SAP®

Technical Specification

Release: 2.00

SAP NetWeaver

Virus Scan Interface (NW-VSI)

Interface specification for an external content scan adapter for SAP NetWeaver

History

Version	Status	Date
0.9	Draft	01.07.2002
0.95	Draft	21.11.2002
1.00	Released	14.03.2003
1.50	Released	15.10.2004
1.70	Released	17.10.2007
1.80	Released	12.02.2012
2.00	Released	09.01.2013

SAP Integration and Certification Center (ICC) storage location:
ftp://ftp.sap.com/pub/icc/nw-vsi

Copyrights and Trademarks

Most recent version available: http://www.sap.com/corporate-en/legal/copyright/index.epx

Partners for NW-VSI should contact icc@sap.com and/or copyrights@sap.com

Contents

0 History and Version

Up to version 1.0, this document was released with the title "**Virus Scan Adapter (VSA)**" and provided a technical description of the connection of external anti-virus products to SAP applications.

The technical extensions for integrating an external content security product are included as of document version 1.50. The certifiable interface for the Virus Scan Adapter is called **"NW-VSI"**. The interface specification has therefore been renamed to the official name of the certifiable interface. The main versions of the adapter specification are, however, always compatible with the main version of the official interface.

The usage as content filter in addition to anti-virus scan was added with versions 1.70 and 1.80.

The version 2.00 defines the content filter and virus-scan together as must for a certification.

History:

Version 0.95:	Beta-version of a virus scan adapter.
Version 1.00:	First implementation of an external virus scan interface using various prototypes of adapters.
Version 1.50:	Extension of the interface with content scan (see multiple extensions of parameters and enumerations with "content" definition). Redefinition of the generic type definitions within the specification to be able to use primitive data types in accordance with the Java language. Addition of a callback interface for the delegation of I/O requests to the caller.
Version 1.60:	New parameter VSA_INIT_TEMP_PATH.
Version 1.70:	Changes for active content scan. New scan type was introduced because of customer demand. Use-Case is for detection of Script in Files (XSS in Files)
Version 1.80:	Extensions for MIME filtering based on content detection. New scenario used, called File-Classification
Version 1.90:	New parameters for license setting and updates of external engines
Version 2.00:	Extensions for usage of as Web Content Filter. Specify memory allocation for New certification: NW-VSI 2.00

1 Overview

This document describes the interface for integrating external anti-virus and content security solutions into SAP applications. This allows SAP's customers to choose their own preferred security products. Security partners don´t need technical knowledge about SAP applications, because SAP applications will always use this interface to invoke the functions in the security product.

The "**NW-VSI"** interface is provided for this purpose. Vendors of security solutions in the anti-virus and content security areas can be certified by SAP AG for this interface.

The term SAP application indicates all existing and new software solutions which are part of SAP AG (SAP Group). This document primarily describes SAP NetWeaver as SAP application platform (formerly also known as R/3 or later SAP Web-Application Server), however products like Business-One (B1) or Sybase provide also VSI.

The abbreviation VSI was created at a time where the classical Viruses infected the personal computers (PC) but did not harm the backend servers. The name for this interface specification today is obsolete; however it was decided to use this abbreviation for further versions. As the classical AV products have changed, changes now NW-VSI, because the threats have changed.

VSI is not intend to protect against viral ABAP code (or any so-called ABAP-Virus), because here there should be either authorizations be used to protect a SAP system or an integrated source analyzing method be used, because ABAP application code is delivered to customers with the source code.

VSI should not be used for structured data, especially where data is transferred into another structured format, e.g. import of CSV files into a database table, based on mappings.

The analysis and protection of (binary) content (mainly documents), exchanged between parties which use SAP in between, and is the target use-case for VSI. Another reason for VSI is the relation between external documents to SAP internal business data, which might be corrupted or lost in case where external security proxies are used in between of SAP systems.

SAP applications always decide where to scan and what to do, so in VSI the scan is always done towards to the external integrated product (so called on-demand scan). The SAP administrator on customer side decides which applications should use VSI and which external product is assigned to. It is a feature of VSI to use several external security products in parallel or for certain applications.

1.1 What Is NW-VSI

The name NW-VSI stands for "SAP **N**et**W**eaver **V**irus **S**can **I**nterface" and relates to the interface between pluggable virus scan adapters and the internal SAP scan API.

This division achieves transparency on both sides, that is, the partner-side is provided by vendors of security solutions, meaning that certified products can be used by SAP without the need to deal with external functions in detail. The SAP side is developed by SAP, that is, the integration of the interface into the individual SAP applications and solutions is performed by SAP, meaning that an external partner does not require any special knowledge about SAP applications.

The vendor of a virus scan adapter does have to be the one who provides the scan product and/or engine. The vendor does not need special knowledge about SAP applications, but knowledge about the SAP platform is needed, e.g. which special files are used in which context on SAP side.

A vendor only need to have a product with specific characteristics certified once for this interface, and does not need to perform any additional integration. This statement applies with one exception with regard to the update procedure (see section 9).

NW-VSI is a "C" interface. Since this is to be certified, SAP AG commits itself not to make any further changes for a fixed period for at least 3 years. A partner must also not make any changes to the definition of the interface while a certification exists. The partner to support his product at least for this period of 3 years.

SAP ensures, that the interface is made available in all SAP products and applications. The integration of VSI in SAP products is part of the SAP Product Standard Security (SAP Library - Secure Programming).

1.2 What Is New in NW-VSI 2.00

The version 1.00 required only the virus-scan functionality, described in chapter 10.1. Thus SAP does not make any statement to the detection rate of the AV engine itself, the certification is done by scanning a test virus (see www.eicar.org).

The version 2.00 requires the content classification and filter, which is described in chapter 10.2. The evaluation of the content type and the return of structure VSA_CONTENTINFO is a MUST in version 2.00. The technical description of this was already available in version 1.00. The data structures in NW-VSI 2.00 are fully compatible to version 1.00.

The enhancements of NW-VSI 2.00 from a security perspective can be reduced to following statements. The protection with NW-VSI 1.00 targeted the clients of a SAP system, e. g. clients which downloads and opens documents from a SAP application, where they were uploaded before by external untrusted clients. NW-VSI 2.00 protects the SAP server itself, because it protects against Cross-Site Scripting (XSS) in files. With XSS an attacker can steal the access data to the server itself.

The features of NW-VSI 2.00 can be activated on SAP side through configuration. The existence of the needed objects in a SAP system are described in Note 1640285 - Determine MIME type with Virus Scan Interface.

The content classification includes:

- Detection of file type and mapping to the corresponding fields in structure VSA_CONTENTINFO
- Ensure that content information cannot be misunterstood in case of mixed file types, e.g. Java Archive Files (*.jar) which technically are ZIP containers.
- Files with invalid content in its structure are detected as error, e.g. pictures or PDF with JavaScript in the first 1024 bytes.
- Files with merged content types must be classified as error, e.g. pictures with JAR content at the end of the file.

The content filter feature stands for:

- Remove or Change of unwanted embedded content
- Block content which violates per-defined content policies

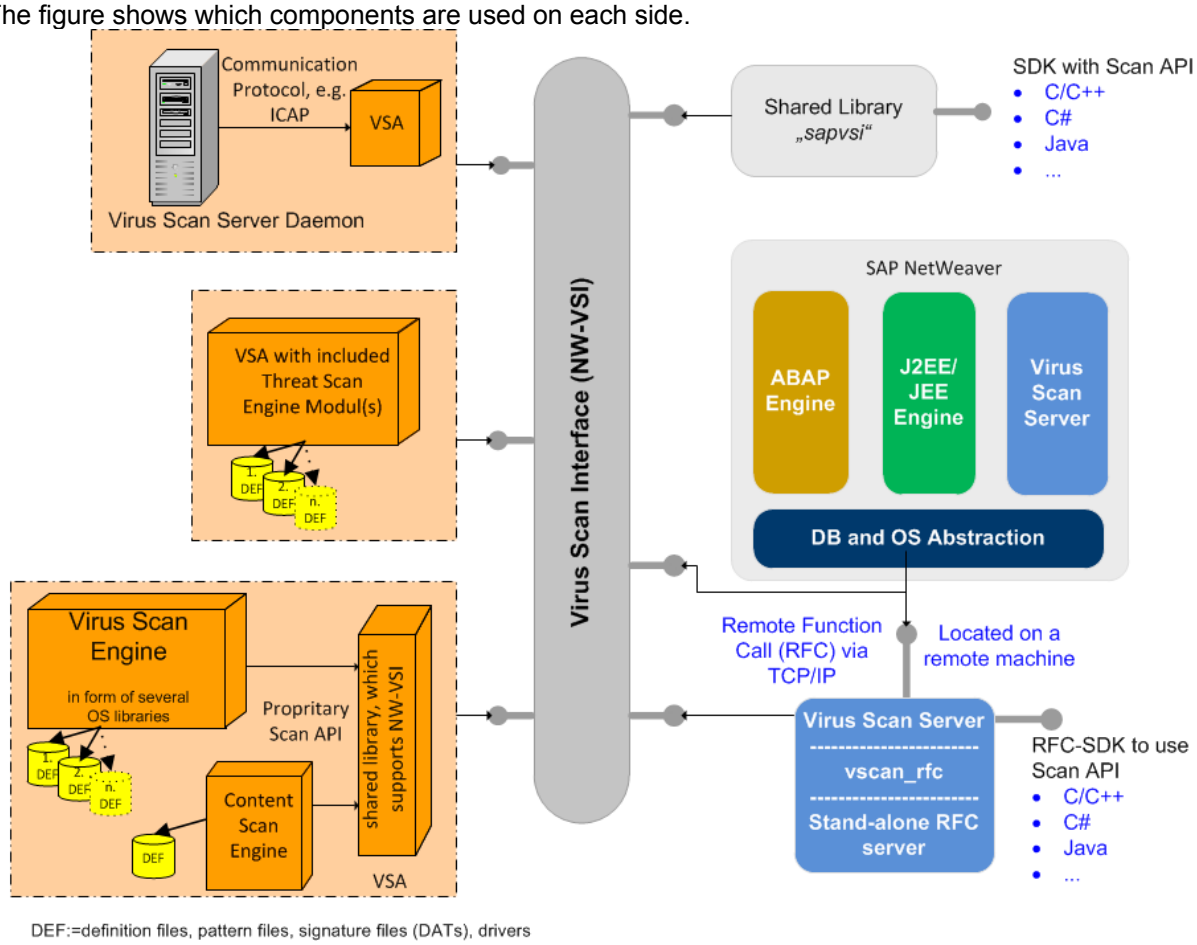
The enhancements in NW-VSI 2.00 increases the effort for the certification process for both sides, because the partners has to agree a certain set of file types which might be different in their understanding. The partner has to integrate files types, which are only known and used inside of SAP applications. Therefore the certification process can only ensure a specific behavior at one certain time, but due to the changes and increasing file types maybe both partners may have to change their software products in the 3 years. SAP ensures this via the patch process and supports external products not only for this 3 years, means the usage of NW-VSI 1.00 compatible products sill is supported with this new interface, however there will be no new products certified with NW-VSI 1.00 only.

The partnership between SAP and its partners for NW-VSI 2.00 therefore needs a closer communication between both development departments.

This closer partnership should enable the partners to learn about the SAP processes on a platform so that security checks beside the NW-VSI defined one are also possible.

1.3 Components of NW-VSI

The figure shows which components are used on each side.



DEF:=definition files, pattern files, signature files (DATs), drivers

Each side is responsible for their own part, that is, SAP AG is responsible for the virus scan adapter (VSA) being used correctly, and that its functions are also implemented correctly in the application systems. SAP AG is responsible for this part of the support.

The operation and support of the VSA and the underlying products, on the other hand, are the responsibility of the partner. This includes, for example, the quality and performance of a virus scanner, and the necessary updates for it. The form in which the partner provides an interface for common customers, as a standalone product, or as an extension of existing security solutions, is left to the partner.

In the following both partner sides are explained. The description about the partner side should show the possibilities. The SAP side has been extended in the past also because of the usage of VSI outside of SAP NetWeaver. The integration is focused on following aspects:

1. Native integration into SAP application process using the Virus Scan Adapter. This is available for the application servers ABAP and Java. Applications which does not run on NetWeaver use the shared library “sapvsi”. All these servers run in a 64 bit process environment, therefore a partner library must provide this architecture.
2. Local integration with a local installed Virus Scan Server (RFC server: vscan_rfc). This is needed in case the partner product is available on the OS of the SAP system, but not in the same process architecture, e. g. often means not available in 64 bit.
3. Remote integration with a local and/or centralized Virus Scan Server. This is needed in case the partner product cannot is not available for the OS of the SAP system. There could other reasons for this use-case, see section 1.4.2.2 Virus Scan Server.

1.4 Integration of the Components

1.4.1 Partner Integration

In NW-VSI 2.00 the partner product as to full-fill both integration scenarios: a purely virus protection solution and an implemented content security solution. The program interface for these scenarios is identical. The two scenarios cannot be certified independently of each other.

The certification partner for NW-VSI 2.00 can be a certain AV software vendor and/or an independent software vender (ISV).

Therefore the integration on the partner side can be performed technically in the following ways:

- Native, static integration into an existing scan engine
- Provision of a separate dynamic library, which communicates with an existing engine using internal API´s or protocols. This connection can mean that the library itself loads the engine as a dynamic library or also communicates with a separate process using COM/IPC, shared memory or (local) sockets.

  Example: The virus scan adapter communicates with an external scan daemon using pure TCP/IP or ICAP.
- Using Cloud Services. Already mention in example before the adapter can communicate with a remote scan server. Thus there are existing scan services are available as so-call cloud service, this can be used, too.
- Adapter library invokes the external scan with a command call / new process execution.

  Example: The AV product does not offer a API to perform the scan but provides a command line tool where all needed data are transferred, then the virus scan adapter can use this command line tool.
- Combination: An integration can combine different approaches, thus are local scan using another scan engine as shared library and a remote scan server in parallel.

The architecture for this is left to the partner, however, the partner must also provide and maintain this architecture and components, whether the parts belong to the partner itself or to other software vendors.

A productive solution MUST also include documentation of all related components on partner side. Partner own installation procedures are recommended, however SAP will provide here tools and help, too. The installation of the partner software was in the past one of the pain-points for customers, because the AV software needs local OS administrative rights and at the end a reboot of the machine. SAP system landscapes are often decoupled from the OS landscapes.

In addition to the settings through the interface, an external scan engine can also have its own configuration and administration tools. The partner configuration must not override any SAP application configurations, but can be used for default settings as well as for features which are not available through the SAP configuration.

There are three areas where a partner could provide a security solution.

1.4.1.1 Virus Scan Interface

The virus scan interface is the basis for the certification, that is, it is mandatory that a partner product contains this part. The minimum prerequisites for a certification are described in section 10 (10 Certification Criteria).

The quality of a virus scan, that is which and how many viruses are found, is not checked, since this does not lie in the competence of SAP. Decisions about a security solution using its quality should be left to the customer. For certification, only the interface definition needs to be fulfilled, and a number of test files used by SAP need to be scanned. These files contain only the “Eicar test virus“ (http://www.eicar.org/anti_virus_test_file.htm) developed by EICAR[1].

The certificate contains only information about the performance of the scan. However, time limits can be set in individual cases.

1.4.1.2 Content Classification Interface

This integration interface is now a must for a certification. The extension of the NW-VSI 2.00 interface with a content scan and classification has two different purposes within the SAP integration, and technically usually requires no extra effort on the partner-side. During a virus scan, a number of analysis with regard to the content to be scanned are usually performed first, and the content is searched during the scan in accordance with specific rules.

The transfer of the detected content type to the SAP application is one new aspect. The correct content type is such important because if an application simply trusts the type which a clients provides to the application or which is retrieved from metadata as from a file extension unwanted execution of code can be the result. The short term for this is XSS, which is always the case when untrusted JavaScript code from an attacker is executed in contexts where it should not be executed. XSS can be prevented with the a correct output encoding framework, e. g. (SAP Cross Site Scripting Prevention Library). JavaScript in files can not be prevented with standard encoding, therefore VSI was enhanced. In many cases the correct usage of content an a defined context prevent these problems, therefore the first aspect of NW-VSI 2.00 means the return of the detect content type.

The other aspect of NW-VSI 2.00 is the filter mechanism which can be used as white- or black list filter based on pre-defined content types. In case the passed content does not fit to the wanted content the partner return the check with a infection information. The SAP applications treats this as virus infection, comparable to classical virus infections and prevents the transfer of this content to other parties (blocks the contents). The partner product also can remove the unwanted content and return the so-called cleaned content back to SAP. The configuration of SAP applications defines whether a check is wanted or the remove of the unwanted content is needed. The aspect of the content filter is for all applications which does not allow the upload of arbitrary documents, but allow one content type as for example in a social media application the upload of the user picture. The protection is given if the content type is ensured and the format structure is not changed, e. g. script code inside the picture.

1.4.1.3 Web Content Filter Interface

This integration is not part of NW-VSI 2.00. The usage scenario for this is textual data as it is used in web applications mainly. An integration based on this interface is only running in the SAP web server process. It ensures that request URIs and data in web formula’s are free of JavaScript in order to prevent XSS.

Beside the standard protection against XSS there are policy rule definitions possible which are based on regular expressions. Such a solution can be used to control the access inside of web resources. SAP delivers here an integration already, see Checking User Input for Program Commands. However if a partner offer a solution for this interface a validation of this can be done by SAP.

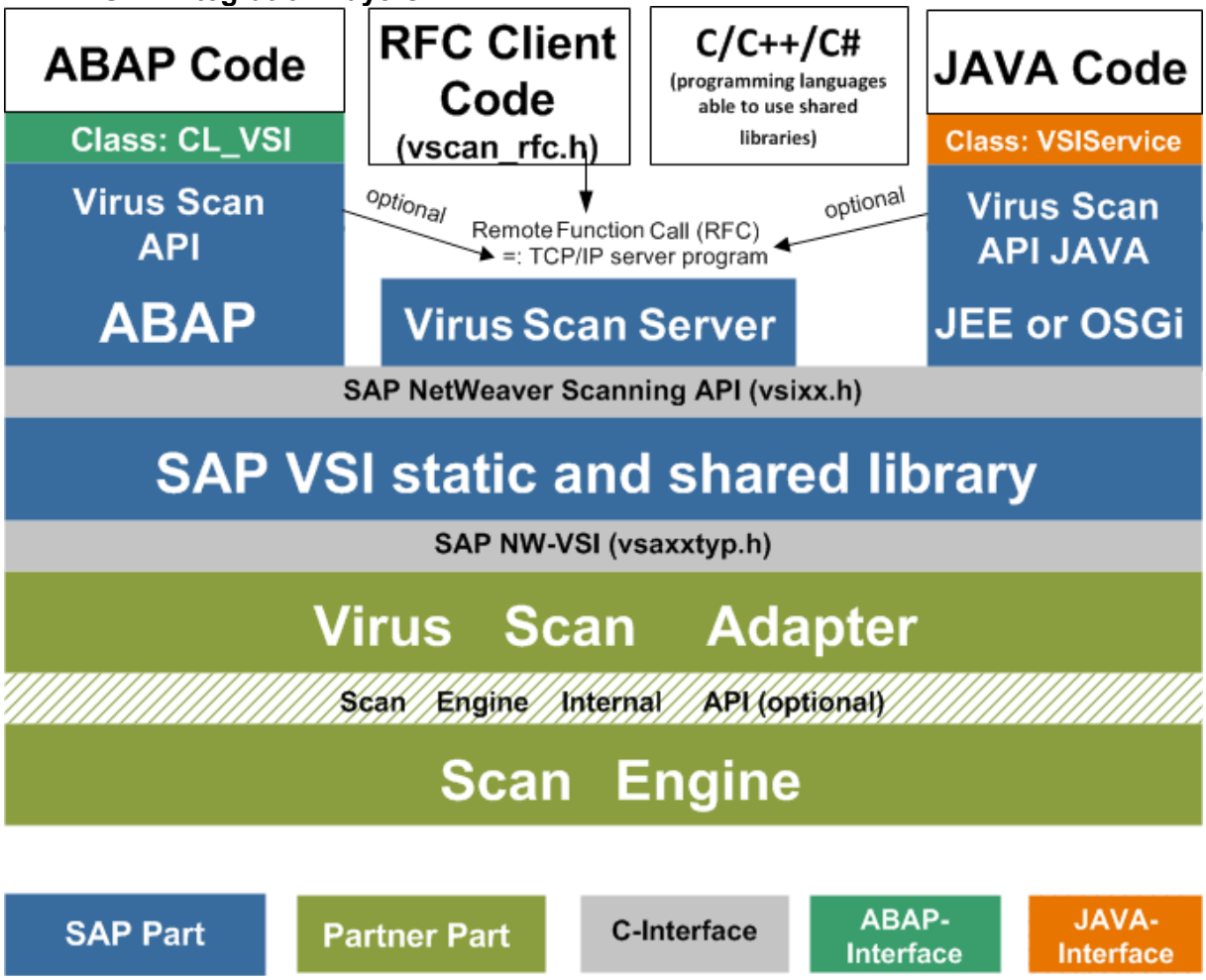
An example of the use in this case would be a content filter in the Web framework of SAP applications. As with the virus scan, the integration work for a filter of this type would be SAP’s responsibility.

[1] EICAR (http://www.eicar.org) is an association of different institutions with the aim of protecting the privacy of individual.

1.4.2 SAP Integration of NW-VSI

The integration of the partner components, technically the virus scan adapter into the various SAP applications should show how the interface is used in the current release of SAP NetWeaver. Extensions or changes in later releases that do not affect the described interface definition are, however, possible without re-certification.

1.4.2.1 SAP Integration Layers



It is easier to explain the SAP integration with the figure. It shows, how SAP integrates the virus scan adapter of a partner into the applications itself. SAP application solutions are mainly based on the SAP NetWeaver application platform today, however, even here different sub-components, such as the SAP Content Server are utilized. Due to this complexity, separate integration guarantees the best-possible use. For more information about SAP NetWeaver, see the SAP Help Portal at http://help.sap.com/, or for information about the SAP NetWeaver 7.31 release, see http://help.sap.com/nw731.

The figure also points out, that SAP integrates VSI itself based on C libraries and provide the API´s for other languages with own abstraction implementations, e. g. JNI layer for Java. Another abstraction for client usage is the Virus Scan Server. This can be used based on the public RFC-SDK.

1.4.2.2 Virus Scan Server

The Virus Scan Server is delivered by SAP as part of SAP NetWeaver. It provides scan services using SAP's own communication protocol RFC. The application options and configuration of this scan server can also be read through the SAP Help Portal http://help.sap.com/ → SAP NetWeaver → Security → System Security → Virus Scan Interface.

The separate server process is primarily used if the partner product is not available for the same operating system environment under which the customer is operating SAP NetWeaver. For more information about SAP's platform matrix, see the SAP Service Portal[2] at http://service.sap.com/pam.

The usage of the Virus Scan Server on remote servers has limitations. The certification for a certain operation system is not possible if the partner is not able to provide a virus scan adapter for this operating system but integrates the product with a remote installed Virus Scan Server. The SAP note 782963 describes the limitations. However the use of a remote virus scan server can make sense in cases where VSI should be used for a specific application area only, e. g. SAP E-Recruiting to check the uploaded documents of the applicants.

The usage of the remote Virus Scan Server scenario is described in SAP note 964305.

1.4.2.3 SAP NetWeaver ABAP

SAP NetWeaver is the platform on which SAP applications run, operating system-independently. It supports both ABAP and Java and Web services. The conversion of the data from both run-time environments is performed in the respective internal scan API. For more information about SAP NetWeaver and its architecture, see the SAP Help Portal http://help.sap.com/nw. The integration into this runtime needs a 64 bit virus scan adapter, because only 64 bit ABAP server platforms are in the market.

The partner integration can be configured in transaction VSCAN, see SAP documentation: Virus Scan Interface

1.4.2.4 SAP NetWeaver JAVA

SAP Application Server JAVA nowadays can be a JEE complaint server or a OSGi server. The J2EE or JEE complaint server always has the JNI abstraction of SAP VSI inside the environment and provides a full administration UI. The integration into this runtime needs a 64 bit virus scan adapter, because only 64 bit server platforms are available.

The base of SAP Cloud products are OSGI complaint server runtimes. SAP supports this environment with a stand-alone JAR (COM.SAP.SECURITY.NW.VSI.JAR), which is part of the VSA-SDK. The JAR contains the native layer inside the shared library "sapvsi" and is automatically used when initializing the VSI framework.

1.4.2.5 SAP Applications Integration

The integration into SAP applications, means integration into SAP E-Recruiting or SAP Content Server, is provided to the usage of so-call VSI Profiles. The profile in context of VSI stands for:

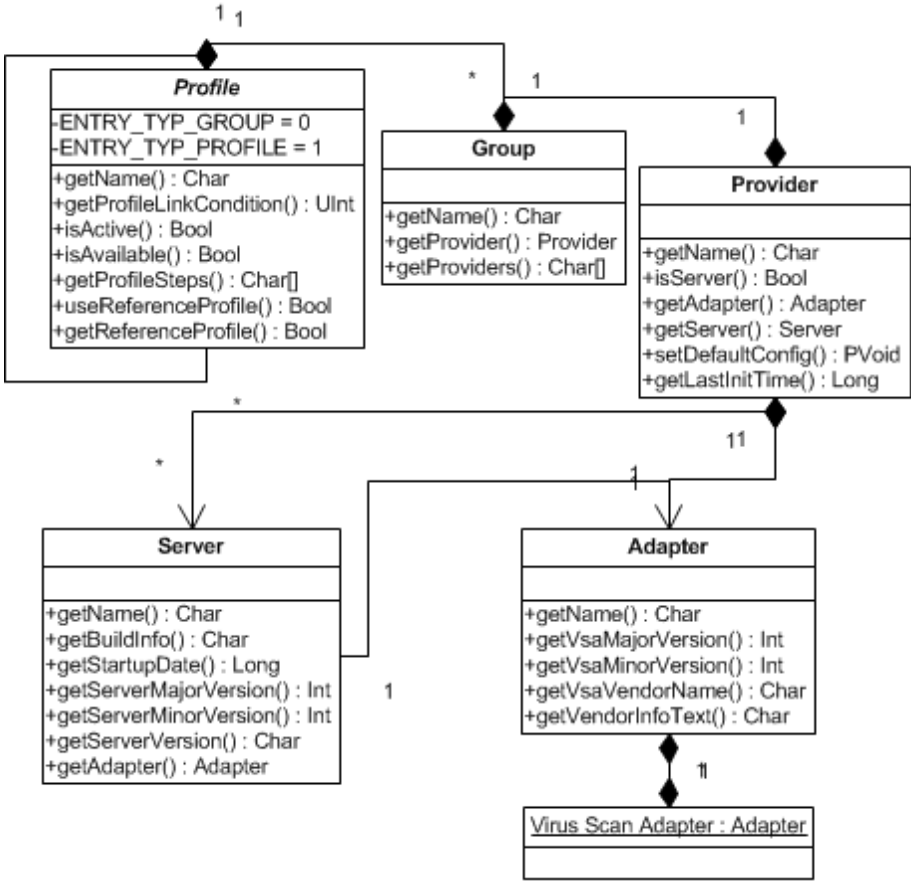
- Switch for usage of VSI inside the application. Default SAP delivers deactivated profiles.
- Meta data configuration for the action. Defines which adapter is used with which settings.

The profile is an abstraction entity, which was introduced because of two reasons:

1. SAP does not deliver a per-configured scan product and therefore by default no scan is done
2. The applications defines in which cases a scan is need, but the administrator defines the security behavior of the scan. The profile defines if AV, Content Scan or Content Filter is needed.

[2] The SAP Service Portal requires access data. SAP customers have this access through their service contract with SAP.

The following figure shows all related entity in this framework.



The profile defines a list of used profiles and groups. This allows a combination of several security products in parallel and it allows to define a rule like: pass an object only if product A AND product B scan is successful.

The entity group is primarily need in case Virus Scan Servers are used, because of the usage of TCP/IP the group provide a load balancing mechanism. The group defines a list of used providers and the group is a label for a “product”.

The entity provider is an abstract term for either a Virus Scan Server or a Virus Scan Adapter.

1.5 Certification of NW-VSI

The certification of the interface between SAP and a partner is legally recorded by a certification contract. Before a certification of this type, however, the technical prerequisites should have been met on the partner’s side. SAP provides information about partner integration for this purpose at http://www.sap.com/icc. To start the process, please contact icc@sap.com and ask for a partner contract.

All certified partners for the NW-VSI interface are stored in a database and can be viewed in the case of customer queries. SAP Note[3] 1494278 contains all known and certified partners.

For information about the current version of the interface, see the above SAP ICC web pages. The current version is **NW-VSI 2.00** (see section 0 History and Version). This is announced in SAP Note 1796762.

[3] SAP Notes are read by SAP customers in the case of problems and questions and are part of the official product documentation. They can be viewed through the SAP Service Portal.

1.5.1 VSA SDK

The virus scan adapter (VSA) SDK can be downloaded through the internet location ftp://ftp.sap.com/pub/icc/nw-vsi or is available on request to icc@sap.com. It allows the development and testing of a virus scan adapter without the need to have an SAP product installed yourself.

This specification is part of the SDK. There are two different versions, a basic version and an extended version. The difference between them is additional information about the SAP archive format (see chapter 8 Scanning SAP Archive Formats), which is contained in the extended version, but which can only be made available if the partner contract for a certification is signed.

The following components are also contained in the SDK:

- VSA Test Suite
  - VSATEST program, which tests the interface. This command line program is available for all SAP NetWeaver platforms. It imports various test descriptions in XML syntax and then calls the virus scan adapter. The program itself runs as a standalone without a database or SAP application component.
  - COM.SAP.SECURITY.NW.VSI.JAR. This Java package contains the VSI framework for all stand-alone java applications and OSGI related applications.
  - VSATEST program documentation
  - VSATEST XML files with test descriptions
- VSA header file VSAXXTYP.H and help files VSA.HTML and VSA.HLP generated from it
- VSA example adapter VSSAP, which finds only the EICAR virus

Before making an appointment for certification, the VSATEST program should be reporting no errors at the end of the test.

1.5.2 Certification Process

The following steps are performed during a certification:

1. Installation of the external product in accordance with instructions provided
2. VSATEST – Test run with various test descriptions.
3. Java based Test using the standalone JAR (com.sap.security.nw.vsi.jar)
4. Setting up of the virus scan interface under ABAP. The configuration is performed, and the parameters are set, in accordance with the partner documentation.
5. Setting up of the JEE Service “Virus Scan Provider“, also in accordance with the partner documentation.

The exact process of the certification can also be seen in the NW-VSI test plan.

2 Glossary

328 ***SAP NetWeaver:***

329 ***Also abbreviated to NetWeaver, this is the basis for SAP solutions***
330 ***on any given hardware. The business applications use the key areas***
331 ***of SAP NetWeaver.***

332 ***API:*** ***Application Programming Interface***

333 ***AV:*** ***Anti Virus***

334 ***CPI-C:*** ***(Common Programming Interface for Communications) – interface***
335 ***for communication between different systems.***

336 ***RFC:*** ***Remote Function Call. An SAP communication protocol based on***
337 ***CPI-C, which is used to execute functions on remote hosts.***

338 ***SDK:*** ***Software Development Kit***

339 ***SDN:*** ***SAP Developer Network (http://sdn.sap.com)***

340 ***VSI:*** ***Virus Scan Interface is the name of the project and the generic term***
341 ***for the architecture of the VSA and VSILIB.***

342 ***NW-VSI:*** ***This is the abbreviation for SAP NetWeaver Virus Scan Interface and***
343 ***is the name of the certifiable interface to the VSA.***

344 ***NDA:*** ***Non-Disclosure Agreement.***

345 ***MIME:*** ***Multipurpose Internet Mail Extensions. The internet standard based***
346 ***on RFC2045 describes the format of content.***

347 ***VSA:*** ***Virus Scan Adapter is the link between an external, proprietary prod-***
348 ***uct and SAP scan interfaces. This can be, for example, a separate li-***
349 ***brary or an engine extended with the required calls.***

Virus Scan Server:

An executable program that includes VSA from certified vendors through the NW-VSI interface and provides scan services to an SAP application server as a registered RFC server.

VSILIB: ***The internal static library that is to be made available within SAP to application developers as an API and which in turn uses the NW-VSI interface to external products.***

SAPVSI: ***The shared library which includes internally VSILIB. The shared library provides the scan API to all applications which are not based on SAP NetWeaver.***

DEF: ***This abbreviation is used in this document for definition files and refers to the virus signature files that are used by AV scanners to identify viruses. Almost every vendor of AV products uses a different term for this.***

DRIVER: ***The technical term for DEF for the VSA specification.***

POSIX: ***(Portable Operating System Interface for Unix). Interface between applications and the UNIX operating system. This standard has become a global industry standard (DIN/EN/ISO/IEC 9945) and is also supported as of Microsoft Windows NT by the NT-compatible Microsoft® operating systems.***

REGEX: ***(RE⇔Regular Expression) There are two variants of regular expressions in accordance with the POSIX Standard 1003.2: basic and extended versions. This language allows the definition of complex search patterns.***

3 Rough Design

3.1 Data Definitions

3.1.1 Character Set

When transferring text parameters, the internal type VS_TYPE_CHAR must be defined as UTF-8 for a certification. The use of UTF-8 (=: unsigned char) applies both for the transfer of text data to VSA functions and for the return. There is an exception for this rule (using UTF-8), in case a local file should be scanned. The low level API on the operating systems treat the file name without encoding, but binary, therefore the filed pszObjectName in structure VSA_SCANPARAM is passed without using UTF-8.

In all other cases this field (pszObjectName) is set in UTF-8.

The data type VS_TYPE_BOOL was defined as “unsigned char”, since, in principle, a C++ interface was not assumed; rather, care was taken to conform to ANSI C as far as possible.

Both SAP and the partner-side are responsible for any required conversion to other character sets (UTF-16, UCS-2, UCS-4, and so on).

Remark: In case a SAP system runs in Unicode, SAP uses internally UTF-16.

3.1.2 Syntax for Search Rules

Search rules are needed for the interface described in 1.4.1.3 Web Content Filter Interface.During the transfer of regular expressions for the content scan, the syntax rules from POSIX Standard 1003.2 (REGEX) extended version are defined. This is needed in parameters

- VS_IP_INITCONTENTPATTERN
- VS_IP_INITREPLACEPATTERN

3.1.3 MIME types

The MIME types has to be compatible to the standard RFC´s 2045, 2046 and 2077. This definition will be used in the parameters

- VS_OP_SCANMIMETYPES
- VS_OP_BLOCKMIMETYPES

and in structure VSA_CONTENTINFO, field pszContentType.

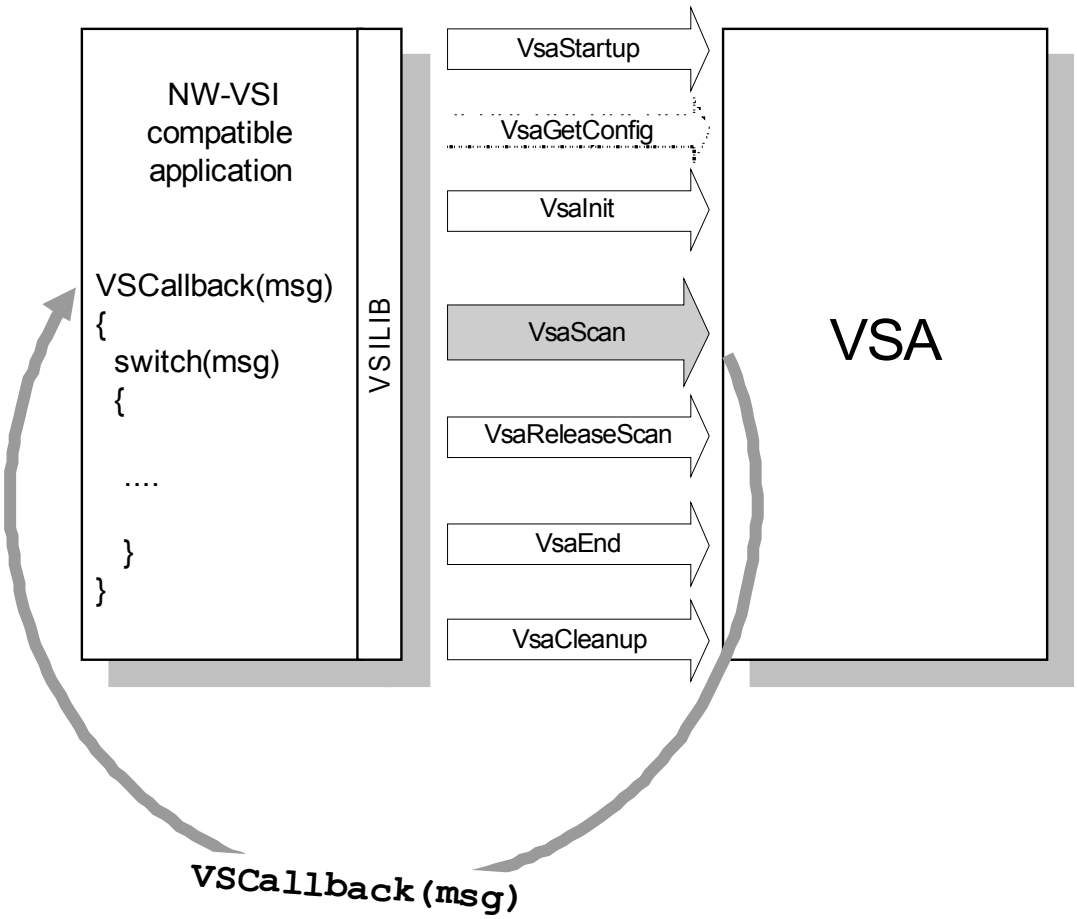
3.1.4 Memory consumption

The adapter allocates memory for various structures. This memory is treated to be allocated from heap, therefore there are functions, which allow to release this memory. If a partner uses Shared Memory (e. g. in case of using IPC), then this has to be adressed to SAP before certification.

3.2 Programming Interface of the Virus Scan Adapter (VSA)

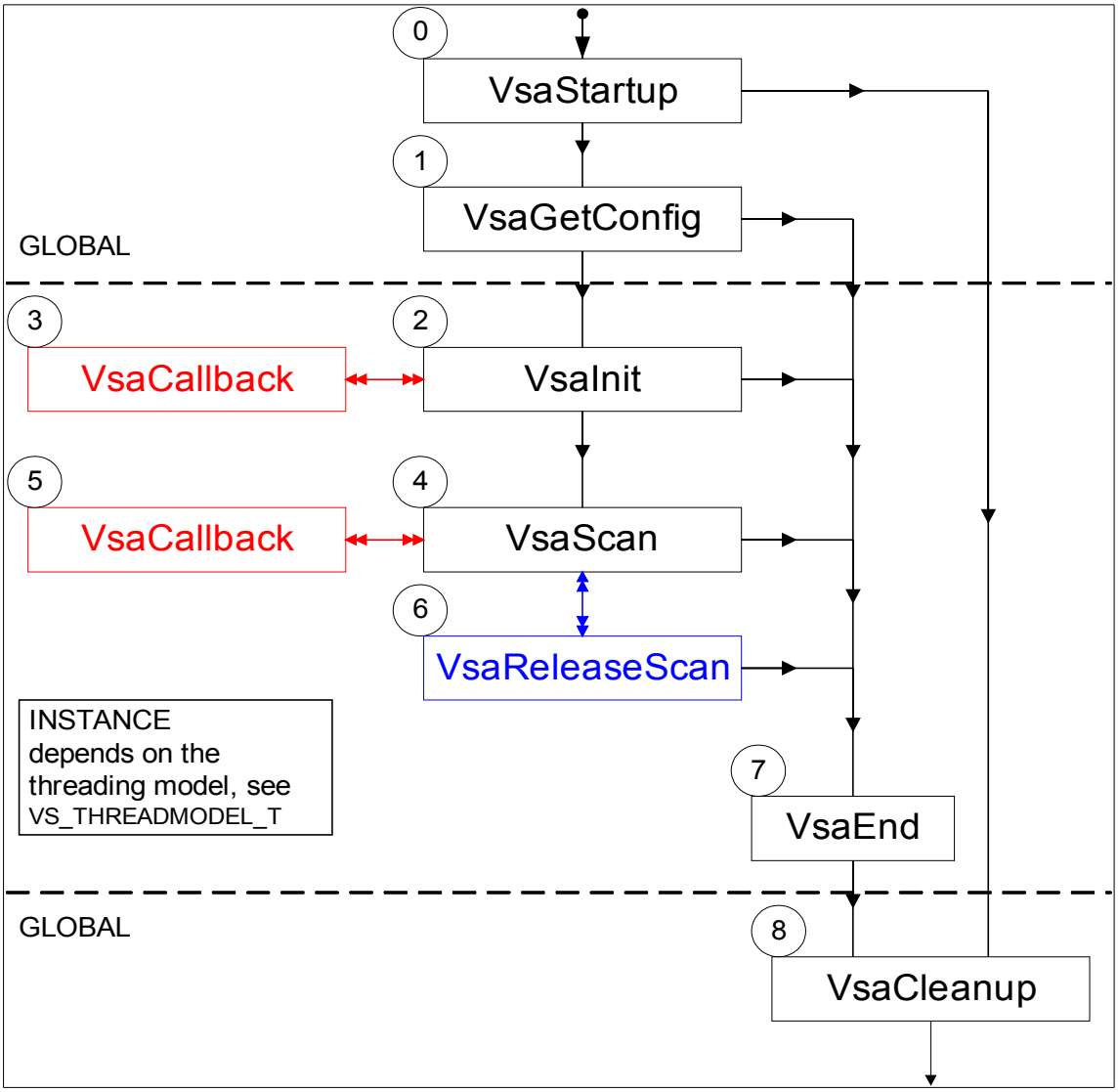
Based on the preceding descriptions and objectives, the following interface is specified as an interface to external products through a VSA:

- `VsaStartup ()`
- `VsaGetConfig (VSA_CONFIG **)`
- `VsaInit (VSA_CALLBACK*, VSA_INITPARAMS*, VSA_INIT **)`
- `VsaScan(VSA_INIT*,VSA_CALLBACK*,VSA_SCANPARAM*,VSA_OPTPARAMS*,VSA_SCANINFO**)`
- `VsaReleaseScan(VSA_SCANINFO**)`
- `VsaEnd(VSA_INIT**,VSA_CONFIG**)`
- `VsaCleanup ()`



3.3 VSA Functional Diagram

The VSA functions are divided into a process-global and thread-local unit (instance). The calls ***VsaStartup*** and ***VsaCleanup*** are the two global functions. At this point, an adapter can (if this is necessary for a partner product) either initialize its global values or in close all open actions in VsaCleanup. If a partner product does not require any global data, the two functions must return at least the return value VSA_OK. An adapter should protect itself without global initialization before the call or can rely on the fact that ***VsaStartup*** is the first function after the library is loaded, and "VsaCleanup" is the last before an unload. The call of ***VsaGetConfig*** has a special status. ***It must be possible to call this function at any time after the global initialization, and it must return a filled VSA_CONFIG structure.*** All of the resources of the VSA are dynamically allocated. This means that it is necessary to release the occupied memory again. The functions **VsaReleaseScan** and **VsaEnd** are used to release an "instance handle".



3.4 Supported Thread Models

After global initialization using VsaStartup, the scan instances can be initialized. This is done in thread-local functions. The thread-local functions that are called depend on the supported “threading model”; that is, if the VSA is thread-safe with regard to multiple calls of ***VsaScan***, it can specify this using the VSA_CONFIG structure. The internal SAP scan API then generates only additional references to a VSA scan instance that has been initialized once for each internal scan instance. If the VSA is not thread-safe, VsaInit() is used to create a separate instance for each new thread, and these are deleted again at the end of the thread using VsaEnd.

This means that different VSA implementations should be able to work with different SAP environments, because the implementations use their own configuration at runtime. The following models are supported:

- **VS_THREAD_APARTMENT**

  Also known as single thread apartment (STA). The VSA supports “only” single threaded instances. If the calling application is multi-threaded, however, the caller must create a separate instance for each thread using VsaInit.

- **VS_THREAD_BOTH**

  The VSA supports both variants. In this case, the SAP application layer can decide whether an instance should be created for each thread or whether the same instance handle should be used for each call in VsaScan.

- **VS_THREAD_FREE**

  Also known as multi-thread apartment (MTA). The VSA supports “only” multi-threaded instances. The SAP application API calls VsiInit once and must always use this instance handle with VsaScan.

The three types are taken from the “Component Object Model“ (COM) architecture, which also has to solve the problem of different partners (single-/multi-thread).

This abstraction is needed, because of SAP applications run in different process environments:

- Single Threaded, but Multi Processes (ABAP runtime)
- Multi Threaded (Java stand-alone runtime)
- Multi Threaded and Multi Processes (J2EE runtime)

3.5 Supported Return Options

In addition to the function result, the VsaScan function has two optional return options: the information structure VSA_SCANINFO and event messages that are returned to the caller using a callback function.

This possibility of **optionally** receiving a detailed information structure and **optionally** receiving callbacks during a scan action allows the user of the internal SAP scan API to decide whether to use a simple, and therefore faster, scan run, or whether more detailed information is to be displayed in each case for any infections or scan errors. It is also possible to include different VSA architectures through the optional use of information structures or support for CALLBACK messages without losing information about a scan run.

Possible Combinations for Evaluating Scan Results:

1. VSA supports CALLBACK and VSA_SCANINFO structure:

Callbacks are made to the caller, and the VSA_SCANINFO structure is filled after completion of the scan.

Call:

```
VsaScan (VSA_INIT*,VSA_CALLBACK*,VSA_SCANPARAM*,VSA_OPTPARAMS*, VSA_SCANINFO **)
```

Scan results can be queried using the return code, the callback function, and VSA_SCANINFO.

=> Function calls in the functional diagram (see 3.3): 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8

2. VSA supports CALLBACK but no VSA_SCANINFO structure:

The caller receives callbacks, but does not want to receive additional VSA_SCANINFO after completion of the scan.

Call:

```
VsaScan(VSA_INIT*,VSA_CALLBACK*,VSA_SCANPARAM*,VSA_OPTPARAMS*, VSA_NO_SCANINFO)
```

Scan results can be queried using the return code and the callback function.

=> Function calls in the functional diagram (see 3.3): 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 7 -> 8

3. VSA does not support CALLBACK but does support a VSA_SCANINFO structure:

The caller does not receive any callbacks, but wants a filled VSA_SCANINFO after the scan is completed.

Call:

```
VsaScan(VSA_INIT*,VSA_NO_CALLBACK, VSA_SCANPARAM*,VSA_OPTPARAMS*,VSA_SCANINFO
**)
```

Scan results can be queried using the return code and VSA_SCANINFO.

=> Function calls in the functional diagram (see 3.3): 0 -> 1 -> 2 -> 4 -> 6 -> 7 -> 8

4. VSA does not support CALLBACK or a VSA_SCANINFO structure:

The caller does not receive any callbacks and also does not want a filled VSA_SCANINFO after the scan is completed.

Call:

```
VsaScan(VSA_INIT*,VSA_NO_CALLBACK,VSA_SCANPARAM*,VSA_OPTPARAMS*,VSA_NO_SCANINFO)
```

Scan results can only be queried using the return code. For more information, see 4.1 Return Values.

=> Function calls in the functional diagram (see 3.3): 0 -> 1 -> 2 -> 4 -> 7 -> 8

Note:

The constant **VSA_NO_SCANINFO** is used when calling VsaScan to provide a clearer visual indicator that no filled VSA_SCANINFO structure is to be filled. It would be just as possible to enter a NULL here for the transfer of the handle for VSA_SCANINFO. (see 4.11)

3.6 CALLBACK Interfaces

The use of the callback interface is not compulsorily defined in the specification of the VSA. It can also be achieved, if necessary, by transferring a function pointer to its own callback function, and by specifying the desired message types in the structure VSA_CALLBACK. In this case, the adapter should check the existence of the function pointer.

Support for the callback interface is also not mandatory. If no CALLBACK functions are supported by a VSA, it must show this by returning 0 in the corresponding VSA_CONFIG parameters, "`uiVsaEvtMsgFlags`" and "`uiVsaCIOMsgFlags`".

3.6.1 Client Input/Output (I/O)

- `<VSA_CIOCBFP> (VSA_ENGINE*,VS_IOREQUEST_T,VOID*,size_t, size_t*)`

This callback function delegates input and output operations to the calling application. This allows the importing of (additional) drivers in the "`VsaInit`" function, and the delegation of the read and write operations during scan runs to the calling application in the "`VsaScan`" function.

In addition to the engine handle, the VSA transfers a request number, a buffer, and information about the size of this buffer. The application must return the size of the used buffer in the last parameter, such as how many bytes were written.

The VSA must keep calling the respective CALLBACK function until it receives **VS_CB_EOF**. This avoids the blocking of the requests. The application can therefore also return 0 for the last parameter of the function.

Example:

If I/O CALLBACK is used to delegate the read operations to an application that itself receives data at a network socket and transfers it to the buffer of the function, a VSA must keep calling the callback function to read data until it receives VS_CB_EOF as a return code. The application must not block here, if there is currently no data for collection. The VSA should call this function using a loop with a wait interval.

3.6.2 Events

- `<VSA_EVENTCBFP> (VSA_ENGINE*,VS_MESSAGE_T,VSA_PARAM*,VSA_USRDATA*)`

This callback function acts as a direct communication interface of the VSA using "`VsaInit`" or "`VsaScan`". In this way, specific events during processing can be directly transferred back to the caller. This allows, for example, an immediate display of scan results or errors in a separate program trace or a progress display when scanning larger requests.

The possibility of a callback also allows interactive intervention during a scan request, however, such as the termination of larger scan requests or explicit queries as to whether infected objects are to be cleaned.

The callback function must be implemented by the calling application (VSILIB), which must also process its events. To restrict the number of queries to the desired quantity, the **uiMsgFlag** value in the VSA_CALLBACK structure can be used to inform the VSA the events for which a callback is to be made. When doing so, multiple messages can be transferred using an OR linkage. The constant **VS_M_ALL** must be understood by every VSA and is used to transfer all messages implemented in the VSA.

In addition to the engine handle, a callback function also transfers the message number (VS_MESSAGE_T), a pointer to a specific information structure (such as VSA_VIRUSINFO, VSA_SCANERROR, or VSA_CONTENTINFO) using VSA_PARAM (<=> void*) and a pointer to user-specific data. The latter must also be set using a previously set parameter and in this case allows the transfer, for example, of a communication or window handle to transfer the transferred data directly back to a partner or application program.

The permitted return values in a callback routine depend on the message type (see section 4.1.2). **Unknown values must be interpreted as VS_CB_TERMINATE and lead to the termination of the action.**

3.7 Parameter Transfer

To treat different AV products equally when connecting through the VSA, the transfer of parameters was generically defined in following types:

- Initialization parameters

  These parameters define the environment for the external product. One or more parameters in an array that are required to create scan instances. For example, a directory that contains the signature files for the engine or, in the case of a scan daemon, the information about servers and ports.

- Scan parameter

  The structure VSA_SCANPARAM defines what to scan. This parameter is transferred to the VSA in function VsaScan. It defines the properties for a scan action.

- Optional scan parameters

  One or more parameters in an array, which optionally define the action in more detail, such as specifying whether or not a compressed object is to be unpacked.

- Optional content scan/block parameters

  One or more parameters in an array, which specify the policies for the adapter to block the content. The return code for this is VSA_E_BLOCKED_BY_POLICY.

These three types should be used to inform the VSA implementations which action is to be executed, and also to provide any additional parameters. The transfer of different parameters using arrays allows you to transfer any number and type of parameters, in a similar way to an argument array for the call of command line programs.

The **VsaGetConfig** function was introduced to solve the "*problem of different VSA implementations*". When it is called, this function must return all known and supported initialization, scan, and optional parameters. Default values for he VSA concerned can be defined at the same time. When setting parameters in the internal SAP scan API (VSILIB), you can therefore immediately determine whether a VSA knows this desired parameters, or permits this parameter type.

573 3.8 Scan Function VsaScan

574 **One** of the following transfer formats for the object to be scanned must be selected for VsaScan:

Object Type for Scan	Value	Description
VSA_SP_BYTES	0x00000002	An area in memory of a defined number of bytes in length.
VSA_SP_FILE	0x00000004	A single file in the local file system
VSA_SP_DIRECTORY	0x00000008	A complete directory in the file system (with sub-directories, if specified with optional parameters)
VSA_SP_HTTP_HEADER	0x00000100	HTTP header only
VSA_SP_HTTP_BODY	0x00000200	HTTP body only. The optional encoding of the body is passed to adapter in field pszObjectName of structure VSA_SCANPARAM
VSA_SP_HTTP_URI	0x00000400	URI for a web request
VSA_SP_HTTP_MESSAGE	0x00000800	Complete raw HTTP message
VSA_SP_MAIL_MESSAGE	0x00001000	Complete raw Mail message

575 Only one value can be specified for this parameter for each call of "`VsaScan`". Specifying bit values
576 should allow a query of known "Features" using the "`VsaGetConfig`" function. In this way, for exam-
577 ple, a VSA can specify whether it can scan directory structures directly. A VSA should specify all sup-
578 ported object types here using an OR linkage.

579 **One** of the following actions must be specified for processing VsaScan:

Action Type for the Scan	Value	Description
VSA_AP_CHECKMIMETYPE	0x00000001	Checks whether an object can be scanned. The information about the object must be set in the structure VSA_CONTENTINFO
VSA_AP_SCAN	0x00000002	Scans an object for viruses.
VSA_AP_CHECKREPAIR	0x00000004	Scans an object and checks, if it is infected, whether it can repaired.
VSA_AP_CLEAN	0x00000008	Scans an object and repairs it if it is infected with a virus.
VSA_AP_BLOCKACTIVECONTENT	0x00000010	Scans an object for script embedded in the object itself and return with an error if found.
VSA_AP_REMOVEACTIVECONTENT	0x00000020	Scans an object and remove the active content embedded. This can therefore be regarded more as a filter function.
VSA_AP_SCANCONTENT	0x00000040	Scans an object in accordance with specific, transferred rules. The rules are set in VS_IP_INITCONTENTPATTERN.
VSA_AP_REPLACECONTENT	0x00000080	Scans an object in accordance with specific rules and replaces the content when doing so. The rules are set in VS_IP_INITREPLACEPATTERN.

580 The same rule applies for this action parameter as for the object parameter: it was assigned with bit
581 values and allows the transfer of supported call actions of a VSA in "`VsaGetConfig`".

4 Detailed Design

In the following, the design described is supplemented by concrete data descriptions. External partners require a description of the data structures used to be able to develop a virus scan adapter. To create a certifiable adapter, the header file **VSAXXTYP.H** is required. The structures and data types definitions are described here.

4.1 Return Values

There are two different types of return values: those returned after a VSA function call and those that evaluated during the processing of the callback function:

4.1.1 Return Values in VSA Functions

Every function in the VSA **must** return a return value of the type VSA_RC. In the case of the callback function, on the other hand, note that there are separate return values for this and if values are returned that do not correspond to one of these, the running action is terminated and "`VsaScan`" returns the return code **VSA_E_CBC_TERMINATED** (in < VSA_EVENTCBFP>) or **VSA_E_CIO_FAILED** (in < VSA_CIOCBFP>).

Return values of VSA functions:

Return Value of Type VSA_RC	Value	VSA Function	Description
VSA_E_BLOCKED_BY_POLICY	-6	VsaScan	At action: VSA_AP_CHECKMIMETYPE VSA_AP_SCANCONTENT, **at least** one match with the rule set was found.
VSA_E_CLEAN_FAILED	-5	VsaScan	At action: VSA_AP_CLEAN VSA_AP_REMOVEALLMACROS VSA_AP_REPLACECONTENT The repair or replacement of **at least** one virus infection or matches or the removal of **at least** on macro (if the action VSA_AP_REMOVEALLMACROS is specified, for example) failed or was not possible.
VSA_E_PATTERN_FOUND	-4	VsaScan	At action: VSA_AP_SCANCONTENT, **at least** one match with the rule set was found.
VSA_E_MACRO_FOUND	-3	VsaScan	At action: VSA_AP_FINDALLMACROS, **at least** one macro was found.

VSA_E_VIRUS_FOUND	-2	VsaScan	At action: VSA_AP_SCAN VSA_AP_CHECKREPAIR During the scan of an object, **at least** one virus was found.
VSA_E_CLEAN_OK	-1	VsaScan	At action: VSA_AP_CLEAN VSA_AP_REMOVEALLMACROS VSA_AP_REPLACECONTENT Viruses/macros/matches were found. The repair/replacement of **all** virus infections or the removal of **all** macros (VSA_AP_REMOVEALLMACROS) was successful.
VSA_OK	0	VsaStartup VsaGetConfig VsaInit VsaScan VsaReleaseScan VsaEnd VsaCleanup	NO ERRORS or nothing found (with the specific scan parameters)
VSA_E_NO_SPACE	1	VsaStartup VsaGetConfig VsaInit VsaScan	Resource reservation in the operating system failed. For example, no memory, disk full, no file handles available, and so on.
VSA_E_NULL_PARAM	2	VsaGetConfig VsaInit VsaScan VsaReleaseScan VsaEnd	A NULL parameter was transferred, where, for example, a handle for a function, a value for a parameter, or the parameter itself was required.
VSA_E_INVALID_PARAM	3	VsaInit VsaScan	**At least** one parameter had an invalid value or type.
VSA_E_INVALID_HANDLE	4	VsaScan	The corresponding handle of the VSA is invalid.
VSA_E_NOT_INITIALISED	5	VsaStartup VsaGetConfig VsaInit VsaScan VsaReleaseScan VsaEnd VsaCleanup	VsaStartup() was not (correctly) called or was not successful. The adapter was not globally initialized. (Note: If a VSA is thread-safe and does not require a global initialization function, 0 should be returned)

VSA_E_EXPIRED	6	VsaInit VsaScan	The scan engine or associated drivers are too old. The definition of “too old“ is vendor-specific and is regarded as a warning by SAP.
VSA_E_LOAD_FAILED	7	VsaInit	Loading the engine itself or one of the underlying required components failed.
VSA_E_BAD_EXPRESSION	8	VsaInit	The transferred regular expressions in the corresponding parameters could not be interpreted or are invalid.
VSA_E_DRIVER_FAILED	9	VsaInit	Loading or initializing the drivers failed or **at least** one driver is invalid.
VSA_E_NOT_SUPPORTED	10	VsaInit VsaScan	Action or parameter is not supported by this VSA
VSA_E_INVALID_SCANOBJECT	11	VsaScan	Invalid object for this scan action, such as VSA_SP_FILE with the call “/opt/directory“
VSA_E_CIO_FAILED	12	VsaInit VsaScan	An unexpected return code was returned during a callback call, and the action was therefore terminated. (Or optionally, the timeout for an I/O operation was exceeded).
VSA_E_SCAN_FAILED	13	VsaScan	A problem occurred during the scan process, or the file could not be scanned at all. See VSA_SCANERROR for additional error analysis.
VSA_E_NOT_SCANNED	14	VsaScan	**At least** one object in the scan request was not scanned. **This is not a scan error**, but rather the object might not be scanned due to the file extension, but a VSA must display this.
VSA_E_END_FAILED	15	VsaEnd	Ending the VSA failed or the engine could not be terminated.
VSA_E_IN_PROGRESS	16	VsaEnd VsaCleanup	There was a problem with the call of VsaEnd/VsaCleanup. This type of situation can occur for threads that still exist and were not yet ended.
VSA_E_CBC_TERMINATED	17	VsaInit VsaScan	During a callback call, either the response **VSA_CB_TERMINATE** was received, or an unexpected return code was returned, and the action was therefore terminated.

596 **Note:**

597 When evaluating or interpreting return codes, you should note that a **VSA always returns a pes-**
598 **simistic** response (or must return this type of response). This means that when scanning an object
599 such as an archive, for example, in which there are multiple subobjects (files), a single virus or scan
600 error means that this return value is returned, regardless of whether the other subobjects were cor-
601 rectly processed. This could lead to a problem if you evaluate only return codes, that is, do not have
602 any callback or VSA_SCANINFO for evaluation. There are also situations, in which multiple errors
603 could occur. The rule of thumb that the lower the value of a return value, the more weight it has as a
604 return value applies here: for example, if multiple scan errors occur in an object, and unscanned ob-
605 jects, and a virus is found, then the return value is VSA_E_VIRUS_FOUND.

606 ***This does not apply for program-critical errors such as VSA_E_NO_SPACE – these are given***
607 ***priority and returned immediately!***

608 4.1.2 Return Values in Callback Functions

609 Depending on the message type, a caller can return the following return values during the processing
610 of a callback function. For more information, see also section 7, Message Types. This defines which
611 return values are permitted for which messages. A pessimistic interpretation of the return value also
612 applies here; that is, if a value that is not permitted or that is unknown is returned, it is interpreted as
613 VS_CB_TERMINATE and the action is therefore completely canceled, and the function is terminated
614 with **`VSA_E_CBC_TERMINATED`**.

615 The following return values are permitted in a callback function:

Return Value <=> VS_CALLRC	Value	Callback	Description
VS_CB_EOF	-1	< VSA_CIOCBFP >	End-of-File. The read/write action is to be terminated. All required data has been read or written.
VS_CB_OK	0	< VSA_CIOCBFP > <VSA_EVENTCBFP>	Continue the action, everything is OK.
VS_CB_NEXT	1	<VSA_EVENTCBFP>	The current action for this object is to be terminated or this object is to be skipped and the function should move to the next object.
VS_CB_TERMINATE	2	< VSA_CIOCBFP > <VSA_EVENTCBFP>	The current action is to be completely terminated. VsaScan or VsaInit ends with the return code VSA_E_CBC_TERMINATED

616 ## 4.2 VsaStartup

617 Global initialization of the adapter. This function is called once. The VSA should either protect itself
618 against multiple calls or implement a counter to ensure a “clean unloading” after VsaCleanup.

619 **`VSA_RC APIENTRY VsaStartup ()`**

620 ### Parameters:

621 IN: NONE

622 OUT: NONE

623 ### Return values:

624 - VSA_OK
625 - VSA_E_NO_SPACE
626 - VSA_E_NOT_INITIALISED

4.3 VsaGetConfig

This function returns a filled VSA_CONFIG structure. The VSA_CONFIG structure is intended as a reference for the user. It should contain all parameters supported by the external product and a list of the supported features and messages.

```
VSA_RC APIENTRY VsaGetConfig (VSA_CONFIG **)
```

Parameters:

IN: MANDATORY Handle for VSA_CONFIG*

OUT: Handle for filled VSA_CONFIG* structure

Return values:

- VSA_OK
- VSA_E_NO_SPACE
- VSA_E_NULL_PARAMETER
- VSA_E_NOT_INITIALISED

VSA_CONFIG

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
pAdapterInfo	PVSA_ADAPTERINFO	Filled structures VSA_ADAPTERINFO, VS_ADAPTER_T, and usVsiVersion, to identify the VSI compatibility.	Pointer to VSA_ADAPTERINFO structure.
uiVsaScanFlags	UInt	VSA_SP_FILE	OR-linked value from VSA_SCANPARAM_T
uiVsaActionFlags	UInt	VSA_AP_SCAN | VSA_AP_CHECKMIMETYPE | VSA_AP_BLOCKACTIVECONTENT	OR-linked value from VSA_ACTIONPARAM_T
uiVsaEvtMsgFlags	UInt	Either 0, if no callback is supported, or OR-linked value from VS_MESSAGE_T	OR-linked value from VS_MESSAGE_T
uiVsaCIOMsgFlags	UInt	Either 0, if no callback is supported, or OR linkage from VS_IOREQUEST_T	OR-linked value from VS_IOREQUEST_T
pInitParams	PVSA_INITPARAMS	Eithe NULL, if no initial parameters are set with default values, or an array of VSA_INITPARAM with vendor default settings, e.g. VS_IP_INITTEMP_PATH which points to OS specific TEMP variable	Pointer to VSA_INITPARAMS, which contains an array of all supported initial parameters. It can be predefined with default values.

pOptParams	PVSA_OPTPARAMS	VS_OP_SCANBESTEFFORT, VS_OP_SCANEXTRACT, VS_OP_SCANMIMTYPES, VS_OP_SCANEXTENSIONS, VS_OP_BLOCKMIMETYPES, VS_OP_BLOCKEXTENSIONS	Pointer to VSA_OPT-PARAMS, which contains an array of all supported optional parameters. It can be predefined with default values.

641 **VSA_ADAPTERINFO**

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
tAdapterID	VS_ADAPTER_T	Value assigned by SAP from VS_ADAPTER_T	Every vendor receives a separate value from VS_ADAPTER_T. If neded before a certification, a new ID can be requested for this by sending an e-mail to icc@sap.com
tThreadingModel	VS_THREADMODEL_T	Supported thread model, that is a value from VS_THREADMODEL_T	A VSA can specify whether it is thread-safe or not, based on the descriptions in COM. VS_THREAD_APARTMENT=0 VS_THREAD_BOTH =1 VS_THREAD_FREE =2
usVsiVersion	UShort	Supported NW-VSI version. The value 1 should be appear to be compatible, as long as customers have not implemented SAP note 1796762	Supported NW-VSI version. The value 1 will be accepted for NW-VSI 2.00 and is recommened as lony as customers are not implemented SAP note 1796762. If can also set here 2.
usVsaMajVersion	UShort	VSA major version	VSA major version
usVsaMinVersion	UShort	VSA minor version	VSA minor version
pszVendorInfo	PChar	Vendor string, support information or other partner related information	Version string of the vendor
pszAdapterName	PChar	Product name (incl. Own version String embedded)	VSA product name

4.4 VsaInit

This function initializes an instance in the VSA and returns information about the scan engine and possible drivers in VSA_INIT.

```
VSA_RC APIENTRY VsaInit(  VSA_CALLACK*,
                          VSA_INITPARAMS*,
                          VSA_INIT **
                       )
```

Parameters:

IN:	OPTIONAL	Handle for callback parameters: VSA_CALLBACK *
	OPTIONAL	Handle for array of initial parameters: VSA_INITPARAMS *
	MANDATORY	Handle for VSA_INIT*
OUT:		Handle for VSA_INIT*

Return values:

- VSA_OK
- VSA_E_EXPIRED
- VSA_E_NO_SPACE
- VSA_E_LOAD_FAILED
- VSA_E_BAD_EXPRESSION
- VSA_E_NULL_PARAM
- VSA_E_INVALID_PARAM
- VSA_E_DRIVER_FAILED
- VSA_E_NOT_SUPPORTED
- VSA_E_CIO_FAILED
- VSA_E_NOT_INITIALISED
- VSA_E_CBC_TERMINATED

667 **VSA_INIT**

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
hEngine	VSA_ENGINE	Engine handle	Internal engine handle
uiViruses	UInt		Number of known viruses
uiExtensions	UInt		Number of extensions to be scanned
uiIntRevNum	UInt		Internal revision number
uiSignatures	UInt		Internal signature of the engine
usDrivers	UShort	If >0, then pDriver<> NULL. See VSA_DRIVERINFO	Number of loaded drivers
pDriver	PVSA_DRIVERINFO		Array of driver information
usEngineMajVersion	UShort		Main engine version
usEngineMinVersion	UShort		Subengine version
pszEngineVersionText	PChar		Freely-definable engine version text
utcDate	time_t	If <>0, this value must contain a UTC date.	Date of engine (UTC)
iErrorRC	Int	If iErrorRC <>0, then pszErrorText must be filled.	VSA internal error number
pszErrorText	PChar		VSA or vendor error text for initialization errors

668 **VSA_DRIVERINFO**

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
pszName	PChar		Name of the driver
usDrvMajVersion	UShort		Main version of the driver
usDrvMinVersion	UShort		Subversion of the driver
utcDate	time_t	If <>0, this value must contain a UTC date	UTC date of the driver
uiViruses	UInt		Number of known viruses for this driver
uiVariants	UInt		Number of known variants of viruses for this driver
iDriverRC	Int		Internal error code of the driver

4.5 VsaScan

This is the actual scan function. If a handle is provided for VSA_SCANINFO*, more detailed information about the scan can be read here.

```
VSA_RC APIENTRY VsaScan(  VSA_INIT *,
                          VSA_CALLACK*,
                          VSA_SCANPARAM *,
                          VSA_OPTPARAMS *,
                          VSA_SCANINFO ** )
```

Parameters:

IN:	MANDATORY	Handle for VSA_INIT *
	OPTIONAL	Handle for VSA_CALLBACK *
	MANDATORY	Handle for VSA_SCANPARAM * parameters
	OPTIONAL	Handle for array of optional parameters: VSA_OPTPARAMS *
	OPTIONAL	Handle for VSA_SCANINFO*
OUT:	OPTIONAL	If a handle is provided, handle with filled VSA_SCANINFO *

Return values:

- VSA_E_BLOCKED_BY_POLICY
- VSA_E_CLEAN_FAILED
- VSA_E_PATTERN_FOUND
- VSA_E_CLEAN_OK
- VSA_E_MACRO_FOUND
- VSA_E_VIRUS_FOUND
- VSA_OK
- VSA_E_NO_SPACE
- VSA_E_NULL_PARAM
- VSA_E_INVALID_PARAM
- VSA_E_INVALID_HANDLE
- VSA_E_NOT_SUPPORTED
- VSA_E_INVALID_SCANOBJECT
- VSA_E_CIO_FAILED
- VSA_E_SCAN_FAILED
- VSA_E_NOT_SCANNED
- VSA_E_NOT_INITIALISED
- VSA_E_EXPIRED
- VSA_E_CBC_TERMINATED

704 **VSA_CALLBACK**

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
pEventCBFP	VSA_EVENTCBFP		Function pointer of type VSA_EVENTCBFP
uiEventMsgFlags	UInt		OR-linked value from VS_MESSAGE_T
pvUsrData	VSA_USRDATA		Pointer to own data structure (void*)
pClientIOCBFP	PVoid		Function pointer of type VSA_CIOCBFP
uiCIOMsgFlags	UInt		OR-linked value from VS_IOREQUEST_T
pvIOData	VSA_IODATA		Pointer to own data structure

705 ## 4.6 <VSA_EVENTCBFP>

706 The callback function < VSA_EVENTCBFP> is used to transfer event messages during a scan to the ap-
707 plication.

```
708 VS_CALLRC CALLBACK <VSA_EVENTCBFP> (   VSA_ENGINE*,
709                                         VS_MESSAGE_T,
710                                         VSA_PARAM*,
711                                         VSA_USRDATA*
712                                     )
```

713 ### Parameters:

714 All parameters are provided and released again by the callback handler.

715 IN: Handle of VSA VSA_ENGINE *

716 Identifier of the type VS_MESSAGE_T

717 VSA_PARAM* (void*) Pointer that contains the user information.

718 VSA_USRDATA * (void*) Pointer in which user's own handle is
719 transported.

720 OUT: NONE

721 ### Return values:

722 - VS_CB_OK
723 - VS_CB_NEXT
724 - VS_CB_TERMINATE

4.7 <VSA_CIOCBFP>

The callback function < VSA_CIOCBFP> is used to delegate read or write operations to the calling application.

```
VS_CALLRC CALLBACK <VSA_CIOCBFP> (VSA_ENGINE*,
                                  VS_IOREQUEST_T,
                                  void *,
                                  size_t,
                                  size_t *
                                  )
```

Parameters:

All parameters are provided and released again by the callback handler.

IN:	Handle of VSA VSA_ENGINE *
	Identifier of the type VS_IOREQUEST_T
	void* Pointer to reserved memory
	Size of the reserved memory area
OUT:	Size of the used memory area

Return values:

- VS_CB_EOF
- VS_CB_OK
- VS_CB_NEXT
- VS_CB_TERMINATE

4.8 VsaReleaseScan

```
VSA_RC APIENTRY VsaReleaseScan(VSA_SCANINFO **)
```

Parameters:

IN:	MANDATORY	Handle for VSA_SCANINFO *, this can, however, be NULL
OUT:		Handle for VSA_SCANINFO *, released and now NULL.

Return values:

- VSA_OK
- VSA_E_NULL_PARAM
- VSA_E_NOT_INITIALISED

4.9 VsaEnd

```
VSA_RC APIENTRY VsaEnd ( VSA_INIT   **,
                         VSA_CONFIG **,
                       )
```

Parameters:

IN:	MANDATORY	Handle for VSA_INIT *, which can also be NULL.
	MANDATORY	Handle for VSA_CONFIG *, which can also be NULL.
OUT:		Handle for VSA_CONFIG*, released and now NULL.
		Handle for VSA_INIT*, released and now NULL.

Return values:

- VSA_OK
- VSA_E_NULL_PARAM
- VSA_E_NOT_INITIALISED
- VSA_E_IN_PROGRESS
- VSA_E_END_FAILED

4.10 VsaCleanup

Global completion or termination of a VSA. See VsaStartup. This function is called last. All other “clean-up actions” should be performed here.

```
VSA_RC APIENTRY VsaCleanup ()
```

Parameters:

IN: NONE

OUT: NONE

Return values:

- VSA_OK
- VSA_E_IN_PROGRESS
- VSA_E_NOT_INITIALISED

4.11 Default Values in Function Prototypes

Various parameters for VSA functions are optional when the function is called, that is, they can be transferred with “NULL”. For example, in the case of VsaInit initialization parameters do not have to be transferred. If no values are transferred, the VSA is to use its default values. If, on the other hand, initialization parameters are required, the response VSA_E_LOAD_FAILED must be returned here, and the VSA_INIT (pszErrorText) error text should detail that a required value was not set.

The following constants have been defined for transferring NULL parameters:

- VSA_NO_INITPARAMS
- VSA_NO_OPTPARAMS
- VSA_NO_SCANINFO
- VSA_NO_CALLBACK

Example:

```
VsaInit(VSA_NO_CALLBACK, VSA_NO_INITPARAMS, &p_init)
VsaScan(p_init,
        VSA_NO_CALLBACK,
        p_scanparams,
        VSA_NO_OPTPARAMS,
        VSA_NO_SCANINFO
        )
```

5 Parameters in the VSA

The transfer of parameters to structure arrays is intended to provide a more flexible way of transferring data between VSILIB and VSA. However it is necessary to define certain parameter codes. Depending on the VSA, it may be necessary, for example, that initial parameters are not used at all. This must be specified by transferring VSA_NO_INITPARAMS (=: NULL) in VSA_CONFIG. All other supported parameters must be added in the associated array when "**`VsaGetConfig`**" is called.

Note on transferring lists:

The transfer of multiple values with the parameter type VS_TYPE_CHAR (unsigned char) is implemented using a list, that is, a delimiter is defined, which is used to separate multiple values. The semi-colon (;) is currently defined as the delimiter (see vsaxxtyp.h): the VSA must check itself how many individual values are transferred in the list or whether only a single value is transferred. The lLength specification for this parameter therefore refers to the entire length of the character array and not to the number of elements in the list.

The VS_TYPE_CHAR specification refers by definition (unsigned char) to the UTF-8 character set. The SAP scan API performs any necessary conversion to other character sets (UNICODE).

5.1 Initial Parameters

Parameter Code	Parameter Type	Description
VS_IP_INITDRIVERS	VS_TYPE_CHAR	A list of "drivers" that are to be used for initialization. Whether only the name or the path and name of the driver file is specified depends on the product.
VS_IP_INITEXTRADRIVERS	VS_TYPE_CHAR	An extra driver is also to be loaded. Is required if, for example, this is in a different directory.
VS_IP_INITDRIVERDIRECTORY	VS_TYPE_CHAR	Default directory in which the required drivers can be found.
VS_IP_INITEXTRADRIVERDIRECTORY	VS_TYPE_CHAR	"Extra" or additional directory in which drivers can be found.
VS_IP_INITSERVERS	VS_TYPE_CHAR	A list of servers:ports that are required to initialize VSAs that are active as scan daemons. The syntax here is: server1.domain.com:1234; server2.domain.com:4321;...
VS_IP_INITTIMEOUT	VS_TYPE_TIME_T	Timeout in seconds for creating the connection to the VSA.
VS_IP_INITRECONNECTTIME	VS_TYPE_TIME_T	Time in seconds after which a VSA should reattempt connection if a TIMEOUT occurs.
VS_IP_INITCONTENTPATTERN	VS_TYPE_CHAR	Regular expression in accordance with POSIX 1003.2 for which a search is to be performed.

VS_IP_INITREPLACEPATTERN	VS_TYPE_CHAR	Expression that is to be used to replace the previous expression if found.
VS_IP_INITTEMP_PATH	VS_TYPE_CHAR	Directory for temporary files which might be created during scan.
VS_IP_INITDIRECTORY	VS_TYPE_CHAR	Default root directory for initialization. Can be used, if own directory structure is used and only the root is needed
VS_IP_INITENGINES	VS_TYPE_CHAR	Threat engine(s) to be used for malware protection, content filtering, etc.
VS_IP_INITENGINEDIRECTORY	VS_TYPE_CHAR	Default directory in which the threat engine(s) can be found
VS_IP_INITUPDATE_URI	VS_TYPE_CHAR	URI (URL or local path) to an update service or updated drivers
VS_IP_INITLICENSE_PATH	VS_TYPE_CHAR	Path where the external product can find a valid license file

5.2 Scan Parameter

The scan parameter specifies the actual action and defines the object type to be processed.

VSA_SCANPARAM

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
tScanCode	VSA_SCANPARAM_T	VSA_SP_FILE, that is, a VSA must be able to scan at least one file.	A value from the enumeration VSA_SCANPARAM_T
tActionCode	VSA_ACTIONPARAM_T	VSA_AP_SCAN, that is a VSA must support at least one virus scan.	A value from the enumeration VSA_ACTIONPARAM_T
pszObjectName	PChar		Transfer of the path or file name in the case of VSA_SP_FILE / VSA_SP_DIRECTORY Transfer encoding type as MIME type in case of VSA_SP_HTTP_MESSAGE / VSA_SP_MAIL_MESSAGE
pbByte	PByte		Transfer of a pointer to bytes in the case of VSA_SP_BYTES
lLenth	size_t		Transfer of the length of either pszObjectName or length of the bytes that are to be checked (pbByte).
uiJobID	UInt	Transfer own ID to be able to assign request again	This ID is used for internal SAP purposes and must simply be transported by the VSA.

5.3 Optional Parameters

Parameter Code	Parameter Type	Description
VS_OP_SCANBESTEFFORT	VS_TYPE_BOOL	The scan should be performed on the "best effort" basis, that is, all (security critical) flags that allow a VSA to scan an object should be activated: such as, SCANALLFILES and SCANEXTRACT, but also internal flags. Details about exactly which flags these are can be stored in the certification.
VS_OP_SCANALLFILES	VS_TYPE_BOOL	Scans for all files regardless of their file extension.
VS_OP_SCANALLMACROS	VS_TYPE_BOOL	Scans for all macros regardless of the file type of the object.
VS_OP_SCANALLEMBEDDED	VS_TYPE_BOOL	Scans for all embedded objects, for example in HEX/BIN/UU/-MIME encoded objects, and also for embedded scripts
VS_OP_SCANEXTENSIONS	VS_TYPE_CHAR	List of the file extensions for which the VSA should scan. Wilcards can also be used here in order to search for patterns, that is, * stands for this location and following, and ? for only this character. The syntax is as follows: exe;com;do?;ht* => `*` therefore means VS_OP_SCANALLFILES
VS_OP_SCANHEURISTICLEVEL	VS_TYPE_INT	Activates heuristic search at level X, 0 means deactivated.
VS_OP_SCANONLYHEURISTIC	VS_TYPE_BOOL	Scans using only heuristic mechanisms
VS_OP_SCANLIMIT	VS_TYPE_INT	Restricts scan/repair of an object to a number. In this way, you can search, for example, for only the first virus or the first match in the case of a content scan.
VS_OP_SCANEXTRACT	VS_TYPE_BOOL	Archives or compressed objects are to be unpacked
VS_OP_SCANEXTRACT_PATH	VS_TYPE_CHAR	Unpack directory
VS_OP_SCANEXTRACT_SIZE	VS_TYPE_SIZE_T	Maximum unpack size
VS_OP_SCANEXTRACT_TIME	VS_TYPE_TIME_T	Maximum unpack time
VS_OP_SCANEXTRACT_DEPTH	VS_TYPE_INT	Maximum depth to which an object is to be unpacked.
VS_OP_SCANEXTRACT_RATIO	VS_TYPE_INT	Maximum ratio of compressed to uncompressed in the case of a packed object.
VS_OP_SCANLOGPATH	VS_TYPE_CHAR	Path for log or trace file of the VSA
VS_OP_SCANDIREXCLUDELIST	VS_TYPE_CHAR	List of the directories to be excluded when scanning subdirectories.

VS_OP_SCANSUBDIRLEVEL	VS_TYPE_INT	Scans subdirectories to a level of X. 0 means that subdirectories are not scanned.
VS_OP_SCANACCESSFILELOCAL	VS_TYPE_BOOL	Object can be accessed locally by the VSA or engine. Informs scan daemons that the object does not need to be sent using sockets.
VS_OP_SCANMIMETYPES	VS_TYPE_CHAR	List of the MIME types to be scanned for.
VS_OP_CLEANRENAME	VS_TYPE_BOOL	An infected object is to be renamed if it is repaired. The rules for this depend on the VSA.
VS_OP_CLEANDELETE	VS_TYPE_BOOL	An infected object is to be deleted during the repair.
VS_OP_CLEANQUARANTINE	VS_TYPE_CHAR	Directory for infected objects. VSA should move the infected objects there.
VS_OP_CLEANNODELETEINARCHIVE	VS_TYPE_BOOL	If an infection is found in an archive file, this subobject should not be deleted.
VS_OP_CLEANNODELETEINEMBEDDED	VS_TYPE_BOOL	If an infection is found in an embedded object (including macros), this subobject should not be deleted. For example, for Microsoft Word documents with embedded macro viruses.
VS_OP_CLEANNODELETEJOKES	VS_TYPE_BOOL	Infected objects should not be deleted if the VSA determines that they are not “real“ viruses, but rather hoaxes or joke viruses.
VS_OP_BLOCKMIMETYPES	VS_TYPE_CHAR	List of MIME types to be used as black list
VS_OP_BLOCKEXTENSIONS	VS_TYPE_CHAR	List of file extensions to be used as black list

Scan actions that must be terminated due to optional parameters, such as exceeding the maximum unpack size of compressed files must return the message VS_M_NOTSCANNED if callback is activated, and the return code VSA_E_NOT_SCANNED. The latter also applies for adapters that do not support callback.

When setting VSA_NO_OPTPARAMS, the VSA should use all internal default values. The certification requirement for the optional parameters are VS_OP_SCANBESTEFFORT, VS_OP_SCANEXTRACT, VS_OP_SCANMIMETYPES, VS_OP_SCANEXTENSIONS, VS_OP_BLOCKMIMETYPES, VS_OP_BLOCKEXTENSIONS. All other parameters or features for scans are checked during the certification and included in the description of the adapter.

6 Evaluating Scan Results

828 SAP applications analyses the scan results in following order:

829 1. Return code of VsaScan (see 4.1.1 Return Values in VSA Functions)

830 2. Evaluation of structure VSA_SCANINFO

831 The usage of Callback messages (see section 7 Message Types in VSA Callback Functions)
832 is still an option, however in NW-VSI 1.00 no partner has used this variant and no SAP application is de-
833 pendent on this, therefore this usage of not longer recommended.

834 Depending on the mode type (see section 3.3), you can also use several results, where the same
835 structure information is transferred using the callback as is transferred in VSA_SCANINFO after the
836 completion of the scan. However, the route using callback transfers this information directly, and it is also
837 not necessary to release any additional memory by calling VsaReleaseScan. These structures
838 (VSA_VIRUSINFO, VSA_SCANERROR, and VSA_CONTENTINFO) are appended in VSA_SCANINFO
839 as an array of structures.

840 6.1 Structure Description

841 VSA_SCANINFO and its substructures are explained in the following:

842 **VSA_SCANINFO**

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
uiJobID	UInt	Transfer own ID, to be able to assign request again.	This ID is for internal SAP purposes and must simply be transported by the VSA.
uiScanned	UInt	Number of scanned objects	Number of scanned objects and also the counter for the number of structures of VSA_CONTENTINFO
uiNotScanned	UInt	Number of objects not scanned	Number of objects not scanned
uiClean	UInt	Number of virus-free objects	Number of virus-free objects
uiInfections	UInt	Number of infected objects	This field is also the counter for the number of structures of VSA_VIRUSINFO
uiScanErrors	UInt	Number of scan errors	This field is also the counter for the number of structures of VSA_SCANERROR
pContentInfo	PVSA_CONTENTINFO	Pointer to an array of VSA_CONTENTINFO	Pointer to an array of VSA_CONTENTINFO
pVirusInfo	PVSA_VIRUSINFO	Pointer to an array of VSA_VIRUSINFO	Pointer to an array of VSA_VIRUSINFO
pScanError	PVSA_SCANERROR	Pointer to an array of VSA_SCANERROR	Pointer to an array of VSA_SCANERROR
pbBytesCleaned	PByte	If VSA has changed/cleaned the object, return a pointer to an array of bytes	Pointer to an array of bytes.
lBytesCleaned	size_t	If VSA has set pbBytesCleaned, set here length array	Length of pbBytesCleaned

Note:

The individual parameters in the structure VSA_SCANINFO provide more detailed information about the current action, that is an adapter can provide more information than is requested here. An example of this is a virus scan that cannot be performed due to a parameter setting: the parameter VS_OP_EXTRACT is set to FALSE (0) and therefore only the archive itself is to be scanned and not its contents. The return code in this case must be VSA_E_NOT_SCANNED, since not all objects were scanned. The parameters in VSA_SCANINFO in this case would have to be: (uiScanned=1, uiNotScanned=X, uiClean=1, uiScanErrors=Y). The VSA can also provide information Y times about VSA_SCANERROR about why the object was not scanned, or set X to the value that contains the number of unscanned objects. This can mean that the VSA could not scan at least one object, or, if the archive itself can be read, also the number of objects in the archive that were not scanned. The information from VSA_SCANERROR is written to a log using the SAP-internal scan API.

The structure VSA_CONTENTINFO has no meaning in the case of a virus scan. It can be filled, but is only compulsorily queried in the case of a content scan.

VSA_CONTENTINFO

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
tObjectType	VS_OBJECTTYPE_T	Value from VS_OBJECTTYPE_T, where at least the major ID is expected.	VS_OBJECTTYPE_T is divided into main areas, which map the MIME types to RFC 2045 to 2049. Below this, there are SAP IDs for a more detailed specification of an object type.
pszExtension	PChar	File extension of the object type	File extension of the object type
pszContentType	PChar	MIME type in canonical format to RFC 2045-2049	MIME type in canonical format to RFC 2045-2049
pszCharSet	PChar		Character or Encoding used in content type
uiJobID	UInt	Transfer own ID, to be able to assign request again.	This ID is used for internal SAP purposes and must simply be transported by the VSA
pszObjectName	PChar	Name of the scanned object	Can be the path name and file name of the file.
lObjectSize	size_t	Size of this object	Size of this file, for example

858 VSA_SCANERROR is queried both in the case of a virus scan and in the case of a content scan, if an er-
859 ror occurs.

860 **VSA_SCANERROR**

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
uiJobID	UInt	Transfer own ID, to be able to assign request again	This ID is for internal SAP purposes and must simply be transferred by the VSA.
pszObjectName	PChar	Name of the scanned object	Can be the path name and file name of the file.
uObjectSize	size_t	Size of this object.	Size of this file, for example.
iErrorRC	Int	VSA internal error code	Is vendor-dependent and can be freely defined.
pszErrorText	PChar	VSA internal error text	Is vendor-dependent and can be freely defined.

861 The structure for information about an infection is queried in the case of a virus scan if there is an infection.

862 **VSA_VIRUSINFO**

Structure Field	Type	Certification Requirement	Description
struct_size	size_t	Size of the structure	Size of the structure
bRepairable	BOOL	If VSA_AP_CHECKREPAIR is specified, this value can be queried to determine whether the infected object can be repaired.	The value is only useful, if a setting was specified for the VSA that it should either check whether repair is possible or perform it immediately.
tDetectType	VS_DETECTYPE_T		Value from enumeration
tVirusType	VS_VIRUSTYPE_T		Value from enumeration
tObjectType	VS_OBJECTTYPE_T		Value from enumeration
tActionType	VS_ACTIONTYPE_T		Value from enumeration
uiVirusID	UInt		Internal virus ID can be specified here.
pszVirusName	PChar	Name of the virus, can also be vendor-specific	Name of the virus, can be vendor-specific
uiJobID	UInt	Transfer own ID, to be able to assign request again	This ID is used for internal SAP purposes and must simply be transported by the VSA
pszObjectName	PChar	Name of the scanned object	Can be the path name and file name of the file.
uObjectSize	size_t	Size of this object	Size of this file, for example
pszFreeTextInfo	PChar		Freely definable text for vendors

6.2 List of Enumerations in VSA_VIRUSINFO

Various information values about a virus infection can be queried using the structure VSA_VIRUSINFO, which is either sent to the caller using a callback or is stored in VSA_SCANINFO. The following enumerations should be used for information purposes for applications (VSILIB) during a virus scan, that is, the quality of the information from enumerations is not the basis of a certification and therefore an application cannot rely on the quality of this type of information.

However, plausibility checks are made for a certification, that is, in the case of an identified virus, there must be at least a value greater than 0 in VS_DETECTTYPE_T. Additional subclassification is not tested here.

VS_DETECTTYPE_T

Code	Value	Description
VS_DT_NOVIRUS	0	No virus
VS_DT_KNOWNVIRUS	1	Known virus
VS_DT_VARIANTVIRUS	2	New but similar to a known virus
VS_DT_NEWVIRUS	3	Unknown virus
VS_DT_ACTIVECONTENT	4	Not a virus but found script in content
VS_DT_MIMEVALIDATION	5	Not a virus but found invalid MIME content
VS_DT_PATTERNMATCH	6	Not a virus but pattern matched
VS_DT_ERROR	7	An error occurred

A VSA can use VS_VIRUSTYPE_T to further categorize the found virus.

VS_VIRUSTYPE_T

Code	Value	Description
VS_VT_NOVIRUS	0	Not a virus
VS_VT_VIRUS	1	Known virus, “normal” for us
VS_VT_TROJAN	2	Trojan
VS_VT_JOKE	3	A bad joke, not a virus
VS_VT_HOAX	4	Wannabe virus; harmless
VS_VT_POLYMORPH	5	A polymorphic virus
VS_VT_ENCRYPTED	6	An encrypted object that looks like a virus
VS_VT_COMPRESSED	7	A compressed object that looks like a virus
VS_VT_APPLICATION	8	Application that behaves like a virus
VS_VT_WORM	9	A worm virus (such as an e-mail worm)
VS_VT_CORRUPTED	10	A corrupted object or an object that cannot be analyzed
VS_VT_TEST	11	A test virus, such as EICAR
VS_VT_BACKDOOR	12	A backdoor (such as a dialer)
VS_VT_EXPLOIT	13	Crash
VS_VT_FLOODER	14	Attempted denial-of-service
VS_VT_SPAM	15	SPAM (Mail)
VS_VT_PUA	16	Potential unwanted application
VS_VT_CHAMELEON	17	Chameleon file found

After an infection was repaired, the enumeration VS_ACTIONTYPE_T should specify what was done with the object.

VS_ACTIONTYPE_T

Code	Value	Description
VS_AT_NOACTION	0	No action
VS_AT_ACTIONFAILED	1	The action failed, error
VS_AT_CLEANED	2	Object was repaired
VS_AT_RENAMED	3	Object was renamed
VS_AT_DELETED	4	Object was deleted
VS_AT_MOVED	5	Object was moved, to quarantine directory
VS_AT_BLOCKED	6	Object was blocked
VS_AT_ENCRYPTED	7	Object was encrypted

In principle, these specifications are not mandatory for a certification of the virus scan parts. However, possible errors or non-implementation of some specifications will be checked and noted in the certificate.

6.3 List of Enumerations in VSA_CONTENTINFO

VS_OBJECTTYPE_T is used with VSA_SP_SCANCONTENT or VSA_AP_CHECKSCAN to identify an object. This specification is used in both VSA_VIRUSINFO and VSA_CONTENTINFO. In principle, however, this information is only evaluated by the SAP-internal scan API during a content scan. However, the information is also transferred to the engine during a virus scan, meaning that in this case the VSA only needs to forward this information.

During a content scan, the value from VS_OBJECTTYPE_T specifies the type. The following classification exists in this case:

The types are divided into certain areas, which are based on the definition of the MIME types, with the difference that there are main types and subtypes. The main type specifies only, for example, that it is an archive. The main types are listed here. The subtypes are maintained in the header **VSAXXTYP.H**, since there can often be extensions here.

VS_OBJECTTYPE_T

Code (Major ID)	Value	Description
VS_OT_TEXT	0	Text objects
VS_OT_IMAGE	200	Image object
VS_OT_VIDEO	300	Video objects
VS_OT_AUDIO	400	Audio objects
VS_OT_BINARY	500	Binary objects
VS_OT_ARCHIVE	600	Archive objects
VS_OT_MULTIPART	700	Multipart objects
VS_OT_MESSAGE	800	Message objects
VS_OT_MODEL	900	Model objects

7 Message Types in VSA Callback Functions

By passing a function pointer to VSA_CALLBACK and setting the event message values, various events can be received in a separate callback function. This means that information can be received and that interactive intervention can be made in the current action. This is possible using various return values in the callback function (see 4.1.2). If a return code is returned that is unknown or not permitted for the message, this is interpreted as VS_CB_TERMINATE, and the running action is terminated.

7.1 Client Input/Output (I/O)

All request codes must be implemented here or the respective requests must be reacted to. The buffer is provided by the adapter. Depending on the options of the virus scan engine, the VSA must either buffer the entire stream or, with particular types, can decide during the processing whether additional scanning is worthwhile or an infection has already been found.

Request Code	Value	Return	Description
VS_IO_OPENREAD	0x00000010	VS_CB_OK VS_CB_TERMINATE	Specifies that a read operation is started.
VS_IO_OPENWRITE	0x00000020	VS_CB_OK VS_CB_TERMINATE	Specifies that a write operation is started.
VS_IO_CLOSEREAD	0x00000040	VS_CB_OK VS_CB_TERMINATE	Specifies that a read operation is to be terminated.
VS_IO_CLOSEWRITE	0x00000080	VS_CB_OK VS_CB_TERMINATE	Specifies that a write operation is to be terminated.
VS_IO_READ	0x00000100	VS_CB_EOF VS_CB_OK VS_CB_TERMINATE	A read operation is requested. The VSA provides a pointer and specifies the size of the buffer available. Finally the VSA can determine how many bytes were read in the last parameter.
VS_IO_WRITE	0x00000200	VS_CB_EOF VS_CB_OK VS_CB_TERMINATE	A write operation is started. The VSA provides a pointer and specifies the size of the area of the bytes to be written.

7.2 Events

The message code VS_M_ALL is a constant default value and represents all supported messages of a VSA. This value is not a callback message, but rather allows the caller to activate messages irrespective of which a VSA ultimately supports. The decision about which messages are included in VS_M_ALL is to be made in the adapter itself.

909 The following messages can be received using an event callback function.

Message Code	Value	Info Value	Return	Description
VS_M_ALL	0xFFFFFFFF	NONE, is not used in the callback itself and is only a representative of messages supported by the VSA.	NONE, is not sent itself.	All messages that are possible or supported in the VSA should be activated here. This code can be set in the structure VSA_CALLBACK.
VS_M_ERROR	0x01000000	Error string	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	Vendor-specific error text of a VSA for errors such as incorrect/missing parameters, failed memory reservations, and so on, including scan errors. Whether the error should be ignored and processing continued, the VSA should skip to the next object, or the action should be terminated.
VS_M_ABORTSCAN	0x02000000	uiJobID	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	Query: whether the running scan/repair is to be terminated or should continue to the next object, or in the case of VS_CB_TERMINATE, whether the action should be completely terminated.
VS_M_VIRUS	0x00010000	VSA_VIRUSINFO	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	At least one virus was found. More information in the transferred VSA_VIRUSINFO. Should the VSA attempt to continue with the action, skip to the next object, or terminate the action completely?
VS_M_CLEAN	0x00020000	uiJobID	VS_CB_OK VS_CB_TERMINATE	Object with request JobID is clean. Scan continues to the next object, therefore VSA_CB_NEXT is not required.

VS_M_NOTSCANNED	0x00040000	VSA_SCANERROR	VS_CB_OK VS_CB_TERMINATE	Object was not scanned. The reason for the error is reported in VSA_SCANNERROR. This message is also sent again in the case of user termination VS_M_ABORTSCAN of a scan by the previously sent return value VS_CB_NEXT.
VS_M_REPAIRED	0x00080000	VSA_VIRUSINFO	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	At least one infection in the object was repaired. Exactly what action was taken with the object is reported in VSA_VIRUSINFO->tActionType
VS_M_NOTREPAIRED	0x00100000	VSA_SCANERROR	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	At least one infection in the object could not be completely repaired, see VSA_SCANERROR. Should the VSA attempt to continue with the action, skip to the next object, or terminate the action completely?
VS_M_OBJECTFOUND	0x00200000	pszObjectName	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	An object was found. This is used when scanning directories, and indicates, for example, an “OnFileFound” event. → This is called before the scan.
VS_M_MACROSCLEANED	0x00400000	uiJobID	VS_CB_OK VS_CB_TERMINATE	All macros in objects were removed. The scan continues to the next object, therefore VSA_CB_NEXT is not required.
VS_M_CONTAINMACROS	0x00800000	uiJobID	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	The object contains macros. Is sent when the first macro is found.

VS_M_SCANACTIONS	0x00000001	Number of scan runs for current object	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	If the number of scans per object is unlimited, the numerical value of the number of scan runs for the current object is transferred here. This is only transferred for objects that contain multiple infections and therefore require multiple scan runs.
VS_M_SCANPROGRESS	0x00000002	Progress percentage	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	Is continuously sent and contains the progress percentage for the current scan action.
VS_M_MATCHPATTERN	0x00000004	VSA_CONTENTINFO	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	Transfers information about the object in accordance with the rules.
VS_M_REPLACEDPATTERN	0x00000008	VSA_CONTENTINFO	VS_CB_OK VS_CB_NEXT VS_CB_TERMINATE	The content was replaced.
VS_M_EXPIRED	0x00000010	NULL	VS_CB_OK VS_CB_TERMINATE	The reload is needed

910 **Note:**

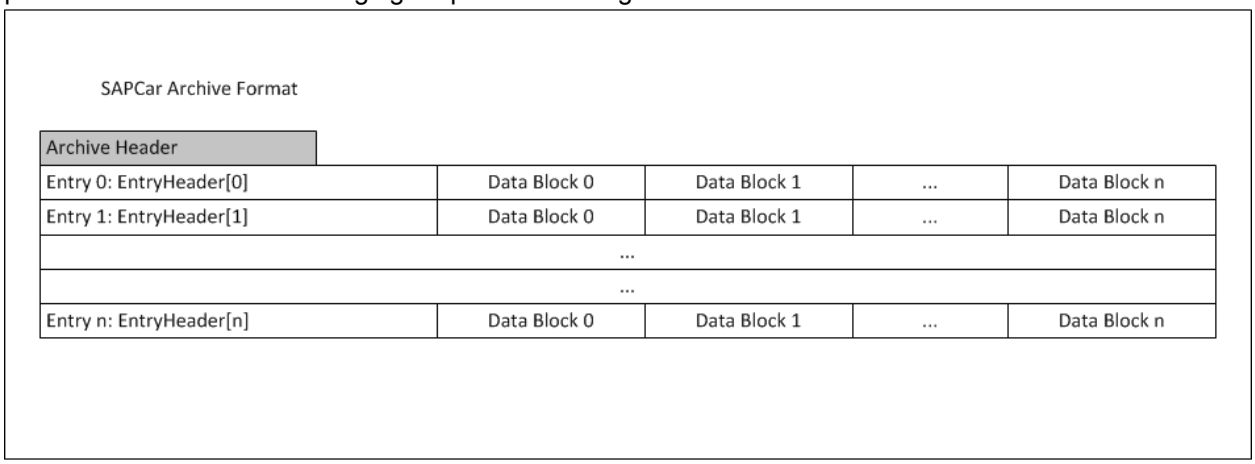
911 The specification “at least one” in this overview means that at least one event occurred during the scan.
912 This applies for objects that have multiple infections and must be scanned more than once. The event is
913 first sent at the first occurrence.

8 Scanning SAP Archive Formats

SAP uses its own proprietary archive format: SAR, for delivering its software components. However, it should also be possible to scan these files for viruses using external adapters (VSA) with a virus scan interface (NW-VSI) delivered by SAP.

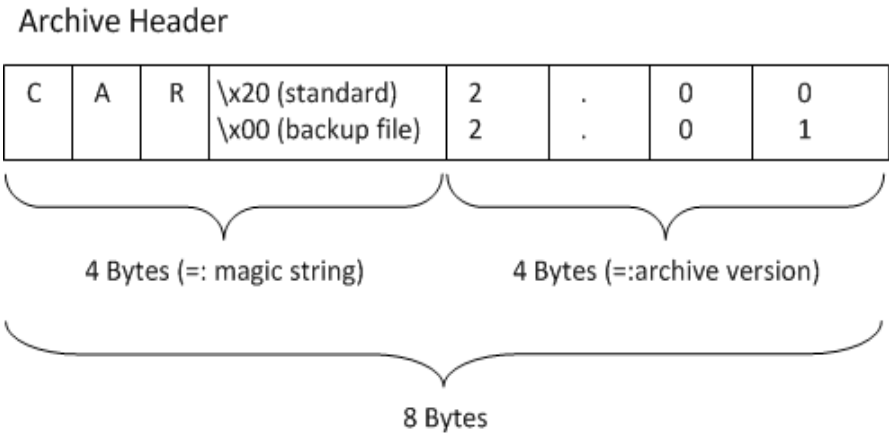
The SAP archive format was released with MaxDB to the OpenSource community and is available here: ftp://ftp.sap.com/pub/maxdb/current/7.6.00/maxdb-source-7_6_00_37.zip .

The stand-alone program SAPCAR is delivered to customers through the SAP kernel and frontend CDs or can be downloaded from the SAP Service Marketplace (http://service.sap.com). The VSA-SDK contains a parser of the archive format. A certified VSA or the virus scan engine itself should use SAPCAR by starting a child process to be able to unpack SAR files or repack them. In principle, objects (in this case, files) are analyzed or identified using the file extension in many virus scan engines anyway, and, if necessary, unpacked. SAPCAR can be included in these routines to ensure the scan inside the archive. Even better is the integration of the archive format into existing unpacker – because all AV scanner need such unpackker libraries. The following figure provides a rough overview about the archive structure.

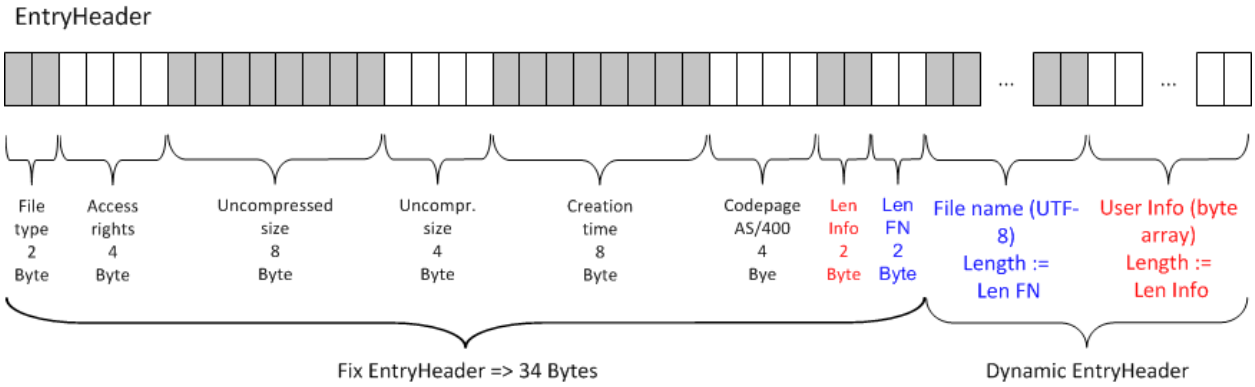


A certified VSA MUST be able to unpack and scan SAR files either with help of SAPCAR or the sources in VSA-SDK. If there are errors, then the return code "VSA_E_NOT_SCANNED" must be returned.

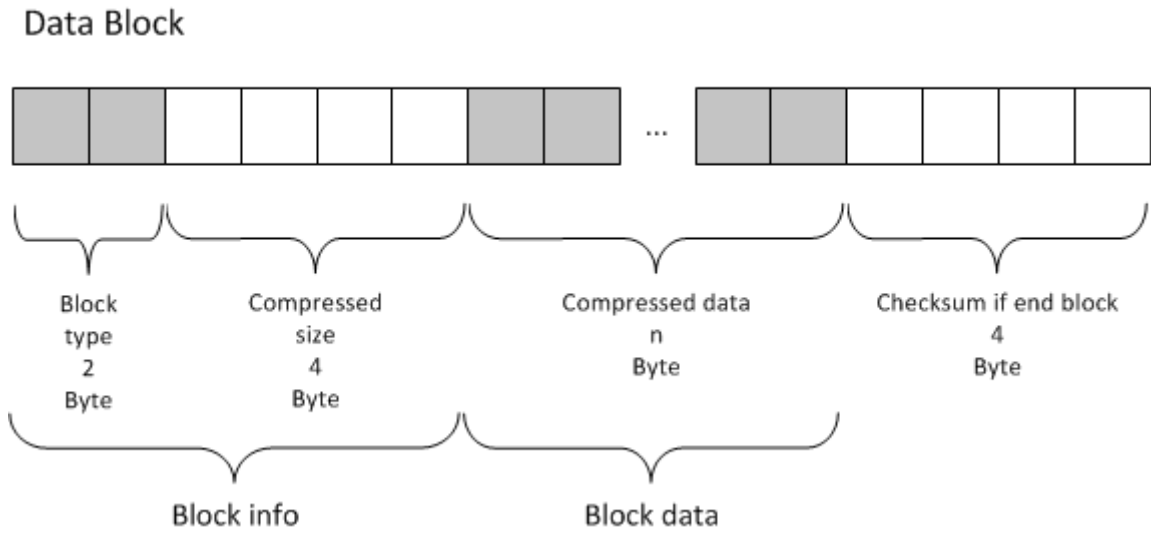
The archive header must be known in order to be able to analyze SAR files. The first four bytes of this format are decisive. These first four bytes contain either "CAR\0" ('CAR0x00') or "CAR " (that is 'CAR0x20'). This means that an archive of this type can be unpacked using SAPCAR. The file extension can be ".car" or ".sar".



934 The content of a SAR is accessibly throuth the entries, which are shown in following figure:



935 The file data is stored in the data blocks



936 The data block is compressed. The algorithms for this compression is available in MaxDB sources. How-
937 ever if you need help in integration, contact SAP.

938 In a NetWeaver environment there is always the command line program SAPCAR avaiable. This pro-
939 gramm can be found in one of the directories defined in the operation system search pathes, e.g. PATH
940 on Windows, LD_LIBRARY_PATH on Linux, etc.

941 Therefore you can rely on the execution of process sapcar.exe / SAPCAR without searching or configura-
942 tion of a path where to search for it.

The use of SAPCAR and its options can be determined by the call of the program:

Unpacking archives:

```
SAPCAR    -x[v] [f archive] [-R directory] [-A filename]
                        [-V] [file1 file2....]
```

Verifying an archive:

```
SAPCAR    -d[v] [f archive] [-V] [file1 file2....]
```

Checking files to be processed:

```
SAPCAR    -l [-A filename] [-X filename] [file1 file2...]
```

Options:

-a	: append files to an archive
-A FILE	: get names to include from file FILE
-C DIR	: change to directory DIR
-e	: redirect output from stdout to file sapcar_output
-f FILE	: use archive file FILE (default DEFAULT.SAR)
-flat	: don't preserve file path when extracting files
-g	: ignore case of archive names while extracting, testing, or listing archives
-h	: do not change permissions of existing directories during extraction
-i	: ignore inaccessible files while creating an archive
-l	: check availability of files to be processed
-lower	: convert filenames to lowercase while extracting
-m	: merge two archives
-n	: print statistical information
-p octalvalue	: set permissions of all files in archive to value
-P	: use absolute path-names (use carefully)
-r	: do not resolve symbolic links/shortcuts while creating an archive
-R dir	: use dir instead of current directory
-s	: do free space check
-T FILE	: rename files to be included in FILE
-v	: verbosely list files processed
-V	: compute or verify checksum (obsolete, always set, for backward compatibility)
-X FILE	: get names to exclude from FILE

The example below shows a call that uses the ANSI C function “execv“ to unpack the sap.sar file:

```
-------------------------------------------------------------------------
      Char *args[10],
            execpath[32]=„c:\\VSAEXECPATH\\SAPCAR.exe”;
      args[0] = execpath;
      args[1] = „-xvf sap.sar”;
      args[2] = “-R c:\\temppath”;
      args[3] = NULL;
      rc = execv( execpath,
                  args
                );
      if ( rc != 0 ) {
            ... /* error handling */
      }
-------------------------------------------------------------------------
```

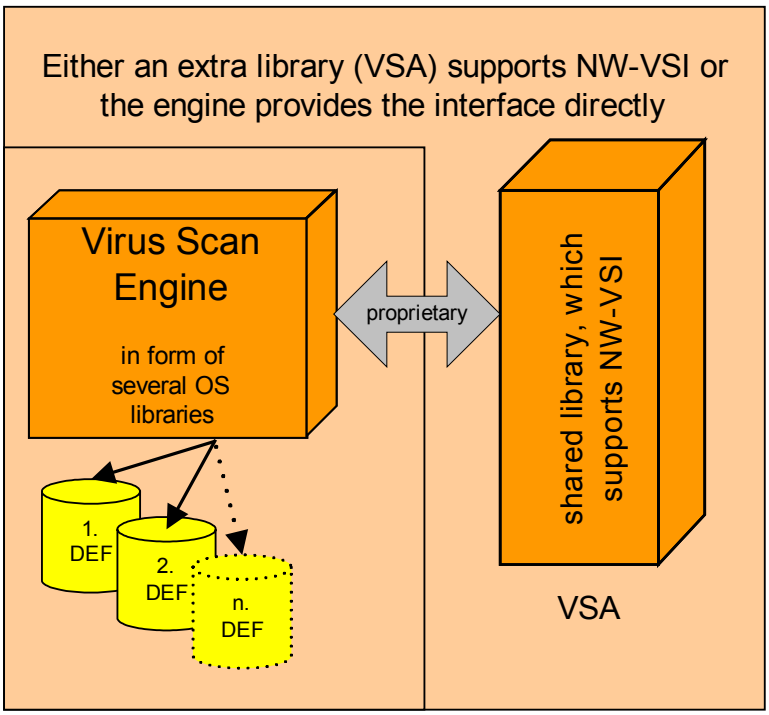
SAPCAR returns the following return codes when it is used:

Return Code	Description
0	no error
3	can't create directory
4	write error (a write failed in that the correct amount of data could not be written)
5	read error (SAPCAR could only read fewer bytes than it should)
6	error opening file
7	directory name is too long
8	can't use permission
9	can't compress
11	can't decompress
12	checksum error
13	can't change permission of file
14	can't change date of file
15	can't change permission of directory
16	can't change date of directory
17	can't query current working directory
18	use option only once
19	command option without filename specified
20	command option unknown
21	no command given
22	don't use C-option together with P-option

23	use option -C only with create (-c)
24	use option -T only with create (-c)
25	use option -X only with create (-c)
26	use option -R only with extract (-x)
27	p-option needs an octal value
28	can't open for writing
29	no filename(s) specified
30	free space check failed
31	use option -s only with printing archive content (-t)
36	error creating a soft link
37	unknown file type encountered
38	can not get free disk space
39	wrong format for -f option
40	some files could not be extracted
41	format error in -T file
42	format error in -A file
43	format error in -X file
44	at least one file was not available for archiving
45	symbolic link/shortcut without a target encountered
46	use option -lower only with extract (-x)
>100	system error codes that SAPCAR does not map on its own error codes (like most standard system errors)

9 Update Procedure for Partner Components

994 The title refers to the updating of signature files (DEF) for a virus scan engine and the scan engine itself,
995 depending on the partner architecture. The update of the VSA itself is not covered, since this ressources
996 are in use even if scan instances are released by SAP applications.



DEF:=definition files, pattern files, signature files (DATs), and so on ...

997 Depending on the architecture of the virus scan adapter, it can be necessary to release all externally-ini-
998 tialized resources before new byte signatures for the virus scan can be loaded. In both cases, the update
999 must be performed by components of the adapter vendor. There is therefore no interface for the update in
1000 the design of NW-VSI.

1001 The following describes the actions that are necessary for each type of update. In both cases, however, it
1002 is necessary to inform the internal SAP scan API about an update.

1003 9.1 SAP Configuration

1004 In this case, the scan instances are unloaded and loaded again using a value in the configuration of SAP.
1005 This reload action releaes all VSA_INIT handles calling function VsaEnd and then creates new instances.

1006 Partners should care about this update and should provide customers help in setting here a reasonable
1007 reload interval. SAP recommends here 24 hours, means once per day.

1008 Customers see here an advantage and often use this mechanism. However for partner update processes
1009 this is not visible and therefore at-hoc updates are not supported with this approach.

9.2 SAP Process Notification

The information about an update of the signature files can be communicated to the application using the signal "**SIGHUP**". The adapter can trigger this signal using the function "raise(...)". Since the VSA always runs in the process space of the SAP application, this signal can also be transferred in Microsoft Windows systems. The "kill" command is otherwise only known under UNIX platforms.

Within the SAP applications (see section 1.2), the signal SIGHUP is used through installed signal handlers to release all open scan instances using VsaEnd and reinitialize in each case using the VsaInit function.

The disadvantage of this approach is that new integrations in new SAP server processes (beside SAP NetWeaver) have to react on signals and this cannot be ensured always.

If the update with signals is working, its an advantage for partner update programs, because their update program simply can look for certain proceses in the environment and if they are running, they can send the SIGUP signal to them and ensure an at-hoc update of SAP instances.

9.2 Virus Scan Adapter Notification

In this case, the VSA can use the return code VSA_E_EXPIRED also in VsaScan or in a Callback Message to notify the SAP application layer.

The scan action will be repeated if this error code (VSA_E_EXPIRED) is returned. However the second trail must not return with this code, because then the scan fails.

10 Certification Criteria

The following list displays the criteria required for certification. It is a prerequisite that the interface was implemented using the C header file **VSAXXTYP.H**, that is, the functions from section 4 were provided in a dynamic library.

Please remark:

NW-VSI 1.00 required only the aspects from 10.1.

NW-VSI 2.00 consists of requirements from10.1 and 10.2.

10.1 Virus Scan

A VSA must fulfill the following features for the virus scan:

- VSA_SP_FILE

  It must be possible to process at least one file.

- VSA_AP_SCAN

  The virus scan of an object must be provided, in which the EICAR virus **must** be found, and an SAP text fragment **must** be recognized, as a virus-free object, as “clean”.

The following conventions must be observed:

- The structure VSA_CONFIG must be returned for the call VsaGetConfig :
  - Parameter `struct_size` must specify the byte size of the structure.
  - Parameter `uiVsaScanFlags` must contain ***at least*** the bit value of VSA_SP_FILE.
  - Parameter `uiVsaActionFlags` must contain ***at least*** the bit value of VSA_AP_SCAN
  - Parameter `pAdapterInfo` must contain an initialized and filled VSA_ADAPTERINFO structure
  - Parameter `pOptParams` must contain ***at least*** one structure of VSA_OPTPARAM, containing at least the optional parameter
    - VS_OP_SCANBESTEFFORT. The meaning of this parameter is the best-possible scan quality from a security point of view.
    - VS_OP_SCANEXTRACT
    - VS_OP_SCANMIMETYPES
    - VS_OP_SCANEXTENSIONS
    - VS_OP_BLOCKMIMETYPES
    - VS_OP_BLOCKEXTENSIONS

- The VSA_ADAPTERINFO structure must contain the following within VSA_CONFIG:
  - Parameter `struct_size` must specify the byte size of the structure.
  - Parameter `tAdapterID` must contain a value for VS_ADAPTER_T, for which a new ID is set by SAP before a certification and this must be valid in the VSA when the certification is made.
  - Parameter `tThreadingModel` must be 0,1, or 2, the meaning of which can be read in section 3.

- Parameter `usVsiVersion` must currently return 1 or 2. 1 is recommended as long as customer has not implemented SAP Note 1796762
- Parameters `usVsaMajVersion` and `usVsaMinVersion` must total more than 0. This is to ensure that the vendor specifies a VSA version that, although freely definable, must be set so that if customer problems occur later, the version can be communicated to the vendor.

- The structure VSA_INIT must be returned when VsaInit is called:
  - Parameter `struct_size` must specify the byte size of the structure.
  - Parameter `hEngine` must not be NULL at VSA_OK, that is, successful initialization.
  - If parameter `usDriver` is greater than 0, the structure VSA_DRIVERINFO must be returned.

- The structure VSA_DRIVERINFO must contain the following when returning driver information:
  - Parameter `struct_size` must specify the byte size of the structure.

- The structure VSA_SCANINFO must contain the following when VsaScan is called and a pointer to PVSA_SCANINFO is set:
  - Parameter `struct_size` must specify the byte size of the structure.
  - Set value for the scan run (uiScanned, uiNotScanned, uiClean, uiInfections, uiScanErrors)
  - Contained structures of VSA_VIRUSINFO and VSA_SCANERROR must each also return at least the parameter `struct_size` with the byte size of the structure.

- If an event callback is supported, the parameter `uiVsaEvtMsgFlags` in VSA_CONFIG must contain at least one message code from VS_MESSAGE_T. The message code VS_M_ALL must also be implemented here for all supported events.
- If the parameter VS_OP_SCANBESTEFFORT is set, the EICAR test virus must be identified, regardless of whether the file is called "eicar.com" or "eicar.txt", that is, in this case, all internal parameters that force a scan must be set. If a scan is not possible, the VSA must react with VSA_E_NOT_SCANNED.
- For all scan actions or clean actions, the VSA must react with the return code VSA_E_NOT_SCANNED if the object cannot be completely scanned in the engine. This does not mean that this return code should always be returned, since a virus can almost never be ruled out, but rather this should show that an engine cannot scan the inner elements of an object. Example: An archive cannot be unpacked. For more information, see also the note in section 4.1.1.
- An SAP SAR/CAR file must be identified as such, that is, VSA_OK can only be returned for a scan in this case if it was possible to unpack this file and the contents of it were identified as “clean”. If support for SAPCAR was not implemented in the adapter, VSA_E_NOT_SCANNED must at least be set as the return code here.
- The external product including the virus scan adapter must contain an installation and documentation. During certification, the setting up of the product is also checked, to avoid later problem messages on the SAP side. For the inclusion of the virus scan adapter into the SAP application, either link to the SAP documentation in the SAP Help Portal http://help.sap.com/ →

SAP NetWeaver → Security → System Security → Virus Scan Interface, or set an environment variable "**VSA_LIB**" with the complete path of the adapter (such as "c:\path\adapter.dll").

10.2 Content Classification

The content scan is an extension of the virus scan and therefore the features and conventions of the virus scan are a prerequisite for it. A VSA must also additionally fulfill the following features for the content scan:

- VSA_SP_FILE

It must be possible to process following.

- VSA_AP_BLOCKACTIVECONTENT (block script embedded in objects, e.g. Jscript in HTML)
- VSA_AP_CHECKMIMETYPE (MIME Detection and Filtering)

The content scan of an object must be offered, that is, at least the structure VSA_CONTENTINFO must be returned with information about the object.

The following conventions must be observed:

- The structure VSA_SCANINFO must contain the following when VsaScan is called and a pointer to PVSA_SCANINFO is set:
  - Parameter `struct_size` must specify the byte size of the structure.
  - Set values for the scan run (uiScanned, uiNotScanned, uiClean, uiInfections, uiScanErrors)
  - In the case of VSA_OK (success): At least a contained VSA_CONTENTINFO structure must be returned with at least the parameter `struct_size` with the byte size of the structure, and:
    - Parameter `tObjectType` must return a valid value from VS_OBJECT_TYPE. "Plausibility checks" with various files are performed during the certification.
    - Parameter `pszExtension` must contain the file extension or be NULL.
    - Parameter `pszExtension` must contain the file extension or be NULL.
    - Parameter `pszObjectName` must specify the name of the object. This must be the name from VSA_SCANPARAM or a valid file name in the case of the VSA_SP_FILE action.
    - Parameter `tObjectSize` must contain the byte size of the object. In the case of VSA_SP_FILE, the file size must be determined here.

- The return code VSA_E_NOT_SCANNED must be returned for scan actions, if it is not possible to scan or analyze the complete object in the engine and to assign a content type. The constant VSA_E_NOT_KNOWN has been defined for the content scan. This is technically equivalent to VSA_E_NOT_SCANNED.

10.3 Web Content Filter

The web content filter is only used in web-comonents of SAP. The virus scan is no must here. A VSA must also fulfill the following features for the content scan:

- VSA_AP_SCANCONTENT (regular expression engine for web/url-filter)

The content scan of an object must be offered, that is, at least the structure VSA_CONTENTINFO must be returned with information about the object.

11 Outlook

This chapter briefly describes the usage of NW-VSI for future.

The interface was designed to perform a check on SAP oexternal content. Since the interface integration in SAP applications in done very deeply into the stack and the enforcement of this is high, there are ideas to use this interface for data leakage prevention (DLP).

The interface is available in the in-coming and out-going channels of the SAP stack. Therefore a DLP solution could be placed in parallel to a AV solution in cases where documents leave the SAP system.

The usage for this scenario affects the API of the Virus Scan Adapter, because a new function, currently called VsaSetConfig is needed to specify the environment of the SAP system. A VSA which acts as DLP should know some application environments, e.g.