
  Default value: 0.
  (100, 350)

- **HEUR_MACRO <0|1>**

  Activate heuristic macro detection.
  Default value: 1.
  (100, 350)

- **HEUR_LEVEL <0-3>**

  Set the heuristic level for the engine.

  Available values:

  0 - disable heuristic detection.

  1 - lazy heuristic detection. This is the lowest possible mode. Detection is not very good, but the false positives number will be low.

  2 - normal heuristic detection. This is the recommended heuristic detection.

  3 - high heuristic detection. This is the highest possible mode, but it will also increase the number of false positives.

  Default value: 2.
  (100, 350)

- **DETECT_<type> <0|1>**

  Activate detection of various other types. `<type>` is one of the types returned by "`GET SELECTABLE_DETECT`". An additional pseudo-type (`ALLTYPES`) is also available to generally represent all types. (`ALLTYPES` is only available for "`SET`" requests, and not

available for "`GET`" requests.)
Default values:

- `DETECT_ADSPY 1`
- `DETECT_APPL 0`
- `DETECT_BDC 1`
- `DETECT_DIAL 1`
- `DETECT_GAME 0`
- `DETECT_HIDDENEXT 1`
- `DETECT_JOKE 0`
- `DETECT_PCK 0`
- `DETECT_PHISH 1`
- `DETECT_SPR 0`

(100, 350)

- **SCAN_TEMP <directory>**

  Set the temporary directory used for scanning files. SAVAPI may use other temporary directories for files that are not being scanned. These other directories can be specified with command-line arguments or in a configuration file.
  (100, 350)

- **SCAN_TIMEOUT <time-in-seconds>**

  Set the maximum number of seconds allowed to scan a file before aborting.
  Default value: 0 (disabled).
  (100, 350)

- **TEXT_MODE <ASCII-PRINT|LOCALE|UTF-8>**

  Specify the character encoding used to read requests and post results. This is covered in details later in the section "`TEXT SUPPORT`".
  Default: `LOCALE`.
  (100, 350)

- **REPAIR <0|1>**

  Activate repairing of infected files. The actual repairing occurs during the "`SCAN`" request.
  Default value: 0 (disabled).
  (100, 350)

- **SAVAPI_NOTIFY_REPAIR <0|1>**

  Activate notification if an infected file can be repaired. This will activate the usage of 420 status codes.
  Default value: 0 (disabled).
  (100, 350)

- **SAVAPI_NOTIFY_OFFICE <0|1>**

  Activate detection of office documents. This will activate the usage of 421 status codes.
  Default value: 0 (disabled).
  (100, 350)

- **SAVAPI_NOTIFY_OFFICE_MACRO <0|1>**

  Activate detection of macros within office documents. This will activate the usage of 422 status codes.
  Default value: 0 (disabled).
  (100, 350)

- **SAVAPI_NOTIFY_ALERTURL <0|1>**

  Display alert URLs for detected alerts. This will activate the usage of 430 status codes.
  Default value: 0 (disabled).
  (100, 350)

- **IFRAMES_URL <0|1>**

  Activates IFRAME detection. IFRAMEs can be used to hide malware in a web page. This will activate the usage of 440 status codes.
  Default value: 0 (disabled).
  (100, 350)

- **REPORT_ENCRYPTED_MIME <0|1>**

  Activates reporting of encrypted mime emails. This setting is ignored if `ARCHIVE_SCAN`, `MAILBOX_SCAN` and `MIME_SCAN` are all disabled.
  Default value: 0 (disabled).
  (100, 350)

- **SCAN_MODE <SMART|ALL|EXTLIST>**

  Set the scanning method. Possible values are:

  - `ALL` – All files are scanned for malware, no matter their content or extension.
  - `EXTLIST` – Only files provided by the user, that mach a list of specific extensions, are scanned for malware content. Files inside archives are not filtered by extension.
  - `SMART` – The files scanned for malware are chosen by SAVAPI. The choice is made based on the file's content. This is the recommended (default) setting.

  (100, 350)

- **MIME_SCAN <0|1>**

  Activates or deactivates the detection and scanning of emails. By default `MIME_SCAN` is enabled. It is recommended to always keep this option enabled.

  If `MAILBOX_SCAN` is enabled, emails contained in mailboxes are always scanned (no matter if `MIME_SCAN` is enabled or not).
  Default value: 1 (enabled).
  (100, 350)

- **ENCODE_FILENAMES <0|1>**

  Activates the filename encoding feature in the SAVAPI protocol. When activated, the path and filename arguments of scanning commands must be encoded in their hexadecimal representation (Every byte of the path and filename must be represented by it's hexadecimal value, with two hexadecimal digits. For example, *savapi.exe* in ASCII encoding: *7361766170692e657865*).

  The paths and filenames in the answers received from SAVAPI will also be encoded in the same way.

  Default value: 0 (disabled)
  (100, 350)

> **Note**
> This option should be used for sending paths and filenames that contain control characters or begin with white-spaces through the socket.

The SCAN command

The most important request (and most complex) is the `SCAN` request. This request invokes the engine for a specified file.

SCAN command syntax:

```
SCAN <file name>
```

SAVAPI scans the file with the specified name.

SCAN command responses

Depending on the SAVAPI settings and the contents of the file, various responses can be returned.

- **200 <scan-keyword> [<scan-keyword> ...]**

  This response means that the file does not contain any alerts. Various keywords can be returned to communicate additional information about the scan. Although no alert was found in the file, depending on the keywords, the file may still contain an alert (i.e.: if the `INCOMPLETE` keyword appears it means that only a part of the file was scanned and only that part of the file is clean. The rest of the file might contain malware). This is a terminal response, meaning that the application is free to make another request.

- **210 <scan-keyword> [<scan-keyword> ...]**

  This response is identical in format to "200". It means that the file does not contain any alerts. However, it has the additional meaning that the file was detected as an archive and its contents were also scanned. Since an archive has many special attributes (multiple files, compression, recursion) there are various keywords that can be returned. Although no alert was found in the archive, depending on the keywords, the file may still contain an alert (i.e.: if the `HIT_MAX_REC` keyword appears, it means that only a part of the archive was scanned and only that part of the archive is clean. The rest of the archive might contain malware). This is a terminal response, meaning that the application is free to make another request.

- **220 timeout reached while waiting commands**

  This message means that the connection was idle for too long and it was closed by the server. For more details regarding the connection timeout, see Connection Related Options.

- **310 [fileA-in-archive[--> fileB-in-fileA] <<<]alert-name ; type ; english-text-message**

  This response means that the file contains an alert. However, it has the additional meaning that the file was detected as an archive and its contents were also scanned. Since archives contain files (and these files may also be archives), a type of archive recursion is present. This recursion is represented with an additional separator " --> " (space minus minus greater-than space). This separator is followed by the name of the file in the next level of recursion. (Note that the file name provided with the "`SCAN`" request is not included.) After all recursion has been represented, a second separator " <<< " is used (space less-than less-than less-than space). After this separator, the alert name, alert type and an English alert description are provided. These fields are separated by " ; " (space semi-colon space).

  The application is responsible for parsing this output. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request. (Note that the " <<< " separator may not appear, if the archive itself, rather than its contents, triggers the alert.)

  The "310" response should not appear multiple times for a single "end file" within an archive, i.e. it must not be possible to generate more "310" responses than the number of files to scan.

- **319 <scan-keyword> [<scan-keyword> ...]**

  This response means that the scanning of an alert-containing object is complete. This is a terminal response, meaning that the application is free to make another request. This response will come after one or more "310" responses. As with response "210", multiple keywords are returned to describe information about the scan.

- **420 [fileA-in-archive[--> fileB-in-fileA] <<<]alert-name ; type ; english-text-message**

  The infected file is repairable. The format of this response is identical to "310". This response will be followed (although not necessarily directly) by a "310" response with the same format. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request. Note: To repair the file, the repair mode must be activated and a scan performed again. Files that have been successfully repaired will no longer cause "310" or "420" responses.

- **421 [fileA-in-archive[--> fileB-in-fileA] <<<]office-type**

  An office document has been found. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request.

- **422 [fileA-in-archive[--> fileB-in-fileA] <<<]macro-name**

  An office document containing macros has been found. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request.

- **430 [fileA-in-archive[--> fileB-in-fileA] <<<]URL**

  A URL that will lead to detailed information about an alert found. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request.

- **440 [fileA-in-archive[--> fileB-in-fileA] <<<] IFRAME iframe_type -> URL**

  A possible malicious IFRAME was detected. Possible values for '`iframe_type`' are: `extrasmall`, `invisible`, `malicious`, `oddpos`. However, applications should be ready to handle other IFRAME types also.

Note
Code "*300: File was not an archive, alert found*" is obsolete.

Here is a list of scan keywords that could appear with "200", "210" or "319" responses.

Note
Please note that scan keywords may change between releases and the application must be prepared to handle new scan keywords.

Scan-keywords	Description
`ABORTED`	scan was aborted by signal (200, 210, 319)
`ENCRYPTED`	contains encrypted contents (210, 319)
`ENCRYPTED_MIME`	contains encrypted mime emails (210, 319)
`HIT_MAX_COUNT`	maximum number of files reached (210, 319)
`HIT_MAX_RATIO`	maximum extraction ratio reached (210, 319)
`HIT_MAX_REC`	maximum recursion limit reached (210, 319)
`HIT_MAX_SIZE`	maximum extraction size reached (210, 319)
`INCOMPLETE`	not completely scanned (210, 319)
`MEMORY_LIMIT`	internal memory limit reached (210)
`OK`	no problems detected (200, 210, 319)
`PARTIAL`	is part of a multi-volume archive (210, 319)
`PROC_ERROR`	processing archive (210, 319)

Scan-keywords	Description
`TIMEOUT`	scan timed out (200, 210, 319)
`UNSUPPORTED`	archive format detected (210, 319)

If the `ENCODE_FILENAMES` option is activated, the path and filename arguments of the `SCAN` command must be encoded in their hexadecimal representation. (Every byte of the path and filename must be represented by it's hexadecimal value, with two hexadecimal digits. For example, *savapi.exe* in ASCII encoding: *7361766170692e657865.*)

The paths and filenames in the answers received from SAVAPI will also be encoded in the same way.

Note
`ENCODE_FILENAMES` should be used for sending paths and filenames that contain control characters or begin with white-spaces through the socket.

Update Support

Internet updates will occur in a background process. Updates may require the SAVAPI Service to restart without informing the connected applications. Applications using SAVAPI must be able to handle unexpected closing of the TCP/ IP socket and try to re-establish a connection before interpreting the event as an error.

The *avupdate* component is a separate binary and it is no longer managed by SAVAPI. See 7. Updating SAVAPI - page 60.

Other commands

The following commands are also available.

- **QUIT**

  Gracefully disconnects from SAVAPI. This signals SAVAPI that the application is finished. Rather than providing a response, SAVAPI will close the connection. The SAVAPI Service will continue to run. (closed socket)

- **RESET**

  Commands SAVAPI to return to the same configuration state as upon the initial connection.

Note
The configuration file will not be re-read.

(100, 350)

- **RELOAD**

  Commands SAVAPI to reload its original configuration file: the configuration file specified when the service is started (on non-Windows systems) or registered (on Windows systems). Global SAVAPI configuration parameters will be ignored and only worker-specific parameters will be read from the original configuration file. This will affect only the current session and the newly opened sessions. The existing sessions will not be affected by this command.
  (100, 350)

> **Note**
> The following parameters will be ignored when a `RELOAD` command is executed: `ListenAddress`, `AttachToGuard`, `LogRotate`, `DisableSystemLogger`, `LoadPlugins`, `SocketPermissions`, `User`, `Group`, `SyslogFacility`, `IncludeConfig`, `AveDir`, `KeyDir`, `VdfDir`, `PidDir`, `LogFileName`, `LogTemplate`, `LogFileSize`, `ReportLevel`, `PoolScanners`, `PoolConnections`.

- **SHUTDOWN**

  ~~Commands SAVAPI Service to stop. SAVAPI will close the connection, without providing a response. This could cause unexpected socket-close events for other connected applications. (closed socket)~~

  ~~On Windows systems, because the SAVAPI Service is configured by default to restart as a failure action, the service will be restarted by the Service Control Manager after the `SHUTDOWN` command is received. The service can be stopped either with "`SAVAPI3 --stop`" command or "`net stop savapi3`" command~~.

> **Note**
> This command is marked as obsolete in SAVAPI version 3.3. It drops the connection and aborts any processing, but does not stop the service. Starting with the next SAVAPI release, the command will be removed and the service will reply with an error.

- **PING [<ping-text>]**

  Test if the SAVAPI daemon is alive. The optional `ping-text` can be a maximum of 128 bytes in length. This same text will be returned in a 199 "pong" response. This command is also useful for synchronizing communication with SAVAPI. (199)

- **HELP [<command> [<command-arg>]]**

  Provides available commands, arguments, and descriptions. The help output is generated using multiple "499" responses. (100, 350)

Other responses

- **499 informational-text**

  A line of informational text. This response appears, for example, after the "`HELP`" request. This is NOT a terminal response, meaning that more responses are guaranteed and the application may not yet make another request.

Signals

SAVAPI also supports signals. These are commands that are sent asynchronously to SAVAPI as a trigger for certain actions. Signals look similar to requests but do not receive a response.

- **SCAN_ABORT**

  SAVAPI aborts a scan and adds the "`ABORTED`" scan keyword to the scan response. This signal is ignored, if SAVAPI is not currently scanning a file.

Text support

The SAVAPI protocol provides three modes for handling text. The modes can be set using the "`SET TEXT_MODE`" request. Modes are applied to both incoming and outgoing data. The modes are described below.

If invalid text is received, the SAVAPI Service will immediately drop the connection.

Since some special characters are used within the SAVAPI protocol itself, some bytes of a file name must be converted to '?' (63) in responses. The bytes of a file name that must be converted are 10, 13, 60, 62. (Note that this does not affect SAVAPI's ability to scan and detect alerts. It is simply a cosmetic change in a response to allow a client to reliably parse the response.)

- **ASCII-PRINT**

  Only printable characters of the ASCII set are considered valid characters. These include the characters 9 and 32-126. All other character values are converted to '?' (63). This is a conservative setting for clients that want to keep communication restricted to 7-bit printable text.

- **LOCALE**

  All characters will be handled "as-is". It is the responsibility of the client to ensure that the SAVAPI Service will be able to correctly interpret the text (for example, with matching locale settings between client, service, and file system).

  0-value characters will not be handled correctly in this mode. Encodings such as UTF-16LE or UTF-32LE are not supported by the SAVAPI TCP/IP protocol.

- **UTF-8**

  The entire text must be correctly encoded as UTF-8. Invalid encodings will result in error responses.

3.4 SAVAPI Service exit codes

The following table contains a list of exit codes, returned by SAVAPI Service when it completes its execution.

Exit code	Description
200	Program aborted, not enough memory available
201	Program aborted, invalid parameter
202	Program aborted, daemon not/already initialized
203	Program aborted, conversion error
204	Error parsing the command line arguments
205	Error parsing the configuration file
206	Invalid port specified (Obsolete, not used anymore)
207	Invalid IP specified (Obsolete, not used anymore)
208	Cannot start on specified interface (invalid IP/port)
209	Error on loading vdf files
210	Problem when trying to start the engine
211	Program aborted because the self check failed
212	No valid license found (Obsolete, not used anymore)
213	Error when trying to start/stop the SAVAPI service
214	Program aborted because GET command failed
215	Program aborted because SET command failed
216	Error when trying to open/read/write a file
217	Error when trying to set uid/gid
218	SAVAPI service timeout
219	SAVAPI service not running
220	Failed to stop SAVAPI service
250	General daemon failure (specific information not available)

Note
Obsolete exit codes:

- *206: Invalid port specified*
- *207: Invalid IP specified*
- *212: No valid license found*

3.5 SAVAPI Service logging

3.5.1 Initialization

The logging in the SAVAPI Service is initialized when the process starts and, depending on the facility used, it can be divided in 3 types:

- **Console logging** – Cannot be configured or deactivated. It uses the STDOUT to report the results of a given command (e.g. `--version`, `--help`) and the STDERR to print the error reason (if any). For example: *"Validation failed for option 'config' with value 'inexistent_path'. Path does not exist."*
- **File logging** – Fully configurable and disabled by default. It can be enabled through the command line parameter:
  `--log-file=<log/file/path>`
  or by using the configuration file parameter:
  `LogFileName=<log/file/path>`.
- **System logging** (syslog daemon on UNIX, Event Logger on Windows) – Partially configurable and enabled by default. It can be enabled/disabled through the configuration file parameter `DisableSystemLogger=0/1`.

3.5.2 Configuration

File logging

The file logging is fully configurable, using the following options:

- **The log file location and name**
  It can be set with the command line option:
  `--log-file=<file/path>`
  or the configuration file option:
  `LogFileName=<file/path>`
  Because the file logging is disabled by default, the user also enables the logging by setting the location of the file.
  SAVAPI Service will try to open/create the file right after the privileges are dropped (i.e. the user&group are changed) and will return an error if the opening fails.
  If the given file is already present on the disk, SAVAPI Service will open it in appending mode, in order to write all messages at the end of the file. Else, if the file does not exist, it will be created by SAVAPI Service with read&write and read permissions for the configured user and configured group, respectively.

- **The logging level**
  The default level of logging is 0 and it can be changed with the configuration file option `ReportLevel=<level>`. Messages on different log levels can be distinguished by the level-specific tag name (e.g. the level `0` has the `ERROR` tag assigned). The available log levels are the following:
  - `0` – Only errors will be logged
    This level includes messages sent when the execution of SAVAPI Service is aborted (i.e. fatal errors during startup or running time). Additionally, SAVAPI Service uses this level in order to log the file-scanning errors (e.g encrypted file, non-writable scan-temp folder).
    The associated tag is: `ERROR`.
  - `1` – Only errors and alerts will be logged
    Setting this level will determine SAVAPI Service to also log the malware found – when a file is detected as infected, a message containing the file path and the malware details will be logged.
    The associated tag is: `ALERT`.
  - `2` - Errors, alerts and warnings
    This level adds the warning messages that do not affect the process condition (state). They are mainly used to point the user to a possible issue or inconvenience: the used VDFs are old, the SAVAPI Library is signed with developer key, etc.
    The associated tag is named: `WARNING`.
  - `3` - Errors, alerts, warnings, info and debug
    This is the most verbose level and adds the informational messages, used to broadly describe SAVAPI Service's actions (command line and configuration file parsing, socket creation, socket message interpretation, etc) and the debug messages – more detailed information about the performed actions.
    Here are 2 tags: one for the info logs, named `INFO`, and the other for the debug messages, named `DEBUG`.

> **Note**
> There are situations when SAVAPI Service is logging messages, regardless of the value of the `ReportLevel`.

  Currently, 2 messages are logged directly on `WARNING` level, in order to warn the user about an accidental stop, performed by another instance:
  - A running SAVAPI Service is stopped by another SAVAPI Service which starts:
    "*WARNING: Another running SAVAPI service instance detected on the specified interface. The existing instance will be replaced by the new one. SAVAPI service will stop*"
  - The SAVAPI Service startup detects another running instance:
    "*WARNING: Another running SAVAPI Service instance detected on the specified interface. The existing instance will be replaced by the new one.*"
- **The log template**
  Besides the message itself, SAVAPI Service can also add the time when the log is written in the file, the host-machine where the process is running, the PID of the process, etc. This extra information is possible due to the configuration file option `LogTemplate=<user/log/template>`, which contains some predefined macros,

automatically expanded by the SAVAPI Service's logger before writing the message in the file. The default log template is:
`"${DAY}/${MONTH}/${YEAR} ${HOUR}:${MINUTE}:${SECOND} ${HOST} ${PROGRAM}[${PID}]: ${SOURCE}: ${LEVEL}: ${MSG}"`
The available log macros are the following (macro X must be written as ${X} in order to be accepted):

- `SOURCE` – specifies the source of the message. Currently, there are 2 possible sources: "Supervisor" (the message is sent by the SAVAPI service's supervisor) and "Service" (the message is sent by the SAVAPI Service itself).
- `LEVEL` – will display the level of the message logged. The available levels are: "ERROR", "ALERT", "WARNING", "INFO" and "DEBUG".
- `YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, WEEKDAY, TZ` (the time zone abbreviation: EEST, CET , etc) and `TZOFFSET` (the time zone offset related to UTC/ GMT: +0300 for Romania, +0200 for Germany), – add the time-related information.
- `HOST` and `FULLHOST` – adds information about system's host name
- `PROGRAM` – adds the name of the binary: "*savapi*" on UNIX and "*savapi.exe*" on Windows
- `PID` – adds the process id
- `MSG` – adds the message itself to the file
- `WINDATE` and `WINTIME` – available only on Windows platforms; adds information about the local date and local time.

- **The log file size**
  By default, the size of the log file is unlimited and this may be an inconvenience when using a verbose level (i.e. 3), because the file size can grow to a large size very quickly. Therefore, SAVAPI Service offers the possibility to set the maximum size of the log file through the configuration parameters: `LogFileSize=<max-size>` and `LogRotate=<0|1>`. By default, log rotation is disabled on UNIX and enabled on Windows.
  If log rotation is activated (i.e. `LogRotate=1`), the log files will be rotated (i.e. file "*log*" is renamed to "*log.001*", file "*log.001*" is renamed to "*log.002*" and so on) when the maximum size is reached. SAVAPI Service keeps a maximum of 10 rotated files – from "*log.001*" to "*log.010*".
  If rotation is not active (`LogRotate=0`) SAVAPI Service will continue to write in the given file, regardless of the configured size.

Note
Control characters (range [1..31] in the ASCII table) will be replaced by the # character in the logfiles.

System logging

SAVAPI Service sends the same messages to the system logger, as for the file logging.

The system logging uses the **syslog** daemon on UNIX and the **Event Logger** on Windows.

The following configuration options are available:

- **Enabling/ Disabling**
  By default, the system logging is enabled on both platforms (UNIX and Windows). The user can enable/ disable it, using the configuration file parameter: `DisableSystemLogger=0|1`
- **The level of logging**
  The logging levels (and default) for system logging are the same as for file logging and they can be configured in the same way.
  Before sending the message to the system logger, SAVAPI Service maps the log level with the corresponding level from the system logger:
  - On UNIX (syslog daemon):
    `ERROR ? LOG_ERR, ALERT ? LOG_ALERT, WARNING ? LOG_WARNING, INFO ? LOG_INFO, DEBUG ? LOG_DEBUG` (check syslog's documentation for more information about its logging levels: http://en.wikipedia.org/wiki/Syslog)
  - On Windows (Event Logger):
    `ERROR ? EVENTLOG_ERROR_TYPE, ALERT and WARNING ? EVENTLOG_WARNING_TYPE, INFO and DEBUG ? EVENTLOG_INFORMATION_TYPE` (see Event Logger's documentation for more information about its logging levels).
- **The facility used** (only for UNIX)
  By default, the syslog facility used is "`user`". This can be changed with the configuration file parameter `SyslogFacility=<facility>`.
  The available syslog facilities are the following:
  - "`mail`",
  - "`auth`" (security/authorization messages),
  - "`authpriv`" (security/ authorization messages – private),
  - "`cron`" (clock daemon),
  - "`daemon`" (system daemons),
  - "`ftp`" (ftp daemon),
  - "`kern`" (kernel messages),
  - "`lpr`" (line printer subsystem),
  - "`news`" (network news subsystem),
  - "`syslog`" (messages generated internally by syslog daemon),
  - "`user`" (random user-level messages),
  - "`uucp`" (UUCP subsystem),
  - "`local0`" to "`local7`" (reserved for local use).

Recommendations

SAVAPI Service uses a synchronous logging mechanism, meaning that the thread (scanning instance) execution is interrupted until the message is written in the configured facilities. The interruption time depends on many factors: the file writing policy of the operating system (caching/ non-caching), the disk I/O performance, the number of threads/ processes writing to the same file, etc.

Therefore, SAVAPI Service may have serious performance penalties due to the heavy logging; Verbose log levels should be set only for debugging purposes.

The following logging configuration is recommended:

- Set a minimum log level: 0 or 1 (errors and alerts)
- Since SAVAPI Service also logs to the System Logger (which is a separate process meaning an asynchronous logging, i.e. less dead-time), you should enable the file logging only if it is really needed. The user can easily find the system logging messages.

Issues with the SAVAPI Protocol when using a telnet client

If the telnet client is used to communicate with the SAVAPI Service and UNICODE characters are sent, it may happen that the connection fails. This is a telnet issue: On certain operating systems, some telnet versions will not be able to display the UNICODE characters returned by SAVAPI and the connection appears to be hanging. When the socket traffic is verified with a tool like `strace`, it can be clearly seen that SAVAPI writes its answers correctly on the socket, but the telnet client does not write them back to the user interface.

4. SAVAPI Client Library

4.1 General description of SAVAPI Client Library

SAVAPI Client Library is a cross-platform library that allows the user to create network connections to the SAVAPI Service, configure the service options and scan files (found on disk or mapped in memory).

> **Note**
> The structure used for scanning objects from memory uses the type: "unsigned int" for the size of the object in memory. This limits the maximum size for a scanned object to approximately 4GB (2^32) on most of the 64-bit systems.

SAVAPI Client Library offers a cross-platform API, written in C language (the same as for the SAVAPI Library). The API allows the SAVAPI integrator to initialize the library, create an instance, register callback functions to monitor the processing progress, configure the instance settings, scan files, destroy the instance and uninitialize the library.

Basically, it does all the tasks related to network connection management:

- initialize the network socket
- configure and start the network socket
- verify that the network socket I/O operations were performed with success
- close the network socket and destroy all the related resources

The users can implement the other parts of their application, without worrying about the network connections management.

4.2 Integration of SAVAPI Client Library

The client-designed applications should (implicitly or explicitly) link to the SAVAPI Client Library and use the SAVAPI Client Library API to access the SAVAPI technology. The SAVAPI Client Library API allows the user to configure the scan engine and other SAVAPI options, to process files and to retrieve information about the processing status.

In order to use the SAVAPI Client Library, the SAVAPI Service has to be installed and started.

A very simple example of how one can use the SAVAPI Library is offered in the SAVAPI SDK kit. (You can refer to the SDK documentation under *doc/README.)*

4.3 Configuration of SAVAPI Client Library

The SAVAPI Client Library can be configured through the exposed API. In order to set an option, the customer just calls the `SAVAPI3_set()` function, with the appropriate parameters.

The SAVAPI Client Library API is the same as the SAVAPI Library API. For more details, see the documentation of SAVAPI Library API.

Note
The timeouts should be set according to the running system's load and performance. If the system's resources are low, exceeding timeouts can break the communication between SAVAPI Client Library and the SAVAPI Service. If the timeouts are infinite or too long, the client connection may block while waiting for a server response.

4.4 Logging in SAVAPI Client Library

The SAVAPI Client Library's logging mechanism is the same as the one in SAVAPI Library. See 2.4 SAVAPI Library logging.

5. Installation

You must have a valid Avira license in order to install and use the product.

5.1 Installation on Windows

The product does not require installation. Just unpack the corresponding archive in a folder and you can start using SAVAPI.

The SAVAPI Service must first be registered in the Windows Service Control Manager. This is done by adding the command line option `--install` to the command line used with the service.

In order to start on a Windows operating system, the SAVAPI Service must retrieve at least the interface it is supposed to start with, from the user. This can be done in two ways:

- Using the command line option `--tcp`. For example:
  ```
  savapi --tcp=9999 --install
  ```
- Using the configuration file option `ListenAddress` and passing the configuration file to the service, using the command line option `--config`. For example:
  ```
  savapi --config=c:/path/to/my_config_file --install
  ```
  where the file `my_config_file` should contain at least the following line:
  ```
  ListenAddress=inet:9999
  ```

For more details about the command line options and the configuration file options, please refer to the following chapters:

- 3.3.1 SAVAPI command line parameters
- 3.3.2 SAVAPI configuration file options

Note
On Windows systems, you might have to install the Microsoft Visual C++ 2008 SP1 Redistributable Package, corresponding to your system's architecture. See 1.2 System requirements.

5.2 Installation on UNIX

The product does not require installation. Just unpack the corresponding archive in a directory and you can start using SAVAPI.

Please copy the license key into the SAVAPI installation directory.

6. Licensing

Your company will receive one dedicated *.key* file, which can be delivered to all your customers together with your own application's key. Please note that you always have to deliver the *.key* file, otherwise SAVAPI will not scan.

This key is unique for you as a company. It relates (among other things) to the expiration date and the Product ID that you will receive from Avira (in the same time with the new key). Your application, which integrates SAVAPI (Library or Service), must send the product ID to SAVAPI, which uses it to initialize the engine. If the product ID and the key do not match, then the product does not scan.

The key you will receive from Avira represents a license which is valid for an agreed amount of time. A license can be renewed or blacklisted whenever necessary. Practically, it is up to you if your customers will even see that they have two licenses. For them, the SAVAPI license key can be just another file included in the package.

Of course, your product must make sure that the key does not expire, otherwise it will not scan anymore.

When the key expires, the product will scan using its existing signatures. If you update the signatures and/or the engine and the expiration date in the key is older than the date in the signatures/engine, SAVAPI will not scan.

If you use Avira Updater (*avupdate.exe/.bin*), you can enable the license verification with the `--check` parameter. See 7. Updating SAVAPI - page 60.

7. Updating SAVAPI

As a protection technology, SAVAPI needs regular updates, to keep its detection patterns up to date and to fix eventual issues. The SAVAPI product comes with an integrated updater module (the Avira Updater), which can be used to update SAVAPI on all supported platforms and has the same functions and options on all supported operating systems.

By using command line parameters or by editing configuration files for the Updater, you can carry out the following operations:

- Check if new updates are available, for any of SAVAPI's components (binary, library files, engine, signature files).
- Update any of the SAVAPI components from Avira's update servers or from user-defined servers, with the proper update structure.
- Mirror the already configured update servers.

The Updater mirrors the modular structure of SAVAPI, meaning that SAVAPI can be updated entirely or by modules:

- Product update - Updating the binaries and libraries, depending on the SAVAPI runmode used in the implementation: Service or Library.
- Engine update - Updating the detection engine files (*ae*.**).
- Signatures update - Updating the virus definition files (**.vdf*).

For each SAVAPI update mode (product, engine, signatures) a set of modules is defined, that needs to be sent to the Updater as parameters, either in command line or in a configuration file. The product kit already includes configuration files for the main update scenarios (the configuration files and parameters are detailed in the following sections):

Configuration file	**SAVAPI runmode**	**Updated modules**
avupdate-savapi-engine.conf	daemon/ service	engine, signatures
avupdate-savapi-product.conf	daemon/ service	all modules
avupdate-savapilib-engine.conf	library	engine, signatures
avupdate-savapilib-product.conf	library	all modules

The binary name of the Updater is *avupdate.bin* (Linux/ UNIX) and *avupdate.exe* (Windows).

You can call the Updater with a certain configuration, using the following command line:
```
<binary_name> -C <avupdate-savapi....conf>
```

You can display the Updater's help message with the command:
```
<binary_name> --help
```

At the end of each update cycle, the status of the update will be displayed on the console. If you need to see the progress status, just add `--show-progress` to the command line.

The messages (error, warning, etc.) displayed by the Updater when executed are set in the binary file *avupdate_msg.avr*. The file has to exist in the same folder as the *avupdate* binary file.

7.1 Mirroring the Updater's server structure

You can mirror the Updater's structure locally, on existing update servers. Using this feature, the Updater copies all files locally, as instructed by the configuration files.

Example for Windows:

```
avupdate.exe --mirror --config=<config file> --install-dir=<path>
```

Example for UNIX/ Linux/ Solaris:

```
./avupdate --mirror --config=<config file> --install-dir=<path>
```

When using the `--mirror` option, specifying the `--config` and `--install-dir` parameters is mandatory.

The `<path>` you enter as `--install-dir` value is the destination of the mirroring process. The directory has to exist and it has to grant writing privileges to the user running the mirroring process.

7.2 Avira Updater's configuration files

There are 4 configuration files for Avira Updater, included in the SAVAPI archive.

- Running SAVAPI as daemon or service, means that the SAVAPI binary is running in the background (Linux/ UNIX) or running as a service (Windows). This also means that when an update is required, the daemon/ service must be stopped before starting the update and started after update. For your convenience, this is done automatically by using the following update scripts:
  - *avupdate-savapi-product.conf*
    The file configures the Updater to update SAVAPI, engine and signatures. This script will also update the Updater itself (self-update).
  - *avupdate-savapi-engine.conf*
    The file configures the Updater to update the engine and signatures.
- Running SAVAPI Library (SAVAPI **not** as daemon or service), means that there is no need to stop and start the SAVAPI binary. You can use the following update scripts:
  - *avupdate-savapilib-product.conf*
    The file configures the Updater to update SAVAPI Library, engine and signatures. This script will also update the Updater itself (self-update).
  - *avupdate-savapilib-engine.conf*
    The file configures the Updater to update the engine and signatures.

7.3 Avira Updater's configuration parameters

Before downloading and updating product files, Avira Updater reads the configuration from *avupdate.conf* from its default location, or from any other configuration file specified when calling the binary.

Note
The commands in command line have a higher priority and override the options in the configuration file.

The Updater supports transfer protocols like: HTTP, HTTPS (with or without proxies) and SMB.

With a few exceptions, the configuration parameters listed in this section can be used both in command line and in configuration files. (In configuration files, the parameters are called without the preceding dashes "--".)

Example of parameter in *avupdate.conf:*
```
temp-dir=/tmp
```

Example of parameter in command line:
```
avupdate --temp-dir=/tmp
```

The general syntax for configuring the Updater in command line is:

`./avupdate.bin [options]` – for Linux

`avupdate.exe [options]` – for Windows

The parameter names are cross-platform and **not** case-sensitive. The parameter values have to be compliant with the running platform specifications.

General parameters:

- **--help**

  Displays the help message, about the Updater options.

Note
This option is available only in command line.

- **--version**

  Displays Updater's version.

Note
This option is available only in command line.

- **--config**

  Contains the path to the configuration file. Example:

  `avupdate.bin --config=avupdate.conf`

  > **Note**
  > This option is available only in command line.

  The default value: `avupdate.conf`

- **--no-config**

  The Updater will not read any configuration file. All parameters are given in command line.

  > **Note**
  > This option is available only in command line.

  The default value: `false`

- **--quiet**

  The Updater will not log messages on screen.

  > **Note**
  > This option is available only in command line.

  The default value: `false`

- **--show-progress**

  Shows the download progress.

  The default value: `false`

Directories and files

- **--temp-dir**

  Temporary directory for downloading update files. Example:

  `avupdate.bin --temp-dir=/tmp`

  The default value: `avupdate_tmp_XXXXXX`

- **--backup-dir**

  Backup directory for the existing files, before updating.

  The default value: `<install directory>\avupdate_backup`

- **--install-dir**

  Specifies the installation directory for updated product files.

  ```
  avupdate.bin -- install-dir=./
  ```

- **--update-dir**

  Specifies the location of engine, vdf and *build.dat* in order to get the needed information for the user-agent string. If this parameter is not given, the Updater will use the path from the `--install-dir` option.

- **--key-dir**

  Specifies the directory in which the Updater should search for a valid key.

- **--master-file**

  Specifies the *master.idx* file. Example:

  ```
  avupdate.bin --master-file=/idx/master.idx
  ```

- **--local-master-file**

  Specifies the full path to a local master file to be used instead of the one from the installation directory.

- **--product-file**

  To specify the idx product file. Example:

  ```
  avupdate.bin -- product-file=/idx/savapi-win64-en.idx
  ```

  The product-file parameter can be used both in command line and in config files. It defines the platform where SAVAPI is running. Using this information, the Updater determines which files to check for updates and to download, if necessary.

  The currently possible values for this parameter are:

  - `[savapi|savapilib]-freebsd_v62-en.idx`
  - `[savapi|savapilib]-linux_glibc22-en.idx`
  - `[savapi|savapilib]-linux_glibc24_x86_64-en.idx`
  - `[savapi|savapilib]-openbsd_v39-en.idx`
  - `[savapi|savapilib]-solaris_sparc-en.idx`
  - `[savapi|savapilib]-solaris_sparc64_v8-en.idx`
  - `[savapi|savapilib]-win32-en.idx`
  - `[savapi|savapilib]-win64-en.idx`

  This list of values may change, when new supported platforms are added.

- **--add-var-pair**

  Used to define pairs of variables and values in the INFO file. Example:

  ```
  DESTINATION=%MY_INSTALL_DIR%
  ```

In command line:

```
avupdate.bin --add-var-pair=MY_INSTALL_DIR=/usr/lib/<yourproduct>
```

- **--internet-srvs**

  The list of Internet update servers.

  ```
  avupdate.bin "--internet-srvs=http://dl1.pro.antivir.de, http://
  dl2.pro.antivir.de, http://dl3.pro.antivir.de"
  ```

  An ftp or an https server can also be used for updates. For example:

  ```
  avupdate.bin --internet-srvs=ftp://server
  ```

  or

  ```
  avupdate.bin --internet-srvs=https://server
  ```

- **--peak-handling-srvs**

  Contains the servers used (ftp, http or https) in case no Internet server is available or if both `--ipv4-peak-server-limit` and `--ipv6-peak-server-limit` are reached (see below).

- **--ipv4-peak-server-limit**

  If both `--ipv4-peak-server-limit` and `--ipv6-peak-server-limit` are reached, the list of servers from `--peak-handling-srvs` will be used for updates.

  If this limit is set to 0, the Updater will try to update from all IPv4 servers (`--internet-srvs`) before trying to update from the `--peak-handling-srvs` list.

  The default value: 0

- **--ipv6-peak-server-limit**

  If both `--ipv4-peak-server-limit` and `--ipv6-peak-server-limit` are reached, the list of servers from `--peak-handling-srvs` will be used for updates.

  If this limit is set to 0, the Updater will try to update from all IPv6 servers (`--internet-srvs`) before trying to update from the `--peak-handling-srvs` list.

  The default value: 0

- **--internet-protocol**

  The Internet protocol version. It can have three values: `auto`, `ipv4`, `ipv6`. Example:

  ```
  avupdate.bin --internet-protocol=ipv6
  ```

  The default value: `auto`

Update mode

- **--mirror**

  Performs a mirror update (meaning no pre / post / unpost applications are executed).

See 7.1 Mirroring the Updater's server structure - page 61. This parameter is not shown in the `--help` output.

The default value: `false`

- **--check-if-update-available**

  If this option is set, the Updater will not install any files. It will only check if an update is available.

  There are two situations:

  - If `--skip-master-file` is set, the Updater will download the *product.info* files and it will check all local files against the online ones. It will also log the files that are dirty and must be updated.
  - If `--skip-master-file` is **not** set, the Updater will download only the *master.idx*. It will check only the local *master.idx* against the online one. If identical, it will return "*Nothing to update*". If not identical, it will return "*Update is available*".

  The default value: `false`

- **--update-modules-list**

  Specifies the modules that must be updated (comma-separated list).

  For both SAVAPI runmodes, the following values can be assigned to this parameter, depending on what needs to be updated:

  SAVAPI daemon/service runmode:

  - `SAVAPI3` - includes SAVAPI's binary files in the update process (Savapi and libsavapi3.so on Linux/ UNIX systems; or savapi.exe, savapi3.dll on Windows systems);
  - `SAVAPI3_COMMON` – special module that enables pre and post update actions;
  - `AVE2` – includes the detection engine files in the update process;
  - `VDF` - includes the virus definition files in the update process;
  - `SELFUPDATE` – includes the files related to the Updater in the update process (avupdate_msg.avr and avupdate.bin on Linux/ UNIX systems; or avupdate_msg.avr and avupdate.exe on Windows systems)

  SAVAPI Lib runmode:

  - `SAVAPI3` - includes SAVAPI's libraries in the update process (libsavapi3.so on Linux/ UNIX systems; or savapi3.dll on Windows systems);
  - `AVE2` – includes the detection engine files in the update process;
  - `VDF` - includes the virus definition files in the update process;
  - `SELFUPDATE` – includes the files related to the Updater in the update process (avupdate_msg.avr and avupdate.bin on Linux/ UNIX systems; or avupdate_msg.avr and avupdate.exe on Windows systems)

- **--skip-master-file**

  Skip master file (Use this option if you want to mirror the entire structure, not only the changed files).

The default value: `false`

- **--skip-selfupdate**

  Skip installing Updater files.

  The default value: `false`

- **--no-deltaupdate**

  Specifies that the Updater will not use the delta update feature.

  The default value: `false`

- **--no-version-check**

  Specifies that the Updater will not check the files version in mirror mode.

  In this case it will only check the md5.

  The default value: `false`

- **--no-signature-check**

  Specifies that the Updater will not check if the files are signed.

  Note
  Only uncompressed binary files are checked.

  The default value: `true`

- **--ext-program-timeout**

  Timeout for waiting for an executed pre / post / unpost application (in seconds).

  The default value: `1800 s`

- **--depend**

  If in the product.info file, module 1 depends on module 2, then "`update-modules-list=module 1`" will update both modules 1 and 2.

  The default value: `false (`meaning the Updater will use the new "depend" functionality`)`

- **--ignore-srvs-list**

  For updates from a share or internal http server, the Updater must ignore the servers list present in the IDX file and download everything from the local network and not from the Internet.

  The default value: `false`

Connection settings

- **--user-agent**

  Specifies the user agent string which is reported to the http server.

The default value:
`AntiVir-UpdateCP/<updater version> (<target>;<product>; AVE <engine version>; VDF <version>; <language>; <operating system>)`

Example:

```
AntiVir-UpdateCP/0.0.21 (SELFUPDATE, VDF, AVE2, SAVAPI3;

SAVAPI3-LINUX_GLIBC22-EN; AVE 8.1.1.23; VDF 7.0.6.157; EN;
LINUX_GLIBC22_I386)
```

- **--system-proxy**

  Tells the Updater to use the system proxy settings.

  The default value: `false`

- **--proxy-username**

  Username for proxy authentication.

- **--proxy-password**

  Password for proxy authentication.

- **--proxy-host**

  The name of the proxy server.

- **--proxy-port**

  Proxy port.

- **--username**

  Username for accessing a shared folder, an ftp server or an http server.

- **--password**

  The password for accessing a shared folder, an ftp server or an http server.

- **--update-auth-type**

  Authentication type that will be used for updates. It can be one of the following:

  - `basic`
  - `digest`
  - `ntlm`
  - `any`

  If '`any`' is chosen, the Updater will first query the site to see which authentication methods it supports and then it will pick one of them.

  The default value: `any`

- **--connect-timeout**

  The maximum time in seconds that the connection to the server is allowed to take.

  This is only used to limit the connection phase.

  The default value: `30 seconds`

- **--receive-timeout**

  The timeout (in seconds) for receiving a response after a request to an update server.

  The default value: `30 seconds`

- **--retries**

  Number of retries.

  The default value: 0

Note
The retry mechanism is only applied to the download errors. In case of connect errors, the '`retries`' number is 0 and cannot be changed.

- **--retry-timeout**

  Timeout between retries (in seconds).

  The default value: `0 seconds`

Notification emails

- **--mailer**

  Specifies the method for sending emails. Available values:

  - smtp - for using your own smtp engine
  - sendmail - for using the sendmail binary

  The default value: `smtp`

- **--sendmail-path**

  When `--mailer` is set to sendmail, this parameter specifies the local path of the sendmail binary.

  On UNIX, the default value is: `/usr/sbin/sendmail`.

  It also searches in `/usr/lib/sendmail`

- **--sendmail-arguments**

  If `--mailer` is set to sendmail, this parameter specifies the arguments for running the sendmail binary.

  On UNIX, the default value is: `-oem -oi`

- **--email-to**

  The recipient of the notification emails, if `--notify-when` is not 0 (see below).

  The default value: `root@localhost`

- **--email-from**

  The sender of the notification emails, if `--notify-when` is not 0 (see below).

  The default value: `root@localhost`

- **--notify-when**

  Sends email notifications to the address set with `--email-to`. Available values:

  - 0 - no email notifications are sent (default value),
  - 1 - email notifications are sent in case of "successful update", "unsuccessful update" or "up to date".
  - 2 - email notification only in case of "unsuccessful update".
  - 3 - email notification only in case of "successful update".

  The default value: 0

- **--email-footer**

  Changes the footer of the notification email.

  The default footer of the notification emails is:

  -- *© 2011 Avira Operations GmbH & Co. KG. All rights reserved*.

  Syntax:

  ```
  avupdate.bin --email-to=<yourmail@domain.com> --email-
  footer=<custom footer>
  ```

- **--auth-method**

  When set in *avupdate.conf*, the Updater requires the smtp login data, `smtp-user` and `smtp-password` (see bellow), in order to send email notifications to the address set with `--email-to`.

  The default value: `false`

- **--smtp-user**

  If `notify-when` is not 0 and `auth-method` is set in *avupdate.conf*, the Updater requires the smtp login data: `smtp-user` and `smtp-password`.

- **--smtp-password**

  If `notify-when` is not 0 and `auth-method` is set in *avupdate.conf*, the Updater requires the smtp login data: `smtp-user` and `smtp-password`.

- **--smtp-server**

  The smtp server for sending email notifications, if `--notify-when` is not 0.

- **--smtp-port**

  The smtp port for sending email notifications, if `--notify-when` is not 0.

  The default value: 25

- **--smtp-timeout**

  Timeout for receiving data when connecting to an smtp server (in seconds).

  The default value: 30 seconds

7.4 Avira Updater's logging

The default behavior of the Updater is to create a logfile for each attempted update. The logfile is created in the directory from which the Updater is called.

The Updater offers logging functionalities that can be configured by using the following parameters:

- **--log**

  Specifies a different name of the logfile.

  The default name: `avupdate.log`

  If this option is not present, a default logfile with the name *avupdate.log* will be created in the same directory as the Updater binary.

- **--log-rotate**

  Overwrites the logfiles by rotation. For maintaining a log history, up to 10 recent logfiles will be kept. For example:

  avupdate.log.001, avupdate.log.002, avupdate.log.003, ..., avupdate.log.010, avupdate.log.001...

  The default value: `false`

Note
This parameter is mutually exclusive with `--log-append` (see below). If both parameters are present, the `--log-append` behaviour will be ignored.

- **--log-append**

  Appends to logfile (By default, the log is overwritten).

Note
This parameter is mutually exclusive with `--log-rotate`. If both parameters are present, the `--log-append` behaviour will be ignored.

- **--log-template**

  Option to specify the format of the logfile.

The default template used for logging is:

```
${DAY}/${MONTH}/${YEAR} ${HOUR}:${MINUTE}:${SECOND}
${FULLHOST}[${PID}]: ${SOURCE}: ${LEVEL}: ${MSG}
```

where:

- `${DAY}/${MONTH}/${YEAR}` - date format
- `${HOUR}, ${MINUTE}, ${SECOND}` - time format
- `${FULLHOST}` - hostname
- `${PID}` - pid of the program generating the log
- `${LEVEL}` - the message level, as set by the program ("*DEBUG*", "*INFO*", "*MESSAGE*", "*WARNING*", "*ERROR*", "*FATAL*", "*MAX_LEVEL*", "*UNDEFINED*")
- `${MSG}` - the message sent to the log

Example of log entry:

16/07/2008 12:34:04 abc-desktop [18428]: UPD: ERROR: Smtp engine returned error: Connection refused

The user can specify the desired information to be logged.

For example:

- in *avupdate.conf*:
  `log-template=${MSG}`
- in the command line:
  `avupdate.bin --log-template=${MSG}`

7.5 Avira Updater's return codes

The Updater's main return codes are:

- 0 - successful update
- 1 - nothing to update
- -1 - unsuccessful update

8. Service

8.1 Support

Support service

All necessary information on our comprehensive support service can be obtained from our website http://www.avira.com.

Forum

Before you contact the hotline, we recommend that you visit our user forum at http://forum.avira.com.

FAQs

Please also read the FAQs section on our website:
http://www.avira.com/en/support-for-business-faq

Your question may already have been asked and answered by other users in this section.

8.2 Contact

Address

Avira Operations GmbH & Co. KG
Kaplaneiweg 1
D-88069 Tettnang
Germany

Internet

You can find further information about us and our products at the following address:
http://www.avira.com

9. Appendix

9.1 SAVAPI binaries

9.1.1 Windows

Assuming SAVAPI is installed in a folder, it will contain the following files:

Description	File name on disk	Installation	Redistributable
SAVAPI binary	*savapi.exe*	Mandatory	YES
SAVAPI library and its dependencies	*savapi3.dll* *savapi3client.dll*	Mandatory	YES
Engine binaries	*ae*.** *unacev2.dll*	Mandatory	YES
SAVAPI key file	**.key*	Mandatory	YES
SAVAPI configuration file	*savapi3.conf*	Optional	YES
Engine VDFs	*vbase000.vdf, ..., vbase031.vdf*	Mandatory	YES
SAVAPI plugin framework	*plugin_framework.dll*	Optional (only if the plugin framework and SAVAPI plugins are used)	YES

The files for the Avira Updater

Description	File name on disk	Installation	Redistributable
Libraries (shared with SAVAPI)	*avupdate_msg.avr*	Optional	YES
Main binary	*avupdate.exe*	Optional	YES
Configuration file	*avupdate-savapi-product.conf*	Optional	YES
Configuration file	*avupdate-savapi-engine.conf*	Optional	YES
Configuration file	*avupdate-savapilib-product.conf*	Optional	YES
Configuration file	*avupdate-savapilib-engine.conf*	Optional	YES

9.1.2 UNIX

If you have any key files, you can install them in the binaries' directory. Make sure that the permissions for the key files are correct.

Description	File name on disk	Installation	Redistributable
SAVAPI binary	*savapi*	Mandatory	YES
SAVAPI library and its dependencies	*libsavapi3.so* *libsavapi3client.so*	Mandatory	YES
Engine binaries	*ae*.**	Mandatory	YES
SAVAPI plugin framework	*libplugin_framework.so*	Optional (only if the plugin framework and SAVAPI plugins are used)	YES
SAVAPI key file	**.key*	Mandatory	YES
Engine VDFs	*vbase000.vdf, ..., vbase031.vdf*	Mandatory	YES
SAVAPI configuration file	*savapi3.conf*	Optional	YES

The files for the Avira Updater

Description	File name on disk	Installation	Redistributable
Libraries (shared with SAVAPI)	*avupdate_msg.avr*	Mandatory	YES
Main binary	*avupdate.bin*	Optional	YES
Configuration file	*avupdate-savapi-product.conf*	Optional	YES
Configuration file	*avupdate-savapi-engine.conf*	Optional	YES
Configuration file	*avupdate-savapilib-product.conf*	Optional	YES
Configuration file	*avupdate-savapilib-engine.conf*	Optional	YES

AVIRA

live *free.*™

Avira Operations GmbH & Co. KG
Kaplaneiweg 1 | 88069 Tettnang
Germany

Telephone: +49 7542-500 0
Facsimile: +49 7542-500 3000
www.avira.com