Space Details

Available Pages

- General principles of the antivirus engine
- The structure of the antivirus engine
- Virus bases boot options

Apps creation methods

Apps created by you on the base of Zillya Antivirus engine SDK may interact with anti-virus engine in several ways:

1. The direct use of antivirus engine libraries.
2. Connection with the core libraries by a service presented in this SDK

The direct use of libraries of antivirus engine complicated to implement and requires a full understanding of the process of the antivirus engine. Application developed by you must arrange loading and unloading of the core, to define the parameters of the work, to provide scan, follow the thread synchronization.

The advantages of direct work with antivirus engine libraries :

1. Full access to antivirus engine functionality
2. Maximum speed
3. Ability to specify any structure of your application

Disadvantages:

1. We need to create functional for the operation of antivirus engine

Additionally, this method is described in section Zillya Core Library

Working through the service is more simple and suitable for programmers who only began to work with this SDK. All the work in this case is done by the service itself. User application needs to send a team to provide scanning and get the results.

The advantages of working with a service:

1. Quick start. You will need a few minutes in order to write your first application.

Disadvantages:

1. Less flexible functionality
2. Need to register and start the service on the computer on which you run your application

Additionally, this method is described in section Zillya AVEngine Service

At the same time your application when using the service may not be sufficiently flexible. For such cases, this SDK includes source code for all submitted applications. You can get the source code of the service, or any other application as a basis, and write your application that will meet your needs.

The package also includes examples of both the direct use of the core and the interaction through the service.

Package contents

- app * \ - Contains the source code of the applications submitted to the package
- bin * \ - The binaries of AvEngine and utility.
- docs * \ - Description and function of SDK
- examples * \ - Examples of engine connection to theapplication
- include * \ - header files needed for engine connection
- lib * \ - library for interaction with the service

System requirements

Processor: 1 GHz or higher

RAM: 512 MB or more

Hard disk space: 80 MB

Operating System: XP SP2/SP3, Vista, Windows 7 (x86 or x64), Windows 8 (x86 or x64), all versions of Windows 2003 Server, Windows 2008 Server

Zillya AVEngine Service

What is Zillya AVEngine Service

Zillya AVEngine Service - is an application that allows you to quickly and easily interact with user applications on the base of anti-virus engine. It takes over the function of downloading and configuring the kernel, providing a scan and save the results, so using it you can work directly with the required functionality.

The service supports multithreaded scanning, so you can start several scans in a row, and create any number of applications that use the service functions.

The service works with system privileges, so you will not have problems with access to the protected files and folders.

Installation

To install the service, you can use the scripts that are in the bin folder of the SDK.

The use of pipe to interact with the service

Introduction

Using pipe, this is another way to interact with Zillya AVEngine Service and verification files for viruses. let us consider a simple example of scanning a file.

It is assumed that when writing a program, the developer already has a basic knowledge of WIN32 API and is able to work with named channels.

Using the code

To work with the service, you need to include the header file zsdk_def.h. It defines the data to be transmitted and received in the service after a scan. Copy this file to the folder of the SDK include folder in your project, and plug it in the code:

```
#include "zsdk_def.h"
```

To send commands to the service, you have to connect to it. Zillya AVEngine Service waits for a connection using the channel

```
"\\\\.\\pipe\\ZSDK-{F671A1CA-7BA6-4e57-9E98-0D2AE0985A42}"
```

After establishing a connection, the service will be expected of your command. The command to scan a file is a filled zRPCRequest structure, which is need to be sent to the service.

```
BOOL bResult;
DWORD cbWritten, dwStatus = 0;
zRPCRequest request;

wcscpy_s(request.szScanPath, MAX_PATH, L"c:\\WINDOWS\\notepad.exe");

bResult = WriteFile(m_hPipe, (LPVOID)&dwStatus, sizeof(DWORD), &cbWritten, NULL);

if (!bResult || cbWritten != sizeof(DWORD))
 return FALSE;

bResult = WriteFile(m_hPipe, (LPVOID)&request, sizeof(zRPCRequest), &cbWritten, NULL);

if (!bResult || cbWritten != sizeof(request))
 return FALSE;
```

After receiving the command, the service will give anti-virus engine to scan specific files, and return you to the results of the scan, using the structure zPRCAnswer.

```
BOOL bResult;
DWORD cbBytesRead, dwStatus = 0;
zPRCAnswer answer;

bResult = ReadFile(m_hPipe, (LPVOID)&dwStatus, sizeof(DWORD), &cbBytesRead, NULL);

if(!bResult || cbBytesRead != sizeof(DWORD))
 return FALSE;
```

```
bResult = ReadFile(m_hPipe, (LPVOID)&answer, sizeof(zPRCAnswer), &cbBytesRead, NULL);

if(!bResult || cbBytesRead != sizeof(zPRCAnswer))
 return FALSE;
```

The results can be returned more than once, depending on how many files were transferred to the analysis. In this case, the service will return ZSDK_REQUEST_MORE_DATA in dwStatus.

Disconnect after getting the results of the scanning of all files.

Zslib use for the interaction with the service

Introduction

zlib - is a library, which allows your application to gave the command to Zillya AVEngine Service. It is created to facilitate the interaction of your application with the service and the fastest scanning organization. It is enough to call several functions to create minimal application.

Setup

To use the library, you need to include the header file zsdk_def.h. Copy this file to the folder of the SDK include folder in your project, and write a program's code:

```
#include "zsdk_def.h"
```

Also, you will need to copy the file zslib.lib from the folder lib of this SDK to a folder with your project, and connect it to the projects. This can be done using your IDE project settings or specifying

```
#define _ZSDK_INCLUDE_STATIC
```

Using the code

We need three variables to work with library:

```
void main()
{
 DWORD hScan; // will contain the handle of the current scan
 DWORD dwRes; // the result of the scan functions
 zPRCAnswer answer; // the result of the scan
```

To begin scanning, you need to call the StartScan function. Its only parameter is a path to scan.

```
hScan = StartScan(L"c:\\WINDOWS\\notepad.exe");
```

To get the results of the scan, you need to call the GetScanData function, passing it the handle scanning and structure to be filled. If you scan multiple files, function should be called for each file.

```
do {

 if((dwRes = GetScanData(hScan, answer)) == ZSDK_REQUEST_ERROR)
  break;

....

} while(dwRes == ZSDK_REQUEST_MORE_DATA);

}
```

Scan results are contained in answer structures. To verify that the file is infected or not, you can use the part of ScanStatus structure

```
if(!answer.ScanStatus)
 printf("no viruses detected");
else
 printf("threat detected");
```

Now we have an application that will connect to the service and to scan files for viruses.

GetScanData

Get the next file scan result

Syntax

```
DWORD GetScanData(
DWORD hScan,
zPRCAnswer &answer
);
```

Parameters

hScan [in]
Scaning thread handle, returned by StartScan

answer [out]
Pointer to the zPRCAnswer structure, that contain information about scan result.

Return value

Constant/Default value	Description
SDK_REQUEST_OK 0	Scaning succeeded
ZSDK_REQUEST_MORE_DATA 1	Scaning succeeded. More data available
ZSDK_REQUEST_ERROR -1	Error scanning

StartScan

Starts new scan thread

Syntax

```
DWORD StartScan(
LPWSTR szPath
);
```

Parameters

szPath [in]
Path that will be scanned

Return value

If success, return handle of the new scan, 0 otherwise.

zRPCAnswer

Contains the results of the scan

Syntax

```
typedef struct zRPCAnswer {
wchar_t szFileName[MAX_PATH];
wchar_t szVirName[VIRNAME_LEN];
DWORD ScanStatus;
DWORD FilesScanned;
DWORD VirCount;
DWORD Action;
} *pZPRCAnswer;
```

Members

szFileName
File that scanned

ScanStatus
File scan status, if file is clean, sets to 0

szVirName
Name of virus found

FilesScanned
Number of objects scanned (in the case of container> 1)

VirCount
The number of bodies discovered viruses

Action
Action performed on the file

zRPCRequest

Provides information about scanning. Used to initiate a new scan

Syntax

```
typedef struct zRPCRequest {
        wchar_t szScanPath[MAX_PATH];
} *pZRPCRequest;
```

Members

szScanPath [in]
Path, that will scanned

Anti-virus engine (hereinafter «CoreAV» or "core") is designed to combat various types of malicious code. The kernel provides functions for detecting and neutralizing malicious code, and registry keys are created by various malicious programs.

CoreFree

Closes an open antivirus core handle

Syntax

```
int TCoreFree(

DWORD CoreHandle

);
```

Parameters

CoreHandle [in]
Pointer to the open antivirus core handle

Return value

Constant/Default value	Description
0	Discharge flow scanning was successful
-1	General error kernel
-2	Invalid pointer thread scanning
-3	Scan the specified stream does not exist

CoreFScan

Scans the file for viruses

Syntax

```
int TCoreFScan (

DWORD CoreHandle,

const PCoreFScan_Interface params


);
```

Parameters

CoreHandle [in]
Opened antivirus core handle, created by a previous call to the CoreInit

params [in|out]
Pointer to the PCoreFScan_Interface structure, which contains information about the scan. It also contains the return value of the scan results.

Return value

Constant/value	Description
0	File is checked successfully, does not contain malicious code
1	Detected suspicious code (heuristic analysis)
2	Malicious code is detected (infected file)
-1	General error of the scan engine
-2	Incorrect number of scanning thread (abnormal range)
-3	Scan the specified thread does not exist (not initialized)
-4	Set too long path for the file
-5	Error opening/reading the file

CoreInit

Initializes the loaded copy of the core. The application must call this function, before before using the kernel.

Syntax

```
DWORD TCoreInit(

const PCoreInit_Interface params

);
```

Parameters

params [in]
Pointer to the TCoreInit_Interface structure that have initializes parameters

Return value

Return CoreHandle value, which would subsequently need to pass the rest of the engine caused by anti-virus functions, or an error code of the kernel initialization.

Constant/value	Description
0	Null value (or utility handle)
1 - 64	Correct copy of the handle to the initialized
0xFFFFFFFF	Failed to initialize AVEHAL
0xFFFFFFFE	Not found AVEHAL
0xFFFFFFFD	Failed to initialize AVEMem
0xFFFFFFFC	Not found AVEMem
0xFFFFFFFB	Failed to load vl001.dat
0xFFFFFFFA	Failed to load vl002.dat
0xFFFFFFF9	Failed to load vl003.dat
0xFFFFFFF8	Failed to load vl004.dat
0xFFFFFFF7	Failed to initialize the internal functions

CoreMScan

Scans the memory block for viruses

Syntax

```
int TCoreMScan (

DWORD CoreHandle,

const PCoreMScan_Interface params


);
```

Parameters

CoreHandle [in]
Opened antivirus core handle, created by a previous call to the CoreInit

params [in|out]
Pointer to the TCoreMScan_Interface structure, which contains information about the scan. It also contains the return value of the scan results.

Return value

Constant/value	Description
0	Memory block is checked successfully, does not contain malicious code
1	Detected suspicious code (heuristic analysis)
2	Malicious code is detected (infected file)
-1	General error of the scan engine
-2	Incorrect number of scanning thread (abnormal range)
-3	Scan the specified thread does not exist (not initialized)
-4	Set too long path for the file
-5	Error opening/reading the file

Remarks

The function CoreMScan not release the memory area containing the object to be scanned. This should make the application, which allocated memory area after the test result (exit the CoreMScan).

Function to check the memory buffer CoreMScan most similar to the previous function CoreFScan (function check the file) and contains only two differences:

1. To test passed is not the way to the file on disk, but memory buffer that contains the necessary checks for an object (file);
2. The current version of the antivirus engine testing of the memory buffer is only possible mode of "report" (that is not possible to treat the virus).

All other inbound and outbound parameters are fully consistent with the parameters of the function checks the file.

CoreOptions

Sets the options of the antivirus engine

Syntax

```
DWORD CoreOptions (

DWORD CoreHandle;

const PCoreOptions_Interface options;

);
```

Parameters

CoreHandle [in]
Opened antivirus core handle, created by a previous call to the CoreInit

options [in]
Pointer to the TCoreOptions_Interface structure, which contains information about the scan options.

Return value

CoreStopScan

Stops scanning the object to the specified stream

Syntax

```
DWORD CoreStopScan(

DWORD CoreHandle

);
```

Parameters

CoreHandle [in]
Opened antivirus core handle, created by a previous call to the CoreInit

Return value

Remarks

Interruption test object (file, archive, memory array, etc.) is carried out using the function scan engine CoreStopScan. At the same anti-virus engine finishes last logical operation that is carried out to test the file and exits caused by the earlier test function object (file or memory buffer).
It is important to bear in mind that in the case of a function call interrupts the scan test result value of the object, the call is interrupted by this function must be considered invalid. For example, the object can be checked only part of the anti-virus, or not tested at all, after which the work was interrupted by anti-virus engine.
Use this feature is recommended if necessary interrupt the scanning process during the test files, installers, and other containers, where no opportunity to wait for the end of the test object entirely.

CoreRScan

Scans the registry key for viruses

Syntax

```
int CoreRScan (

DWORD CoreHandle,

const PCoreRScan_Interface params


);
```

Parameters

CoreHandle [in]
Opened antivirus core handle, created by a previous call to the CoreInit

params [in|out]
Pointer to the TCoreRScan_Interface structure, which contains information about the scan. It also contains the return value of the scan results.

Return value

Constant/value	Description
0	Registry key is checked successfully, does not contain malicious code
1	Detected suspicious code (heuristic analysis)
2	Malicious code is detected (infected file)
-1	General error of the scan engine
-2	Incorrect number of scanning thread (abnormal range)
-3	Scan the specified thread does not exist (not initialized)
-4	Set too long path for the file
-5	Error opening/reading the file

CoreInitMerge

Merges the extending set of bases, with initial set, when core was load in reduced mode

Syntax

```
DWORD CoreInitMerge(

const PCoreInit_Interface params

);
```

Parameters

params [in]
Pointer to the TCoreInit_Interface structure that have initializes parameters

Return value

Remarks

The core can be initiated in one of two modes: full version of bases, and an reduced version of bases. If the kernel has been loaded with an reduced version of the database, TCoreInitMerge performs refilled missing virus entries. For more details see Virus bases boot options

RegInfoCallback

Callback function that is called, when the scan engine finishes checking the registry key

Syntax

```
DWORD RegInfoCallback (
DWORD CoreHandle,
LPCSTR VirusName
);
```

Parameters

CoreHandle
Opened antivirus core handle, which identifies the scanning stream.

VirusName
The name of the virus found

Return value

If function successed, it should return 0.

UnpackingInfoCalback

Callback function that is called, when the scan engine finishes checking the next file in the archive

Syntax

```
DWORD UnpackingInfoCallback (
DWORD CoreHandle,
LPCWSTR FileName,
CHAR Status,
LPCSTR VirusName
);
```

Members

CoreHandle
Opened antivirus core handle, which identifies the scanning stream.

FileName
The full path and display name of the extracted files (contains both the way to the container, and the path to the inside of the container)

Status
File check status. The statuc can be one of the list values:

Constant/value	Description
crOk 0	File is verified successfully, does not contain malicious code
crSuspected 1	Suspicious code (heuristic analyzer)
crInfected 2	Suspicious code (file infected)
crCommonError -1	A common error of the antivirus engine
crErrorScanThreadNumber -2	Scanning stream wrong number
crErrorScanThreadExistence -3	The specified thread does not exist
crErrorFilePathTooLong -4	Specified path is too long
crErrorOpenReadFile -5	Open file error

VirusName
The name of the virus found

Return value

If function successed, it should return 0.

Remarks

CallBack function UnpackingInfoCallBack declared in an application that uses anti-virus engine, and is used to display additional information, initiated by the anti-virus kernel: results of the test objects inside a container.
CallBack function is UnpackingInfoCallBack function for displaying the status of files retrieved to check out file containers (archives, installers, email databases, etc.). This function must be declared in the body of an application that uses anti-virus engine. By using multiple scanning threads, each of them can be passed as a pointer to the different (individual) functions as well as one and the same function as a unified whole application.

TCoreData

Contains information about the loaded copy of the kernel

Syntax

```
typedef struct TCoreData {
DWORD AVEVers;
DWORD AVERecCount;
} *PCoreData;
```

Members

AVEVers
Antivirus core version

AVERecCount
Antivirus bases records count

Remarks

Structure is filled by the kernel when the CoreInit function called. Included in the TCoreInit_Interface structure.

TCoreFScan_Interface

Contains information about the file to be scanned. Return valuesafter the scanning is also written here.

Syntax

```
typedef struct TCoreFScan_Interface {
LPCWSTR FName;
DWORD ToDo;
DWORD BaseMode;
PCoreUnpackMode UnpackMode;
LPSTR VName;
PCoreResult CoreResult;
} *PCoreFScan_Interface;
```

Members

FName [in]
Name of file, that will scanned

ToDo [in]
File process mode. This option can be one of this values:

Constant/value	Description
0x00000000	The file only will be tested
0x00000001	The file will be cured
0x00000002	The file will be deleted

BaseMode [in]
This parameter sets the range of BaseMode anti-virus and heuristic algorithms are used to test anti-virus kernel of these objects. This parameter specifies the binary masks:

Constant/value	Description
0x000000FF	a core set of databases
0x0000FF00	additional databases. This mode is active only when the core set of databases
0x00FF0000	basic heuristic algorithms
0xFF000000	additional (extended) heuristic algorithms. This mode is active only when the base heuristic algorithms

UnpackMode [in]
Sets the configuration container extraction, if the file is such. Pointer to the TCoreUnpackMode structure.

VName [out]
Pointer to the virus name buffer. Must be at least 64 characters

CoreResult [out]
Pointer to the TCoreResult structure. Contains the results an object scanning.

TCoreInit_Interface

Contains information about the copy of the kernel is initialized

Syntax

```
typedef struct TCoreInit_Interface {
LPCWSTR EPath;
LPCWSTR VPath;
DWORD Options;
PUnpakingInfoCalback UnpakingCalback;
PRegInfoCalback RegCalback;
LPCWSTR LogFPath;
PCoreData CoreData;
} *PCoreInit_Interface;
```

Members

EPath
The path to the folder that contains anti-virus engine

VPath
The path to the folder containing the virus database

Options
The options of the core being loaded

Constant/value	Description
0x00000001	Stop checking the archive, on detected threats
0x00000002	Run core in debug mode. With this option core writes maximum log information
0x00000004	Load reduced base

UnpakingCalback
A pointer to the CallBack function UnpackingInfoCalback that displays information about the verification of embedded objects in containers.

RegCalback
A pointer to the CallBack function RegInfoCallback that displays the results of the check register

LogFPath
Path to LOG-file with error messages

CoreData
Pointer to the TCoreData struct.

TCoreMScan_Interface

Contains information about the memory block to be scanned. Return valuesafter the scanning is also written here.

Syntax

```
typedef struct TCoreMScan_Interface {
LPCVOID MemPointer;
DWORD MemSize;
DWORD BaseMode;
PCoreUnpackMode UnpackMode;
LPSTR VName;
PCoreResult AVEResult;
} PCoreMScan_Interface;
```

Members

MemPointer [in]
A pointer to memory containing the object to be scanned.

MemSize [in]
The size of the buffer memory

BaseMode [in]
Mode of virus bases usage

UnpackMode [in]
A pointer to the TCoreUnpackMode structure parameters and unpacking of containers

VName [out]
Pointer to the virus name buffer. The buffer size can be 64 bytes.

AVEResult [out]
A pointer to the TCoreResult structure of scan results

TCoreOptions_Interface

Contains information about the additional core options

Syntax

```
typedef struct TCoreOptions_Interface {
DWORD Options;
LPCSTR LogFPath;
DWORD Reserved1;
DWORD Reserved2;
DWORD Reserved3;
DWORD Reserved4;
DWORD Reserved5;
} *PCoreOptions_Interface;
```

Members

Options
Additional options

LogFPath
Path to the log file

Reserved1
reserved

Reserved2
reserved

Reserved3
reserved

Reserved4
reserved

Reserved5
reserved

TCoreResult

Structure consiscts the result of the scaning

Syntax

```
typedef struct TCoreResult {
DWORD FilesCount;
DWORD VirCount;
DWORD InfectionType;
DWORD ActionRes;
DWORD IsContainer;
} *PCoreResult;
```

Members

FilesCount
Number of objects scanned (in the case of container> 1)

VirCount
The number of bodies discovered viruses

InfectionType
Supported infected files operations

Constant/value	Description
0x00000001	Cure
0x00000002	Remove

ActionRes
Action performed on the file

Constant/value	Description
0x00000000	Skipped
0x00000001	Cured
0x00000002	Removed

IsContainer
Flag detection container:

Constant/value	Description
0x00000000	No
0x00000001	Yes

TCoreRScan_Interface

Contains information about the registry key to be scanned.

Syntax

```
typedef struct TCoreRScan_Interface {
LPCSTR RegKey;
} *PCoreRScan_Interface;
```

Members

RegKey
Pointer to the registry key, that will scanned.

TCoreUnpackMode

The structure sets the parameters of TCoreUnpackMode antivirus kernel with archives, installers, mail databases and other containers which may contain other full-file objects (files)

Syntax

```
typedef struct TCoreUnpackMode {
DWORD MaxArchSize;
DWORD MaxArchDepth;
UINT64 ArchMode;
UINT64 InstMode;
UINT64 ContMode;
UINT64 Reserved1;
UINT64 Reserved2;
UINT64 Reserved3;
UINT64 Reserved4;
UINT Flags;
} *PCoreUnpackMode;
```

Members

MaxArchSize
The maximum size of the container being unpacked. If 0, no limits.

MaxArchDepth
The maximum depth of container extraction. If 0, no limits.

ArchMode
Mode of unpacking archives. Formats that are necessary to unpack the bitmask. To extract all the formats, you must specify a value of -1
Bitmap Type of archive/container

Constant/Default value	Description
0x0000000000000001	ZIP
0x0000000000000002	RAR
0x0000000000000004	7Zip
0x0000000000000008	ACE
0x0000000000000010	ARJ
0x0000000000000020	MS CAB
0x0000000000000040	IS CAB
0x0000000000000080	GZip (GZ)
0x0000000000000100	BZ2 (BZ)
0x0000000000000200	RPM
0x0000000000000400	DEB
0x0000000000000800	LZH

0x0000000000001000	TAR
0x0000000000002000	CPIO
0x0000000000004000	ISO

InstMode
Mode of unpackers installers. Formats that are necessary to unpack the bitmask. To extract all the formats, you must specify a value of -1

Constant/Default value	Description
0x0000000000000001	NullSoft NSIS
0x0000000000000002	WISE Installer

ContMode
Mode of use of different containers unpackers (MSI, CHM files, mail databases and other types of containers). Formats that are necessary to unpack the bitmask. To extract all the formats, you must specify a value of -1

Constant/Default value	Description
0x0000000000000001	MSI
0x0000000000000002	CHM
0x0000000000000004	MultiEXE

Reserved1
reserved

Reserved2
reserved

Reserved3
reserved

Reserved4
reserved

Flags
Additional flags bitmask:

Constant/value	Description
0x00000001	Display the type of container extracted

Create a simple application on based on Zillya CoreLibrary

Introduction

We describe how to create a scanning applications, working directly with the anti-virus engine. This method is faster and gives you more control over the work of the core, but you will need to configure your own anti-virus engine for its work. Moreover, in this example, we'll create just one thread scan.

Using the code

Lets include the header file engine_n.h, which defines all the functions and structures that are needed in work. The file itself can be found in the include folder of this SDK.

```
#include "engine_n.h"
```

We need a number of structures for work. TCoreInit_Interface - to initialize the core, TCoreFScan_Interface - to set the scan settings, TCoreUnpackMode - to set parameters for extracting container, TCoreResult - for scan results, coreData - for core parameters.

```
TCoreInit_Interface initParams;
TCoreFScan_Interface scanParams;
TCoreUnpackMode unpackMode;
TCoreResult scanResult;
TCoreData coreData;

memset(&initParams,0,sizeof(TCoreInit_Interface));
memset(&scanParams,0,sizeof(TCoreFScan_Interface));
memset(&unpackMode,0,sizeof(TCoreUnpackMode));
memset(&scanResult,0,sizeof(TCoreResult));
```

Download the core and needed functions. We need three functions to scan the file: coreInit - to initialize the core, coreFScan - actually scanning, coreFree - to unload the core.

```
PCoreInit coreInit;
PCoreFree coreFree;
PCoreFScan coreFScan;

if(!hLib = LoadLibrary(L"CoreMain.dll"))
 return 0;

coreInit = (PCoreInit)GetProcAddress(hLib, "CoreInit");
coreFree = (PCoreFree)GetProcAddress(hLib, "CoreFree");
coreFScan = (PCoreFScan)GetProcAddress(hLib, "CoreFScan");

if(!coreInit || !coreFree || !coreFScan)
 return 0;
```

Initialize the core. You need to fill TCoreInit_Interface structure and pass it to a function CoreInit. For the simplest initialization you need only to specify the core file and virus databases.

```
initParams.CoreData = &coreData;
initParams.EPath = szCorePath;
initParams.VPath = szCorePath;
```

```
hCore = coreInit(&initParams);
```

After initialization, the core can start scanning. You need to fill the scan and call the coreFScan. The given code is the example of simple filled TCoreFScan_Interface structure, and do not specify the parameters of extracting archives. This means that files will not be checked.

```
scanParams.FName = L"C:\\WINDOWS\\notepad.exe";
scanParams.VName = vNameBuffer;
scanParams.BaseMode = 0xffffffff;
scanParams.UnpackMode = &unpackMode;
scanParams.CoreResult = &scanResult;

result = coreFScan(hCore, &scanParams);
```

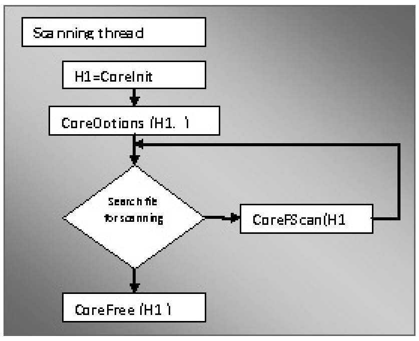
Function gives back the results of file scanning after finishing. You can find the additional information in TCoreResult structure.

```
if(result == 0)
 printf("no viruses found");
```

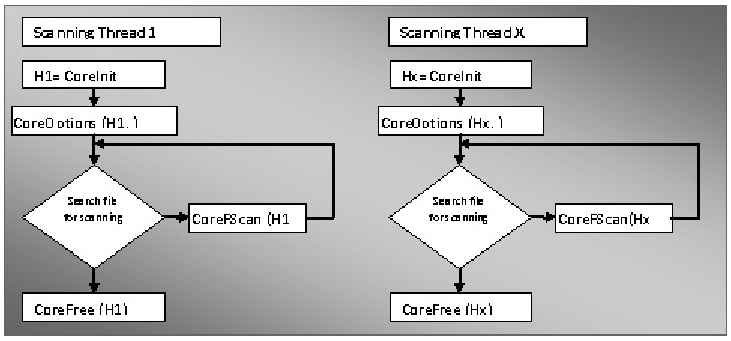
The core should be released in the end of the programme work.

```
coreFree(hCore);
FreeLibrary(hLib);
```

Anti-virus engine is multithreaded CoreAV multimodule kernel. In one application, no matter how many copies of the kernel is used, must be loaded into memory only one copy of the kernel, but for each thread (Thread), which will work with anti-virus kernel, you must perform your own initialization function. Whole kernel supports the ability to work up to 32 streams within a single application
To initialize the main file to load the scan engine (CoreMain.DLL), Handle having libraries of functions to determine the address of the antivirus engine. Next, for each thread to scan for viruses to initialize a copy of the kernel (CoreInit), by calling the number you get your copy of the kernel within the application. This number will always be used to work with the copy of the kernel.
For example: you need to scan files at once anti-virus kernel in 2 threads. In this case, you call CoreInit twice (saving the return values of two variables, such as H1 and H2). Further, for that would give the team the first copy of the kernel to check the memory, you only need from any thread (Thread) application call the CoreFScan (H1, ...). For that would abort the copy, you need to call CoreStopScan (H1, ...). To change the settings for your copy of the kernel you are using the CoreOptions (H1, ...).
With this you can use two different threads (Thread) cause the application simultaneously check two or more files, specifying in each call to a number of copies of the initialized engines.
Examples of the use of anti-virus engine in a single-threaded and multithreaded applications:
Single-threaded application:



Multi-threaded application with equivalent currents:



In the case of checking scanning archives, installers, and other types of containers, from which anti-virus engine extracts the attached files, test results of the attached files are transferred to the CallBack function whose address is specified when initializing a copy of the antivirus engine. Pay attention to the fact that different copies of the antivirus engine you can specify how the individual functions CallBack, and the exact same one (at all) function, as when it is called anti-virus kernel is also sent copies of the antivirus engine room

The structure of the antivirus engine

Antivirus kernel CoreAV is a set of files that are loaded by the loading master file of the engine as a dynamic link library, and call the initialization function of the library.
Files of antivirus engine:

- CoreMain.DLL – main interface file scan engine;
- Core*.DLL – additional service core library;
- VL*.dat – service modules, the scan engine in its own file format;
- VS0000??.DAT – basic anti-virus
- VS99????.DAT – heuristic base kernel
- All other files with mask VS*.DAT – additional anti-virus bases (updates)

Virus bases boot options

There are several modes of loading during core initialization. The kind of boot modes of databases is determined by the core initialization options.

1. Download of the complete set of bases. In this mode, the scan engine loads all the bases.
2. Download of the subsets of the database. In this mode, the scan engine loads only part of the whole set of databases.

Shortened base - is the records added to the database in the last year (as well as a few thousand records, and especially complex mechanisms of detection of complex viruses, worms, rootkits), plus all of the records that are over three years of the product at least once worked on the computer of any user of Zillya! antivirus.
The information on which viruses were detected, collected anonymously from users' computers when the user has given his permission to install.