



Intel® Active Management Technology Storage Design Guide

Version 3.0.5, June 2007

Information in this document is provided in connection with Intel products. No license, express or implied, by estoppels or otherwise, to any intellectual property rights is granted by this document. Except as provided in Intel's Terms and Conditions of Sale for such products, Intel assumes no liability whatsoever, and Intel disclaims any express or implied warranty, relating to sale and/or use of Intel products including liability or warranties relating to fitness for a particular purpose, merchantability, or infringement of any patent, copyright or other intellectual property right. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

The API and software may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

Copyright © 2005, 2006, 2007 Intel Corporation.

* Third party other names and brands may be claimed as the property of others.

Table of Contents

1	Introduction.....	5
2	Reference Documents.....	5
3	Library Source Code	5
4	Storage Library Overview.....	5
4.1	ISV Application General Information.....	6
4.1.1	Name Registration	7
4.1.2	Request-Response Protocol.....	7
4.1.3	Function Fragmentation	7
4.1.4	Remote Communication	8
4.2	Partners, Enterprises and the Storage Administration Interface.....	8
4.3	General Assumptions.....	8
4.3.1	Concurrency.....	9
4.3.1.1	API Access Serialization.....	9
4.3.1.2	Application Multiplicity	9
4.3.1.3	Storage Operations	9
4.3.1.4	Function calling scheme	9
4.3.2	Data Confidentiality.....	9
4.4	Naming Conventions.....	9
4.4.1	Parameters	9
4.4.2	Commonly Used Terms	9
4.4.3	Function and Data Type Naming Convention.....	10
4.4.3.1	PT_ Data Types Prefix	10
4.4.3.2	ISVS_ Data Prefix.....	10
4.4.3.3	PTSDK_ Data Types Prefix.....	10
4.5	Data Structures	10
4.5.1	PT_UUID.....	10
4.5.2	PTSDK_APPLICATION_ATTRIBUTES.....	10
4.5.3	ISVS_APPLICATION_HANDLE	10
4.5.4	ISVS_INVALID_HANDLE	11
4.5.5	PTSDK_BLOCK_ATTRIBUTES	11
4.5.6	ISVS_BLOCK_HANDLE.....	11
4.5.7	ISVS_GROUP_HANDLE.....	11
4.5.8	ISVS_GROUP_PERMISSIONS	11
4.5.9	PTSDK_PERMISSIONS_GROUP_ATTRIBUTES	12
4.5.10	PT_BOOLEAN	12
4.5.11	PT_STATUS	12
4.5.12	SESSION_AUTHENTICATION_INFO.....	12
4.5.13	Shared Error Codes.....	12
4.5.14	Specific Error Codes.....	13
4.5.15	ISVS_VERSION.....	15
4.5.16	PTSDK_UNICODE_STRING.....	15
4.5.17	SESSION_HANDLE	16
4.6	Non-Volatile Storage API	16
4.6.1	Initialization and Registration Functions	16
4.6.1.1	ISVS_Initialize.....	16
4.6.1.2	ISVS_Uninitialize	17
4.6.1.3	ISVS_GetHostUUID.....	18
4.6.1.4	ISVS_CreateAuthInfo	18
4.6.1.5	ISVS_FreeAuthInfo.....	19
4.6.1.6	ISVS_RegisterApplication	20
4.6.1.7	ISVS_RegisterApplicationEx	21
4.6.1.8	ISVS_UnregisterApplication	24
4.6.1.9	ISVS_RemoveApplication	25

4.6.1.10	ISVS_InitializeCOMinThread.....	25
4.6.1.11	ISVS_UninitializeCOMinThread.....	26
4.6.2	Storage Configuration.....	26
4.6.2.1	ISVS_GetRegisteredApplications.....	28
4.6.2.2	ISVS_GetCurrentApplicationHandle.....	29
4.6.2.3	ISVS_GetApplicationAttributes.....	29
4.6.2.4	ISVS_AllocateBlock.....	30
4.6.2.5	ISVS_DeallocateBlock.....	31
4.6.2.6	ISVS_SetBlockVisibility.....	32
4.6.2.7	ISVS_SetBlockName.....	32
4.6.3	Data Storage Functions.....	33
4.6.3.1	ISVS_GetAllocatedBlocks.....	34
4.6.3.2	ISVS_GetBlockAttributes.....	36
4.6.3.3	ISVS_LockBlock.....	36
4.6.3.4	ISVS_UnlockBlock.....	37
4.6.3.5	ISVS_ReadBlock.....	38
4.6.3.6	ISVS_WriteBlock.....	39
4.6.3.7	ISVS_GetBlockWriteEraseLimit.....	40
4.6.4	Permissions.....	41
4.6.4.1	ISVS_GetPermissionsGroups.....	42
4.6.4.2	ISVS_GetPermissionsGroupAttributes.....	43
4.6.4.3	ISVS_AddPermissionsGroup.....	44
4.6.4.4	ISVS_RemovePermissionsGroup.....	45
4.6.4.5	ISVS_AddPermissionsGroupMembers.....	46
4.6.4.6	ISVS_RemovePermissionsGroupMembers.....	47
4.6.4.7	ISVS_SetPermissionsGroupName.....	48
4.6.4.8	ISVS_GetPermissionsGroupMembers.....	49
4.6.4.9	ISVS_SetPermissionsGroupPermissions.....	50
4.6.5	General Functions.....	51
4.6.5.1	ISVS_GetTimeoutValues.....	51
4.6.5.2	ISVS_GetLastNetworkError.....	52
4.6.5.3	ISVS_GetAPIVersion.....	52
4.6.5.4	ISVS_GetAPIVersionEx.....	53
4.6.5.5	ISVS_GetBytesAvailable.....	55
5	Appendix A: Application Migration Flow-Chart.....	56

1 Introduction

The ISV Storage Library is an API used to access the Intel® Active Management Technology (Intel AMT) non-volatile storage feature. Independent Software Vendors (ISVs) use this API to allocate, maintain and specify access rights to non-volatile memory blocks maintained locally by Intel AMT.

The data placed in a non-volatile memory block is persistent. As a result, the data can be accessed while the OS is not functioning or between OS installations provided the system is physically connected to power and has a network outlet. Both local and remote ISV applications can access and update stored data. Also, multiple applications from the same vendor can share stored data.

2 Reference Documents

The following documents are companion and supporting specifications for the ISV Storage interface:

[SMBIOS]	System Management BIOS (SMBIOS) Reference Specification, v2.3.4, December 6, 2002.
[UNICODE]	The Unicode Consortium. The Unicode Standard, Version 4.0.0, defined by: The Unicode Standard, Version 4.0 (Boston, MA, Addison-Wesley, 2003. ISBN 0-321-18578-1).
[IPMI]	IPMI - Intelligent Platform Management Interface Specification v1.5

In addition, the Intel AMT Software Development Kit (SDK) contains extensive sample code demonstrating the use of ISV Storage.

3 Library Source Code

The source code for the ISV Storage library is supplied as part of the Intel AMT SDK. It is intended for use as a reference to assist in the development of applications that use the storage library. Additionally this will allow ISVs to compile the library for use with operating systems for which a version of the library is not supplied as part of the SDK. The ISV Storage library source code is in the folder “**Windows\Intel AMT SDK\Src\StorageLib\LIBCODE**” and in the expanded Linux tar file in the folder “**IntelAMTSDK\Src\StorageLib\LIBCODE**” in the SDK.

The library source code is not meant for ISV modification, reuse or distribution and remains the intellectual property of Intel Corporation.

The library comes with Microsoft Visual Studio* .NET 2003 and 2005 projects for use under Microsoft Windows* environments. For instructions on building the library for x64 environments, see “Working with the x64 version of the ISV Storage Library” in the *Intel AMT SDK User Guide*.

The SDK contains a make file for Linux that supports multiple Linux flavors. It is supplied as a sample, and some configuration may be necessary to compile the Library in any given Linux environment.

4 Storage Library Overview

The Intel Active Management Technology storage library focuses on four primary tasks:

- Initialization and Registration – initializing the storage library locally within an application and registering the application with Intel AMT.
- Storage Configuration – Interactions between the application and Intel AMT to allocate or deallocate storage and to give visibility to other applications so that they can see allocated non-volatile memory blocks.
- Accessing Non-Volatile Storage – functions to write to or read from non-volatile storage.
- Access Privileges - functions that assign group privileges to other applications to read and/or write memory blocks

In Intel AMT Release 2.0 and later releases, when local applications use the storage library, the library communicates with the local Intel AMT device the same way that remote applications do, by encapsulating storage commands in a SOAP/HTTP message. However, when the library sends a SOAP/HTTP message addressed to the local Intel AMT host name, the Local Manageability Service (LMS), which listens for traffic directed to the host name, intercepts the message and routes it to the Intel® Management Engine Interface driver. The Intel Management Engine Interface multi-threaded connection-based driver communicates over the Intel Management Engine Interface, where the counterpart driver receives the information and passes it to the Intel AMT embedded IP stack. The Intel Management Engine Interface driver supports multiple connections operating simultaneously. If the driver is interrupted because the CPU has gone into a “hibernate” state, all connections are terminated and the Intel Management Engine Interface driver returns all pending requests with an error. Application developers should take this into account and provide for error handling. Applications can also register for power events and not perform operations during events such as being in a “hibernate,” “standby,” or other low power state.

In Intel AMT Release 1.0 the KCS driver completes the current transaction consisting of a single step and leaves any remaining transactions in the queue.

The Intel-provided static library must be compiled with the ISV application, as it provides all of the logic to communicate with the Intel AMT-enabled hardware and firmware.

4.1 ISV Application General Information

The Intel AMT Storage Manager is responsible for ensuring that registered ISV applications receive appropriate access to non-volatile storage. An ISV application may either be a local ISV agent or a remote console application. The ISV application interface with the Storage Manager is a “C” language based API (C-API) interface, making the interface operating system independent.

Remote applications perform storage operations over a network using SOAP over HTTP messages. The storage library implements all necessary network operations.

Local applications communicate differently with Intel AMT depending on the version of the SDK and the version of Intel AMT.

- Early versions of the SDK that supported the Intel AMT Release 1.0 architecture required that the URL in a registration call was NULL, and connected using WMI and the KCS driver. A local application compiled with an early SDK library will run with Intel AMT Release 2.0 or later releases when it is configured in AMT Release 1.0 legacy mode.
- An application compiled with SDK version 2.7 but **modified** to include a URL and user credentials in the registration call **and** compiled with an SDK for Intel AMT Release 2.0 or later will work with platforms running Intel AMT Release 2.0 or later and can take advantage of secure communications between the application and Intel AMT that is available with Release 2.0 or later releases.
- Applications run locally that are compiled with the Intel AMT Release 2.0 SDK or the SDK that supports later releases, are compatible with all existing versions of Intel AMT Release

1.0 hardware. They will connect using WMI to Intel AMT Release 1.0 machines and to Intel devices with AMT Release 2.0 or later running in legacy mode. They will connect using SOAP/HTTP over LMS to devices with Intel AMT Release 2.0 or later running in normal mode.

The following table summarizes these combinations.

Storage Library SDK Version	Intel AMT Release 1.0 Architecture	Intel AMT Release 2.0 or greater in AMT Release 1.0 Legacy Mode	Device with Intel AMT Release 2.0 or greater
2.7 or below (Intel AMT 1.0 only)	Uses WMI and KCS driver (no URL in registration call)	Uses WMI and Intel Management Engine Interface driver.	No URL in registration call: Not Compatible
			URL defined in registration call: uses SOAP/HTTP, LMS and Intel Management Engine Interface driver.
Later versions	Uses WMI	Uses WMI	Uses SOAP/HTTP, LMS and Intel Management Engine Interface driver.

Storage Library limitation: The WMI default permissions in Windows Vista (both regular and x64 Edition) and Windows XP x64 Edition do not allow user accounts to perform the operations required by the storage library. This applies to all applications and versions. If the user executing a local storage application does not have Administrator permissions on the platform, the operation will fail. This is true also for Intel AMT Release 2.0 or greater, where SOAP is used by applications exclusively, because the storage library uses WMI internally.

4.1.1 Name Registration

An ISV application uniquely identifies itself to the Intel AMT firmware when it registers using a combination of Vendor Name, Application Name, Enterprise Name and the machine's UUID (Universally Unique Identifier) which identifies the host on which the application is running. This combination enables the Intel AMT Storage Manager to differentiate between instances of the same executable application running on different hosts (e.g. the local host and a remote host, or different remote hosts). Unique identification is essential to prevent corruption of storage and session data in the Intel AMT storage manager when different instances of the same application attempt to access it. An application instance should use the same UUID in every registration process to enable the Intel AMT storage manager to identify the application correctly in subsequent registrations. The library uses these values as input to the registration protocol. When the registration step has been completed the library acquires a Session Integrity key (SIK), and all subsequent calls sent by the application to the Intel AMT enabled hardware are protected using a session Integrity Check Value (ICV) with the SIKs.

4.1.2 Request-Response Protocol

The library transforms the C-API type application input to an API defined Request-Response buffer pair. The input values (including the function type) are translated to a request buffer, which the library sends to the Intel AMT-enabled hardware. The associated response buffer is transformed to a C-API type output value before being handed to the application.

4.1.3 Function Fragmentation

The size of the API buffer used to communicate with the Intel AMT device is 4KB. The library provides fragmentation support for storage block Read or Write requests that exceed this buffer limit. Read or Write requests that exceed the 4KB maximum are fragmented into multiple request-response pairs, relieving the ISV application of the burden of implementing fragmentation support.

4.1.4 Remote Communication

The storage library performs all network tasks required to complete storage requests, including implementation of the SSL protocol. The calling application is responsible for providing the complete URL to the remote Intel AMT-enabled hardware's storage service. The application can use the following static URL <http://hostname:16992/StorageService> or <https://hostname:16993/StorageService> (use of http or https depends on the security settings of the device). In AMT Release 2.0 or greater, all communication, even when the application is on the local machine, is performed over the network protocol and a URL must be supplied. When working in https the URL must be the local machine's actual name as declared in the Server Certificate assigned to that machine.

4.2 Partners, Enterprises and the Storage Administration Interface

The third-party data storage (3PDS) that is accessed with the ISV Storage Library is divided into two areas: Partner Storage and Non-Partner Storage.

Partner storage is pre-allocated in the Intel AMT firmware. As described in the *Network Interface Guide*, the list of partner and non-partner applications can be accessed using the commands of the Storage Administration Interface. `EnumerateStorageAllocEntries` returns a list of handles to entries in the Factory Partner Access Control List (FPACL). `GetStorageAllocEntry` returns the parameters associated with an entry.

The amount of storage available to a specific application is limited by the amount allocated to it in the FPACL, and by the total amount available to partner and non-partner applications. The sum of memory allocated to all partner applications is greater than the amount actually available to the applications – the partner space is “overbooked” based on the assumption that a single enterprise will not deploy every possible ISV application. Storage Administration commands can be used to change the value allocated to different applications. If there are unallocated blocks in the non-partner partition, `ISVS_AllocateBlock` will allocate blocks to partner applications from this area if the entire partner partition has been allocated.

Non-partner applications are those that are not in the FPACL. Their storage is allocated from the non-partner partition only. The Storage Administration Interface command `GetGlobalStorageAttributes` returns the current allocations. The command `SetGlobalStorageAttributes` can be used to redistribute the available storage between partner and non-partner applications.

The Enterprise Access Control List (EACL) lists enterprises that are allowed to register applications using `ISVS_RegisterApplication` or `ISVS_RegisterApplicationEx`. The `EnterpriseName` is an input parameter to these functions. There must be at least one entry in the EACL so that the storage library is able to register applications. The EACL is also managed with functions that are part of the Storage Administration Interface.

4.3 General Assumptions

Developers should observe the following guidelines when creating applications that use the Intel AMT storage capability.

4.3.1 Concurrency

4.3.1.1 API Access Serialization

The caller should serialize access to all of the API functions for a given registered application (i.e. all accesses using identical identification data: Vendor, application, Enterprise names and UUID).

4.3.1.2 Application Multiplicity

Accessing an Intel AMT enabled device with identical registration information from multiple processes running concurrently, either on the same host or from different hosts is invalid. Due to the storage registration protocol, the session key used for a given registered application session cannot be shared between multiple consumers.

4.3.1.3 Storage Operations

Unless stated otherwise, all storage operations can only be performed by an application that has first successfully registered by calling [ISVS_RegisterApplicationEx](#) or [ISVS_RegisterApplication](#).

4.3.1.4 Function calling scheme

Unless stated otherwise:

All output parameters in the API functions are pointers to variables allocated by the calling function and are filled by the API function. If the passed argument includes an embedded pointer (e.g. `PTSDK_UNICODE_STRING`) this pointer should be initialized, as well, by the caller.

Pointers should never be NULL.

All Output variables are valid only on return status `PT_STATUS_SUCCESS`.

4.3.2 Data Confidentiality

ISV applications that write data to the 3PDS area of the flash must perform their own data protection. Intel AMT does not encrypt this information, and it is stored in the form that it was sent. It is up to the ISV application to properly protect any sensitive information written into this repository.

4.4 Naming Conventions

The following naming and style conventions are used to describe the storage function library.

4.4.1 Parameters

Input: Input parameters in API functions.
 Output: Output parameters in API functions.
 Input/Output: Input/Output parameters in API functions.
 Optional: Some parameters are denoted as Optional in their description. This designation is applicable to pointer parameters only if the calling application can validly set an Optional pointer parameter to NULL, in which case the library will perform a default behavior described by the specific API.

A `fixed-width font` is used to distinguish pseudo-code from other text.

4.4.2 Commonly Used Terms

The following terms are used in data structure and function descriptions.

A **Session** is the period of time between registering and un-registering an application with the Intel AMT device, via the [ISVS_RegisterApplicationEx](#) or [ISVS_RegisterApplication](#) and [ISVS_UnregisterApplication](#) functions.

An **Application** is a software entity that registers with an Intel AMT device. The application could be either a local agent or a remote console application.

An **Agent** is an application that executes locally on the same platform as the Intel AMT device.

4.4.3 Function and Data Type Naming Convention

4.4.3.1 PT_ Data Types Prefix

All functions and data types prefixed with “PT_” describe general Intel AMT functions and data types.

4.4.3.2 ISVS_ Data Prefix

All functions and data types prefixed with “ISVS_” describe specific ISV Storage-related functions and data types.

4.4.3.3 PTSDK_ Data Types Prefix

All Data Types prefixed with “PTSDK_” describe specific data types used by the storage library.

4.5 Data Structures

4.5.1 PT_UUID

PT_UUID is defined to represent Universally Unique Identifier (UUID). UUIDs are used as part of the application registration process.

```
typedef UINT8 PT_UUID[16];
```

4.5.2 PTSDK_APPLICATION_ATTRIBUTES

PTSDK_APPLICATION_ATTRIBUTES is defined to represent the attributes associated with a registered ISV application.

```
typedef struct _PTSDK_APPLICATION_ATTRIBUTES
{
    PTSDK_UNICODE_STRING    VendorName;
    PTSDK_UNICODE_STRING    ApplicationName;
} PTSDK_APPLICATION_ATTRIBUTES;
```

Field	Value or Description
VendorName	The name of the vendor, as specified by the vendor and provided in the ISVS_RegisterApplicationEx.
ApplicationName	The name of the ISV application, as specified by the vendor and provided in the ISVS_RegisterApplicationEx.

4.5.3 ISVS_APPLICATION_HANDLE

Intel AMT assigns a handle to each application as it registers (see [ISVS_RegisterApplicationEx](#)). This handle is used to reference an application in subsequent third-party storage operations. Once an application handle value is assigned to an ISV application, it remains permanently associated with that application, even when it isn’t registered, as long as the application has not been removed. This allows any application to reference another application, whether it is currently registered or not.

```
typedef uint32 ISVS_APPLICATION_HANDLE;
```

4.5.4 ISVS_INVALID_HANDLE

ISVS_INVALID_HANDLE is used to indicate an invalid handle value. This value is not used by Intel AMT as a valid block, application, or permission group handle.

```
#define ISVS_INVALID_HANDLE 0xFFFFFFFF
```

4.5.5 PTSDK_BLOCK_ATTRIBUTES

ISVS_BLOCK_ATTRIBUTES lists the attributes associated with a block of ISV storage.

```
typedef struct _PTSDK_BLOCK_ATTRIBUTES
{
    uint32                BlockSize;
    PT_BOOLEAN            BlockHidden;
    PTSDK_UNICODE_STRING BlockName;
} PTSDK_BLOCK_ATTRIBUTES;
```

Field	Value or Description
BlockSize	The size of the block in bytes
BlockHidden	PT_TRUE if block is hidden from other applications, PT_FALSE otherwise
BlockName	An optional name for the block, set by the owner

4.5.6 ISVS_BLOCK_HANDLE

Intel AMT assigns a handle to each allocated block of ISV storage. This handle is used to reference the block in storage operations.

```
typedef uint32 ISVS_BLOCK_HANDLE;
```

4.5.7 ISVS_GROUP_HANDLE

Intel AMT assigns a handle to each ISV storage permissions group. This handle is used to reference the group in ISV storage operations.

```
typedef uint32 ISVS_GROUP_HANDLE;
```

4.5.8 ISVS_GROUP_PERMISSIONS

ISVS_GROUP_PERMISSIONS is defined to represent the permissions associated with a block of ISV storage.

```
typedef enum _ISVS_GROUP_PERMISSIONS
{
    ISVS_GROUP_PERMISSIONS_READ_ONLY = 1,
    ISVS_GROUP_PERMISSIONS_READ_WRITE = 2
} ISVS_GROUP_PERMISSIONS;
```

Code	Value or Description
ISVS_GROUP_PERMISSIONS_READ_ONLY	Group has permission to read the block, but not to write it
ISVS_GROUP_PERMISSIONS_READ_WRITE	Group has permission to read and write the block

4.5.9 PTSDK_PERMISSIONS_GROUP_ATTRIBUTES

PTSDK_PERMISSIONS_GROUP_ATTRIBUTES is defined to represent the attributes associated with a storage block permissions group.

```
typedef struct _PTSDK_PERMISSIONS_GROUP_ATTRIBUTES
{
    PTSDK_UNICODE_STRING      Name;
    ISVS_GROUP_PERMISSIONS    Permissions;
} PTSDK_PERMISSIONS_GROUP_ATTRIBUTES;
```

Field	Value or Description
Permissions	The permission flags for this group
Name	Name of the permissions group.

4.5.10 PT_BOOLEAN

PT_BOOLEAN is defined to represent Boolean values 'true' and 'false'.

```
typedef uint8 PT_BOOLEAN;
const PT_BOOLEAN PT_FALSE = 0;
const PT_BOOLEAN PT_TRUE = 1;
```

Code	Value or Description
PT_FALSE	Boolean True.
PT_TRUE	Boolean False.

4.5.11 PT_STATUS

PT_STATUS defines status codes that are returned by the ISV Storage API calls.

```
typedef uint32 PT_STATUS;
```

4.5.12 SESSION_AUTHENTICATION_INFO

SESSION_AUTHENTICATION_INFO represents a handle to an internal API data structure that holds the OS specific information needed to support mutual authentication. It also identifies the HTTP authentication scheme (Digest or Kerberos). For more information on mutual authentication, refer to the *Network Interface Guide* document within the SDK.

See [ISVS_CreateAuthInfo](#) for information on obtaining an SESSION_AUTHENTICATION_INFO pointer.

```
typedef struct _SESSION_AUTHENTICATION_INFO
SESSION_AUTHENTICATION_INFO;
```

4.5.13 Shared Error Codes

Unless stated otherwise each function call can return the following error codes:

Code	Value or Description
PT_STATUS_SUCCESS	The requested operation was successfully executed.

Code	Value or Description
PT_STATUS_INTEGRITY_CHECK_FAILED	The Integrity Check Value field of the request message sent by Intel AMT enabled device is invalid. This can either indicate that the response message was tampered with, or that the application has exceeded the number of messages allowed in a single session (2^{32}). The application should attempt to re-register with the Intel AMT enabled device.
PT_STATUS_INTERNAL_ERROR	Intel AMT has identified an internal HW error.
PTSDK_STATUS_INTERNAL_ERROR	An internal SDK error occurred.
PTSDK_STATUS_REQUESTOR_NOT_REGISTERED	The application that sent the request message is not registered. Usually indicates the registration timeout has elapsed. The caller should re-register with the Intel AMT enabled device.
PTSDK_STATUS_NOT_INITIALIZED	An ISV operation was called while the library was not initialized
PTSDK_STATUS_HARDWARE_ACCESS_ERROR	The library has identified a HW Internal error.
PTSDK_STATUS_NETWORK_ERROR	A network error has occurred while processing the call. The caller can get further error information by calling ISVS_GetLastNetworkError .
PTSDK_STATUS_RESOURCES	The SDK could not allocate sufficient resources to complete the operation.
PTSDK_STATUS_INVALID_PARAM	One of the parameters is invalid (usually indicates a NULL pointer or an invalid session handle is specified)
PTSDK_STATUS_COM_NOT_INITIALIZED_IN_THREAD	For Windows only. ISVS_InitializeCOMinThread was not called by the current thread.

4.5.14 Specific Error Codes

Code	Value or Description
PT_STATUS_UNSUPPORTED_ISVS_VERSION	The specified ISVS version is not supported

Code	Value or Description
PT_STATUS_INVALID_REGISTRATION_DATA	This error may occur in the following: Either an invalid name was entered or an "Enterprise" name was specified that was not pre-registered. The current registration was attempted from an interface different from the one used for the initial registration of the application.
PT_STATUS_APPLICATION_DOES_NOT_EXIST	The application handle provided in the request message is not valid.
PT_STATUS_NOT_ENOUGH_STORAGE	The number of bytes requested cannot be allocated in ISV storage.
PT_STATUS_INVALID_NAME	The specified name is invalid.
PT_STATUS_BLOCK_DOES_NOT_EXIST	The specified block does not exist.
PT_STATUS_INVALID_BYTE_OFFSET	The specified byte offset is invalid.
PT_STATUS_INVALID_BYTE_COUNT	The specified byte count is invalid.
PT_STATUS_NOT_PERMITTED	The requesting application is not permitted to request execution of the specified operation.
PT_STATUS_NOT_OWNER	The requesting application is not the owner of the block as required for the requested operation.
PT_STATUS_BLOCK_LOCKED_BY_OTHER	The specified block is locked by another application.
PT_STATUS_BLOCK_NOT_LOCKED	The specified block is not locked.
PT_STATUS_INVALID_GROUP_PERMISSIONS	The specified group permission bits are invalid.
PT_STATUS_GROUP_DOES_NOT_EXIST	The specified group does not exist.
PT_STATUS_INVALID_MEMBER_COUNT	The specified member count is invalid.
PT_STATUS_MAX_LIMIT_REACHED	The request cannot be satisfied because a maximum limit associated with the request has been reached.
PT_STATUS_INVALID_INDEX	An invalid numeration start index specified
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector
PTSDK_STATUS_PARAM_BUFFER_TOO_SHORT	Specified container can not hold the requested string

Code	Value or Description
PTSDK_STATUS_LIB_VERSION_UNSUPPORTED	The requested library I/F is not supported by the current library implementation.
PTSDK_STATUS_URL_REQUIRED	The URL parameter was not optional in current configuration.

4.5.15 ISVS_VERSION

ISVS_VERSION is a pair of numbers that identify the version of the API implemented by the installed Intel AMT firmware image.

```
typedef struct _ISVS_VERSION
{
    uint8    MajorNumber;
    uint8    MinorNumber;
} ISVS_VERSION;
```

Field	Value or Description
MajorNumber	The major version number. This number is updated only if a change in ISVS results in incompatibility with previous versions
MinorNumber	The minor version number – this number is updated if the ISVS is extended in a way that does not affect backwards compatibility

4.5.16 PTSDK_UNICODE_STRING

PTSDK_UNICODE_STRING is a Unicode string in UTF-16 format. Unicode strings are used by Intel AMT to describe Applications, Storage Blocks and Permission groups. The Intel AMT device maintains strings in UTF-8 format. As a result, the fixed length value of these strings can hold only a limited number of UTF-16 characters that require two or three bytes per character. The following table provides byte and estimated character limit for the various strings:

String Type	String Length (bytes)	Longest string in Latin alphabet characters (Standard ASCII), in UTF-8 encoding U+ 0000 – U+007F	Longest string in Cyrillic, Arabic, Hebrew characters in UTF-8 encoding U+0080 – U + 07FF	Longest string Chinese Alphabet in UTF-8 encoding U+0800 – U+FFFF
Vendor Name	64	64	32	21
Application Name	64	64	32	21
Enterprise Name	64	64	32	21
Block Name	32	32	16	10
Permissions Group Name	16	16	8	5

Note: If one of the values in the Buffer array, (Buffer[0] – Buffer[Length - 1]), provided by the application is a Null termination character ((UINT16)0), the library will consider the PTSDK_UNICODE_STRING as invalid and will fail the API call requested by the user.

```
typedef struct _PTSDK_UNICODE_STRING
{
    uint16    Length;
```

```

        uint16  MaximumLength;
        uint16  *Buffer;
    } PTSDK_UNICODE_STRING;

```

Field	Value or Description
Length	The length in bytes of the string stored in Buffer. If the string is NULL-terminated, Length does not include the trailing NULL.
MaximumLength	The maximum length in bytes of Buffer. If Buffer is NULL a 0 value must be set.
Buffer	Pointer to a buffer used to contain a string of Wide characters.

Note: This Unicode string definition is identical to the UNICODE_STRING definition in Microsoft's Windows Security Platform SDK

http://msdn.microsoft.com/library/en-us/secauthn/security/unicode_string.asp

4.5.17 SESSION_HANDLE

SESSION_HANDLE is a handle to an internal API data structure where Intel AMT maintains the state of an application session. This handle is created in the [ISVS_RegisterApplicationEx](#) or [ISVS_RegisterApplication](#) and is used in subsequent calls throughout the session. The session handle is invalid after calling [ISVS_UnregisterApplication](#) or [ISVS_RemoveApplication](#).

```
typedef _SESSION_HANDLE *SESSION_HANDLE;
```

4.6 Non-Volatile Storage API

There are four groups of storage API functions. In addition, there are several general API functions that support the storage API.

4.6.1 Initialization and Registration Functions

Use the registration process each time you initiate a session. First call ISVS_Initialize to activate the static library used for access to the Intel AMT firmware. Then register the application with Intel AMT. Immediately following the registration, the application will be able to use the non-volatile storage space or perform any logical operation in the Intel AMT device.

4.6.1.1 ISVS_Initialize

ISVS_Initialize is called by the ISV application to initialize the static library. This function must successfully return before making any other ISVS call.

Note: ISVS_Uninitialize must be called in the same thread that called ISVS_Initialize.

Function Header

```

PT_STATUS ISVS_Initialize (
    uint32 *LibMajorVersion,
    uint32 *LibMinorVersion,
    uint32 *LibBuildNumber
);

```

Function Parameters

Parameters	Input/Output	Description
------------	--------------	-------------

Parameters	Input/Output	Description
LibMajorVersion	Input/Output	Input: The current library major version assumed by the calling application – caller should set the value: ISVS_VERSION_MAJOR as defined in library header file, “...\Intel AMT SDK\Include\iamt_api.h” in the Windows directory, and “...\IntelAMTSDK\Include\iamt_api.h” in the Linux directory. Output: The actual library major version – The library will return an error status when input and output versions are incompatible. This value is valid on any return status code.
LibMinorVersion	Input/Output	Input: Current library minor version assumed by the calling application – caller should set the value: ISVS_VERSION_MINOR as defined in library header file, “...\Intel AMT SDK\Include\iamt_api.h” in the Windows directory, and “...\IntelAMTSDK\Include\iamt_api.h” in the Linux directory. Output: The actual library minor version – The library will return an error status when input and output versions are incompatible. This value is valid on any return status code.
LibBuildNumber	Output	Indicates an incremental library update that does not incur an interface change. This value is valid on any return status code.

Function Return Status

Status	Description
PT_STATUS_SUCCESS	Request completed successfully.
PTSDK_STATUS_RESOURCES	The library could not allocate the resources needed to complete the operation.
PTSDK_STATUS_LIB_VERSION_UNSUPPORTED	The requested library I/F is not supported by the current library implementation

4.6.1.2 ISVS_Uninitialize

ISVS_Uninitialize is called by the ISV application to un-initialize a previously initialized library. The application should not be registered with any Intel AMT devices, remote or local, when calling this function. However, the storage library will attempt to silently unregister any sessions left open when ISVS_Uninitialize is called.

Note: ISVS_Uninitialize must be called by the same thread that called ISVS_Initialize.

Function Header

```
PT_STATUS ISVS_Uninitialize (
);
```

Function Parameters

Parameters	Input/output	Description
None	N/A	None

Function Return Status

Status	Description
--------	-------------

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PTSDK_NOT_INITIALIZED	The library is not initialized.

4.6.1.3 ISVS_GetHostUUID

ISVS_GetHostUUID is called by an ISV application to obtain either a copy of the host machine's UUID or a UUID generated by the library. If the machine supports SMBIOS version 2.1, then the function will return the machine's UUID; otherwise, it will return a generated UUID. The UUID is used along with additional naming input, to guarantee application namespace uniqueness for storage applications. The application passes the value returned from this to the [ISVS_RegisterApplicationEx](#) command along with other naming input. To facilitate recovery operations, the application should save the value returned from this call along with the other naming data in a file that is backed-up at the same time as other relevant application data. When an application sees that the UUID has been changed, due to an external, non-recoverable operation, it should migrate all data blocks associated with the "old", registered application, which registered with the old UUID, to a "newly" registered application using the new UUID. See [Appendix A](#) for a flow chart for migrating to a new UUID..

Function Header

```
PT_STATUS ISVS_GetHostUUID (
    PT_UUID      UUID
);
```

Function Parameters

Parameters	Input/output	Description
UUID	Output	Host machine UUID value

Function Return Status

Status	Description
PT_STATUS_SUCCESS	Request completed successfully.
PTSDK_NOT_INITIALIZED	The library is not initialized
PTSDK_STATUS_INVALID_PARAM	UUID points to NULL
PTSDK_STATUS_INTERNAL_ERROR	The library failed to acquire or store the Host UUID information.

4.6.1.4 ISVS_CreateAuthInfo

Create and return a handle to an internal API data structure that holds the OS specific information needed to support mutual authentication. The handle is passed to ISVS_ResisterApplicationEx. The SESSION_AUTHENTICATION_INFO must be released at the end of the session after ISVS_UnregisterApplication has been called by calling ISVS_FreeAuthInfo. For more information on mutual authentication, refer to the *Network Interface Guide* document within the SDK.

Function Header for MS Windows environments:

```
SESSION_AUTHENTICATION_INFO* ISVS_CreateAuthInfo(
    TCHAR      *certificateName,
    PCCERT_CONTEXT certificate
    BOOL      krb
);
```

Function Parameters

Parameters	Input/output	Description
certificateName	Input	The Common Name attribute of client certificate to be retrieved for the certificate stored. If a Certificate is specified this will be ignored. In which case this may be NULL.
Certificate	Input	A certificate specification may be null if a certificateName is specified. If non-null this value will be used even if a certificateName is specified.
Krb	Input	If TRUE, authentication will be according to the Kerberos authentication method. If FALSE, Digest authentication will be used.

Note - certificateName and certificate should not both be null.

Function Header for Linux environments:

```
SESSION_AUTHENTICATION_INFO* ISVS_CreateAuthInfo(
    TCHAR *certificateName,
    TCHAR *certificatePass
);
```

Function Parameters

Parameters	Input/output	Description
certificateName	Input	The full path to a client certificate file
certificatePass	Input	The password to the client certificate file.

4.6.1.5 ISVS_FreeAuthInfo

Should be called once for every SESSION_AUTHENTICATION_INFO that was created to release the internal memory it occupies. This should only be done once there is no further need for the SESSION_AUTHENTICATION_INFO, typically after calling ISVS_UnregisterApplication.

Function Header

```
VOID ISVS_FreeAuthInfo(
    SESSION_AUTHENTICATION_INFO* AuthInfo
);
```

Function Parameters

Parameters	Input/output	Description
AuthInfo	Input	SESSION_AUTHENTICATION_INFO previously allocated by a call to ISVS_CreateAuthInfo

4.6.1.6 ISVS_RegisterApplication

*** This function has been deprecated. Use ISVS_RegisterApplicationEx in its place.**

An ISV application calls `ISVS_RegisterApplication` to register the application with Intel AMT. The ISV application must register with Intel AMT each time it initializes the storage library. Following a successful completion, Intel AMT returns a Session handle to the application. The application specifies this handle in subsequent ISV storage calls until it sends a new registration request (with identical information).

Once an application has initially registered with unique registration information ("Vendor Name", "Application Name", Enterprise Name" and UUID) from either the local or the remote interface, Intel AMT will reject registration attempts with identical registration data from a different interface. This prevents masquerading of local agents from the network and vice versa.

Function Header

```
PT_STATUS ISVS_RegisterApplication (
    SESSION_HANDLE          *SessionHandle,
    wchar_t                 *Username, // optional
    wchar_t                 *Password, // optional
    char                    *TargetUrl, // optional
    PT_UUID                 MachineUUID, // optional
    PTSDK_UNICODE_STRING    *VendorName,
    PTSDK_UNICODE_STRING    *AppName,
    PTSDK_UNICODE_STRING    *EnterpriseName
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Output	An opaque handle to a data structure maintained by the Intel AMT device, identifying the registered application. The handle is used in subsequent API calls. The handle is valid only when the return status is <code>PT_STATUS_SUCCESS</code> . It remains valid until the next call to <code>ISVS_RegisterApplication</code> , <code>ISVS_RegisterApplicationEx</code> , or <code>ISVS_UnregisterApplication</code> .
Username	Input	A pointer to a null terminated string used to authenticate with a remote Intel AMT device, using HTTP Digest Authentication. These credentials will be used to establish authentication in all subsequent calls during an established session. This parameter is not used when communicating with a local Intel AMT device. In this case this parameter should be NULL. The username must meet the criteria defined in the <i>Network Interface Guide</i> .
Password	Input	A pointer to a null terminated string used to authenticate with a remote Intel AMT device, using HTTP Digest Authentication. These credentials will be used to establish authentication in all subsequent calls during an established session. This parameter is not used when communicating with a local Intel AMT device. In this case this parameter should be NULL. The password must meet the criteria defined in the <i>Network Interface Guide</i> .

Parameters	Input/output	Description
TargetUrl	Input	A pointer to a null terminated string that specifies the URL pointing to the Storage service on a remote host. Registration and all subsequent storage calls for the current session will be sent to the specified URL. If TargetURL is NULL, the session will be opened with the local Intel AMT device. When a local agent registers to a platform with Intel AMT Release 2.0 or greater, it must supply a target URL. See ISVS_RegisterApplicationEx .
MachineUUID	Input	The caller specifies either the UUID value returned from ISVS_GetHostUUID , or sets UUID to NULL. When the UUID is NULL, the Intel AMT device will use the host UUID. See ISVS_GetHostUUID for more information.
Vendor Name	Input	See PTSDK_APPLICATION_ATTRIBUTES
AppName	Input	See PTSDK_APPLICATION_ATTRIBUTES
EnterpriseName	Input	A pointer to a null terminated Unicode string specifying a pre-registered enterprise name.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_UNSUPPORTED_ISVS_VERSION	The Intel AMT device does not support the ISVS version requested by the library.
PT_STATUS_INVALID_REGISTRATION_DATA	This error occurs when an invalid name or a non pre-registered “Enterprise” name was specified, or when the current registration was attempted on an interface from the different from the one used on initial registration.
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.
PT_STATUS_MAX_LIMIT_REACHED	Maximum limit on number of simultaneously registered applications has been reached.
PTSDK_STATUS_NETWORK_ERROR	A network error has occurred while processing the call. This might happen if Username, Password or TargetUrl are incorrect. For other possible reasons see shared error code table.
PTSDK_STATUS_URL_REQUIRED	URL was not supplied when trying to access machine running Intel AMT Release 2.0 or greater.

4.6.1.7 ISVS_RegisterApplicationEx

`ISVS_RegisterApplicationEx` is the extended version of `ISVS_RegisterApplication` designed to accommodate providing the client certificate used for Mutual Authentication when enabled. For more information on mutual authentication, refer to the *Network Interface Guide* document within the SDK.

An ISV application invokes `ISVS_RegisterApplicationEx` to register with Intel AMT. The ISV application must register with Intel AMT each time it initializes the storage library. Following a successful completion, Intel AMT returns a Session handle to the application. The application

specifies this handle in subsequent ISV storage calls until it sends a new registration request (with identical information).

Once an application has initially registered with unique registration information ("Vendor Name", "Application Name", Enterprise Name" and UUID) from either the local or the remote interface, Intel AMT will reject registration attempts with identical registration data from a different interface. This prevents masquerading of local agents from the network and vice versa.

Function Header

```
PT_STATUS ISVS_RegisterApplicationEx (
    SESSION_HANDLE          *SessionHandle,
    wchar_t                 *Username,
    wchar_t                 *Password,
    char                    *TargetUrl,
    PT_UUID                 MachineUUID, // optional
    PTSDK_UNICODE_STRING    *VendorName,
    PTSDK_UNICODE_STRING    *AppName,
    PTSDK_UNICODE_STRING    *EnterpriseName,
    SESSION_AUTHENTICATION_INFO *AuthInfo //
optional
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Output	An opaque handle to a data structure maintained by the Intel AMT device, identifying the registered application. The handle is used in subsequent API calls. The handle is valid only when the return status is <code>PT_STATUS_SUCCESS</code> . It remains valid until the next call to <code>ISVS_RegisterApplication</code> , <code>ISVS_RegisterApplicationEx</code> , or <code>ISVS_UnregisterApplication</code> .
Username	Input	A pointer to a null terminated string used to authenticate with a remote Intel AMT device, using HTTP Digest Authentication. These credentials will be used to establish authentication in all subsequent calls during an established session. This parameter must be specified when communicating with a local Intel AMT Release 2.0 device or later release. This parameter is not used when communicating with a local Intel AMT Release 1.0 device, in this case this parameter should be an empty string. The username must meet the criteria defined in the <i>Network Interface Guide</i> .

Parameters	Input/output	Description
Password	Input	A pointer to a null terminated string used to authenticate with a remote Intel AMT device using HTTP Digest Authentication. These credentials will be used to establish authentication in all subsequent calls during an established session. This parameter must be specified when communicating with a local Intel AMT Release 2.0 device or later release. This parameter is not used when communicating with a local Intel AMT Release 1.0 device. In this case this parameter should be an empty string. The password must meet the criteria defined in the <i>Network Interface Guide</i> .
TargetUrl	Input	A pointer to a null terminated string that specifies the URL pointing to the Storage service on a remote host. Registration and all subsequent storage calls for the current session will be sent to the specified URL. This parameter must not be null. To specify access to the local host when Intel AMT is not configured for TLS , use the local host machine name or "localhost" or 127.0.0.1 as the machine address (e.g. http://localhost:16992/StorageService). When Intel AMT is configured for TLS , the host name in the URL must match the host name signed in the Intel AMT certificate.
MachineUUID	Input	The caller specifies either the UUID value returned from ISVS_GetHostUUID , or sets UUID to NULL. When the UUID is NULL, the Intel AMT device will use the host UUID. See ISVS_GetHostUUID for more information.
Vendor Name	Input	See PTSDK_APPLICATION_ATTRIBUTES
AppName	Input	See PTSDK_APPLICATION_ATTRIBUTES
EnterpriseName	Input	A pointer to a null terminated Unicode string specifying a pre-registered enterprise name.
AuthInfo	Input	Required if mutual authentication or Kerberos is enabled on the machine specified by TargetURL, otherwise it may be NULL. See ISVS_CreateAuthInfo .

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_UNSUPPORTED_ISVS_VERSION	The Intel AMT device does not support the ISVS version requested by the library.
PT_STATUS_INVALID_REGISTRATION_DATA	This error occurs when an invalid name or a non pre-registered "Enterprise" name was specified, or when the current registration was attempted on an interface from the different from the one used on initial registration.
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.

Status	Description
PT_STATUS_MAX_LIMIT_REACHED	Maximum limit on number of simultaneously registered applications has been reached.
PTSDK_STATUS_NETWORK_ERROR	A network error has occurred while processing the call. This might happen if Username, Password or TargetUrl are incorrect. For other possible reasons see shared error code table.
PTSDK_STATUS_URL_REQUIRED	URL was not supplied when trying to access an Intel AMT Release 2.0 machine or later release.

4.6.1.8 ISVS_UnregisterApplication

A registered ISV application sends an `ISVS_UnregisterApplication` request to unregister itself with Intel AMT. This command removes the application from the Application Session List (ASL), but leaves it in the Application Registration List (ARL) and leaves its data intact. Any block locks that the application acquired will be released. Following a completion of this call, the `SessionHandle` is invalid and should not be used in subsequent ISVS operations.

Intel AMT will immediately remove a session once the `ISVS_UnregisterApplication` call is made. You should distinguish between an application that is registered and an application that has an open session:

- Once an application has registered with Intel AMT, it will remain registered until it is removed by the [ISVS_RemoveApplication](#) command (or the equivalent Storage Administration network command).
- A session will expire after the session timeout is reached or once the application calls [ISVS_UnregisterApplication](#) command.

An application that is registered but does not have a current active session can be added to permissions groups. It will be listed by [ISVS_GetRegisteredApplications](#).

When a registered application initiates a new session, it will get the existing application handle for the registered application and will therefore be able to participate in permissions groups and other functions.

Function Header

```
PT_STATUS ISVS_UnregisterApplication (
    SESSION_HANDLE SessionHandle
);
```

Function Parameters

Parameters	Input/output	Description
<code>SessionHandle</code>	Input	The session handle of the application that is sending the message.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request succeeded.

4.6.1.9 ISVS_RemoveApplication

A registered ISV application sends an `ISVS_RemoveApplication` request to release all allocated Intel AMT storage resources associated with that ISV application. If the application owns any storage blocks, they are released, along with any associated permissions group resources. If the application is in the Partner Allocation Control List (PACL), the PACL entry will not be removed. The application is removed from all permissions groups associated with storage blocks that are owned by another application. Following a successful completion of this call, the `SessionHandle` input is invalid and should not be used in subsequent ISVS operations.

Function Header

```
PT_STATUS ISVS_RemoveApplication (
    SESSION_HANDLE SessionHandle
);
```

Function Parameters

Parameters	Input/output	Description
<code>SessionHandle</code>	Input	The session handle of the application that is sending the message.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
<code>PT_STATUS_SUCCESS</code>	Request completed successfully
<code>FLASH_WRITE_LIMIT_EXCEEDED</code>	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.

4.6.1.10 ISVS_InitializeCOMinThread

Note: Relevant to Windows operating systems only*

This function must be called by every thread, including the main thread, which uses the library, before any other ISVS call in that thread (except `ISVS_Initialize`).

Recommended use of initialization functions (in Windows):

<code>ISVS_Initialize(...)</code>	-Once per application
<code>ISVS_InitializeCOMinThread()</code>	-For each thread using the library
Any ISVS library calls	
<code>ISVS_UninitializeCOMinThread()</code>	-For each thread using the library
<code>ISVS_Uninitialize()</code>	-Once per application

Function Header

```
PT_STATUS PT_STATUS ISVS_InitializeCOMinThread (
);
```

Function Parameters

None.

Function Return Status

See [Shared Error Codes](#) table for additional values.

Status	Description
PT_STATUS_SUCCESS	Request completed successfully

4.6.1.11 ISVS_UninitializeCOMinThread

Note: Relevant to Windows operating systems only*

This function must be called at the end of execution of every thread that called `ISVS_InitializeCOMinThread`. See [ISVS_InitializeCOMinThread](#) description for more details.

Function Header

```
PT_STATUS PT_STATUS ISVS_UninitializeCOMinThread (
    );
```

Function Parameters

None.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully

4.6.2 Storage Configuration

The storage configuration functions are used to allocate and deallocate blocks of non-volatile memory and to control visibility of the blocks. The permission functions, described later, control the ability of other applications to read from and/or write to the blocks.

While none of these functions are used in isolation, this section is intended to provide clarity on the necessary programmatic flow for allocating and configuring non-volatile storage blocks. The flowchart of the process is shown in

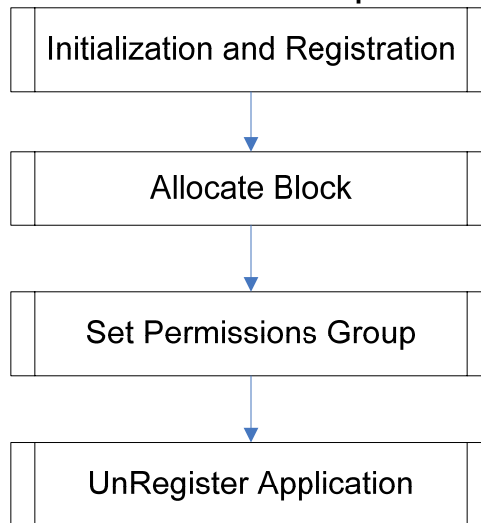


Figure 1.

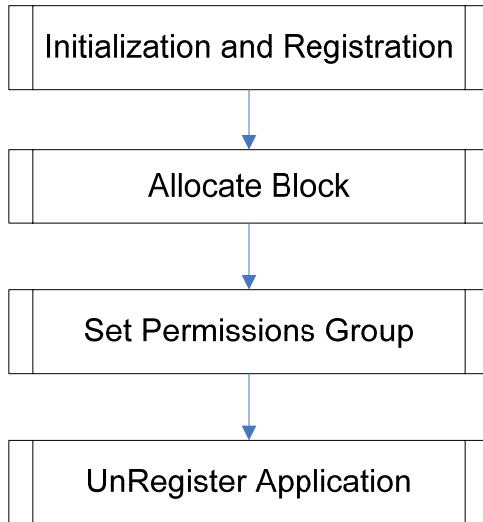


Figure 1: The allocation and configuration flowchart

This section describes the “Allocate Block” steps.

While there are discrete commands that can be used to allocate and configure storage blocks, it is recommended to use a single API call to achieve this block allocation operation. Figure 2 shows the steps involved to perform the block allocation.

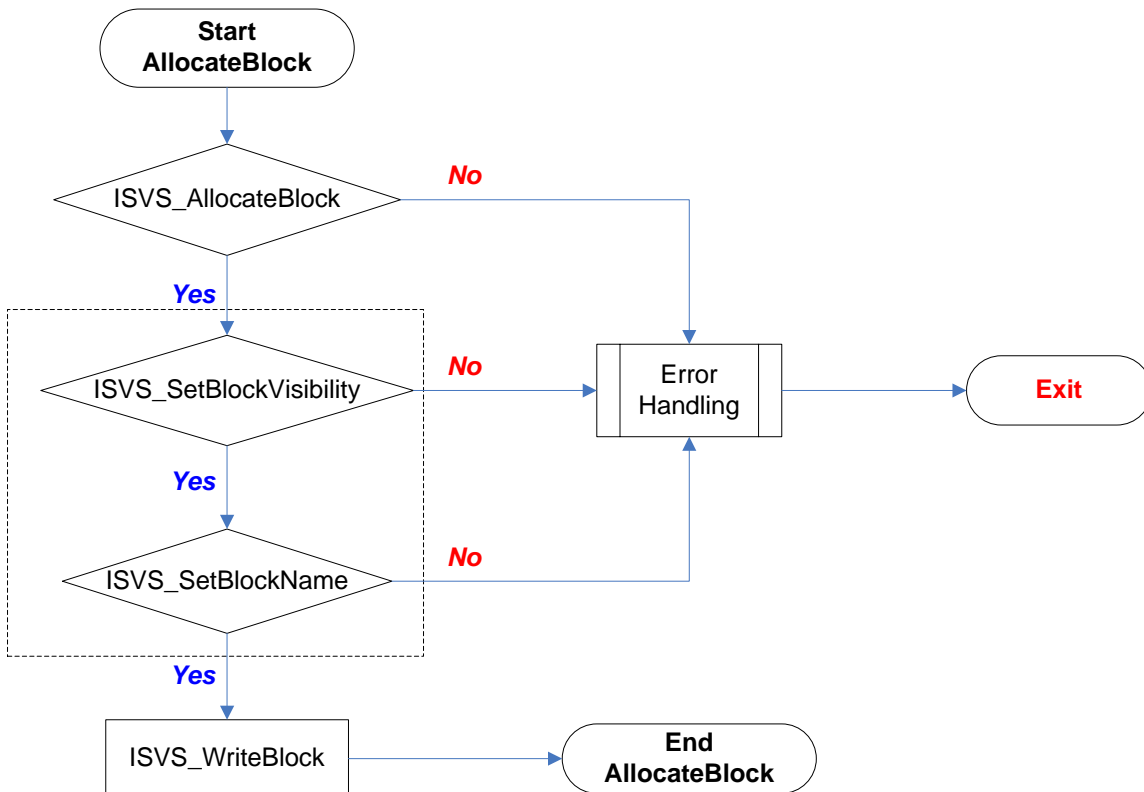


Figure 2: The block allocation flow, including the discrete subordinate API calls

The final step in the configuration process is to update the permission groups. This step is necessary because the applications registered with Intel AMT may have changed since the last initialization. The details of the permission process are described in the [Permission API](#) section.

4.6.2.1 ISVS_GetRegisteredApplications

ISVS_GetRegisteredApplications function is sent by an ISV application to get a list of handles for ISV applications that at some point have registered with Intel AMT. The function will return only those handles of applications that have the same VendorName as the current application. The handles can be used in subsequent storage calls to access storage objects owned by one of these applications.

Function Header

```
PT_STATUS ISVS_GetRegisteredApplications (
    SESSION_HANDLE          SessionHandle,
    uint32                  StartIndex,
    uint32                  *TotalRegisteredAppCount,
    uint32                  *AppHandleCount,
    ISVS_APPLICATION_HANDLE AppHandles[]
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
StartIndex	Input	A handle used by the API library in subsequent enumeration in the case that the buffer provided by the caller is not sufficiently large to fit all the registered applications handles, as indicated by TotalRegisteredAppCount. On first call, this value should be set to 1. To continue enumeration the caller should set: StartIndex += AppHandleCount
TotalRegisteredAppCount	Output	A pointer to the number of applications that at some time have registered with Intel AMT. The value in this field is always greater than or equal to the output value in the AppHandleCount field.
AppHandleCount	Input/Output	Input: A pointer to the number of handles that can fit in the provided AppHandles array. Output: A pointer to the number of application handles actually placed in the Handles array. The output value is always smaller than or equal to the input value.
AppHandles	Output	An array of application handles. AppHandleCount specifies the number of handles in this array. The calling function must allocate enough space to accommodate the number of application handles stated in the input value of AppHandleCount.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_INVALID_INDEX	StartIndex is invalid.

4.6.2.2 ISVS_GetCurrentApplicationHandle

ISVS_GetCurrentApplicationHandle is sent by a registered and authenticated ISV application to retrieve its application handle from the storage library. This handle can be used in subsequent query operations (ISVS_GetApplicationAttributes, ISVS_GetAllocatedBlocks), indicating that the requested information is associated with the current application. The library maintains the current application handle internally within the SessionHandle data structure.

Since this function does not query the Intel AMT device, it does not update the registration timeout counter in the device.

If the application registration has timed out, the function will return an error.

Function Header

```
PT_STATUS ISVS_GetCurrentApplicationHandle (
    SESSION_HANDLE          SessionHandle,
    ISVS_APPLICATION_HANDLE *CurrentAppHandle
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
CurrentAppHandle	Output	The current application handle, maintained in the SessionHandle data structure

Function Return Status

Status	Description
PT_STATUS_SUCCESS	Request completed successfully.
PTSDK_STATUS_REQUESTOR_NOT_REGISTERED	The application that sent the request message is not registered.
PTSDK_STATUS_NOT_INITIALIZED	An ISV operation was called while the library is not initialized
PTSDK_STATUS_INVALID_PARAM	One of the input parameters is invalid (usually indicates a NULL pointer or an invalid session handle is specified)

4.6.2.3 ISVS_GetApplicationAttributes

ISVS_GetApplicationAttributes is sent by a registered ISV application to get a listing of attributes for any single ISV application that at some point registered with Intel AMT.

Function Header

```
PT_STATUS ISVS_GetApplicationAttributes (
    SESSION_HANDLE          SessionHandle,
    ISVS_APPLICATION_HANDLE ApplicationBeingRequested,
    PTSDK_APPLICATION_ATTRIBUTES *ApplicationAttributes
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
ApplicationBeingRequested	Input	The application handle whose attributes are being requested
ApplicationAttributes	Input/Output	Input: A pointer to a PTSDK_APPLICATION_ATTRIBUTES structure. The members should indicate sufficient storage to hold the applications' Vendor, Name and Enterprise properties. Output: The attributes of the application specified in ApplicationBeingRequested.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_APPLICATION_DOES_NOT_EXIST	The specified queried application is not registered or contains a different Vendor Name attribute then calling application.
PTSDK_STATUS_PARAM_BUFFER_TOO_SHORT	Specified container can not hold the requested string

4.6.2.4 ISVS_AllocateBlock

ISVS_AllocateBlock is sent by a registered ISV application to allocate a portion of the Intel AMT non-volatile storage area, sets the block visibility and defines the block name. Once the block is allocated the application becomes the owner of the block.

Function Header

```
PT_STATUS ISVS_AllocateBlock (
    SESSION_HANDLE          SessionHandle,
    uint32                 BytesRequested,
    PT_BOOLEAN              BlockHidden,
    PTSDK_UNICODE_STRING   *BlockName,
    ISVS_BLOCK_HANDLE      *BlockHandle
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BytesRequested	Input	The number of bytes requested by the sender must be a multiple of the physical flash block size, 4KB.
BlockHidden	Input	The value should be TRUE if the block should be hidden from other applications, FALSE otherwise. See ISVS_SetBlockVisibility for additional information
BlockName	Input	An optional name that the application can assign to this block. If a null value or an empty string is provided, then no name is assigned to the allocated block.
BlockHandle	Output	A pointer to a block handle generated by Intel AMT on successful return. The handle can be used to reference the block in subsequent operations.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_NOT_ENOUGH_STORAGE	Could not allocate requested number of bytes
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.
PT_STATUS_INVALID_BYTE_COUNT	Invalid block size specified, e.g. not a multiple of 4 KB.

4.6.2.5 ISVS_DeallocateBlock

ISVS_DeallocateBlock enables an application that owns an allocated block to return it back to the system. This command will also erase all physical flash blocks that are covered by the allocation.

Function Header

```
PT_STATUS ISVS_DeallocateBlock (
    SESSION_HANDLE    SessionHandle
    ISVS_BLOCK_HANDLE BlockHandle,
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BlockHandle	Input	Identifies the block to be de-allocated

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden
PT_STATUS_NOT_OWNER	The operation is only permitted to the block's owner. The requesting application does not own the block.

4.6.2.6 ISVS_SetBlockVisibility

ISVS_SetBlockVisibility enables an application that owns an allocated memory block to set the block visibility attribute. When a memory block is hidden it will not be listed when other applications call ISVS_GetAllocatedBlocks with the handle of the block owner. Applications which are members of a permission-group associated with a “hidden block” are still granted the access permission to the block (i.e. read the data in the block). In addition to setting the block visibility, the owner must not assign I/O access permissions to other applications if it wishes to completely deny access to the block data *and* attributes.

Function Header

```
PT_STATUS ISVS_SetBlockVisibility (
    SESSION_HANDLE      SessionHandle,
    ISVS_BLOCK_HANDLE   BlockHandle,
    PT_BOOLEAN          BlockHidden
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BlockHandle	Input	Identifies the block being modified
BlockHidden	Input	If TRUE, block will be hidden from other applications; if FALSE, block will be visible to other applications.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.
PT_STATUS_NOT_OWNER	The operation is only permitted to the block's owner. The requesting application does not own the block.

4.6.2.7 ISVS_SetBlockName

ISVS_SetBlockName enables an application that owns an allocated block to modify the block's name.

Function Header

```
PT_STATUS ISVS_SetBlockName (
    SESSION_HANDLE      SessionHandle,
    ISVS_BLOCK_HANDLE   BlockHandle,
    PTSDK_UNICODE_STRING *BlockName
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BlockHandle	Input	Identifies the allocated block of ISV storage being named
BlockName	Input	The name being assigned to the storage block.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.
PT_STATUS_NOT_OWNER	The operation is only permitted to the block's owner. The requesting application does not own the block.

4.6.3 Data Storage Functions

The data storage functions not only perform reads and writes, but also can lock and unlock blocks so that one application will not interfere with the activities of another application. Figure 3 demonstrates the expected flow for accessing the non-volatile storage.

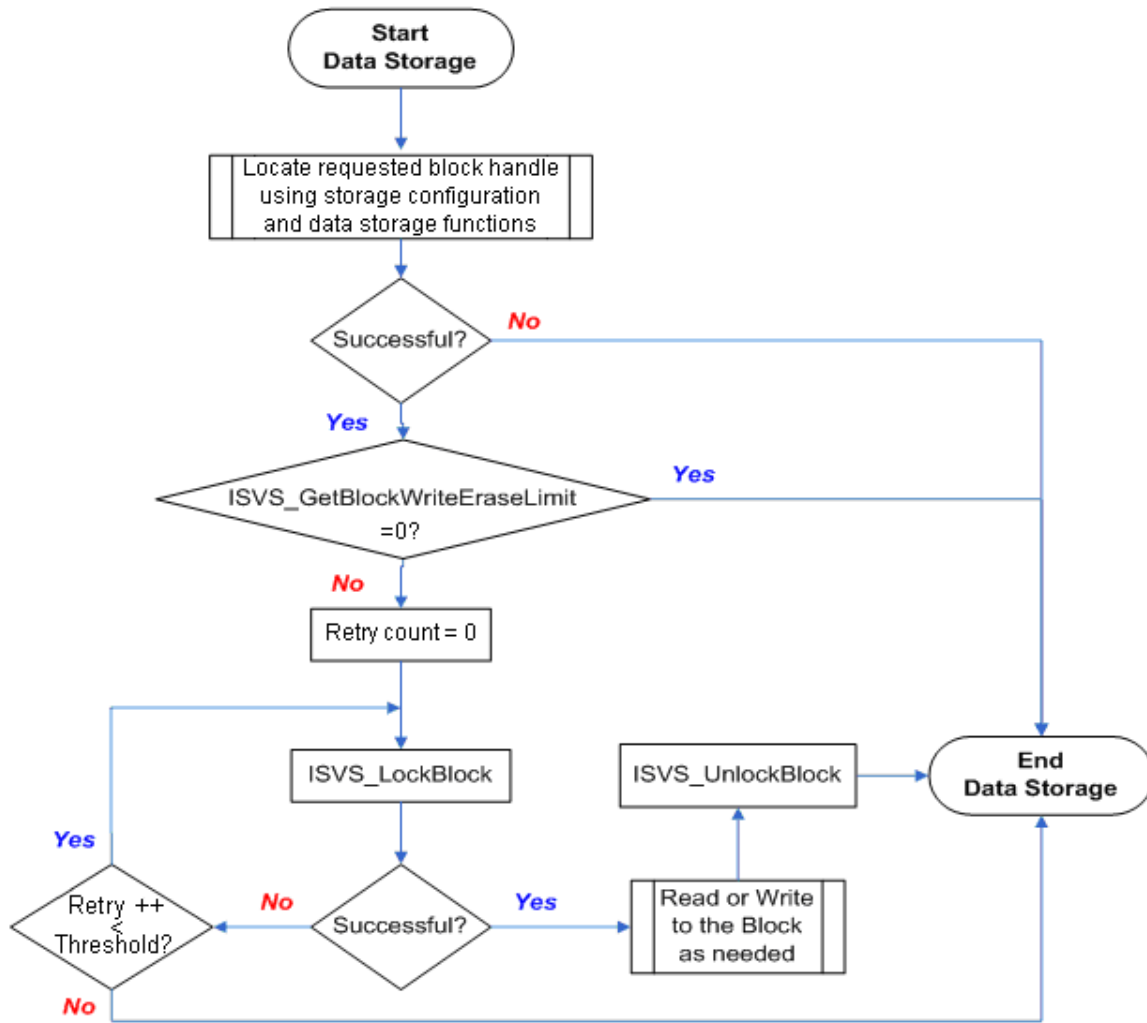


Figure 3: Flowchart representing the accessing a non-volatile storage block

The first task in accessing non-volatile storage blocks is to identify those blocks that are accessible by an application.

4.6.3.1 ISVS_GetAllocatedBlocks

ISVS_GetAllocatedBlocks enables an application to get a listing of block handles for those blocks allocated by a given application. Blocks allocated with “Hidden” visibility attribute will not appear in this command unless the command is issued by the block owner, even if the requesting application has been granted I/O access on them by the owner through a permissions group.

Note: There is no connection in-between I/O permission and block enumeration.

Function Header

PT_STATUS ISVS_GetAllocatedBlocks (

```

SESSION_HANDLE          SessionHandle,
ISVS_APPLICATION_HANDLE BlockOwnerApplication,
uint32                  StartIndex,
uint32                  *TotalAllocatedBlockCount,
uint32                  *BlockHandleCount,
ISVS_BLOCK_HANDLE      BlockHandles[]
);

```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
BlockOwnerApplication	Input	The handle of the application whose block list is being retrieved
StartIndex	Input	A handle used by the library in subsequent calls to continue current enumeration in case the buffer provided by the caller is not sufficiently large to fit all the block handles owned by the specified application – on first call this value should be set to 1.
TotalAllocatedBlockCount	Output	A pointer to the number of blocks currently owned by the specified application; the value in this field is always greater than or equal to the output value in the BlockHandleCount field – valid only on return values: PT_STATUS_SUCCESS
BlockHandleCount	Input/Output	Input: a pointer to the number of block handles allocated in the BlockHandles array Output: a pointer to the number of block handles actually placed in the BlockHandles array; the output value is always smaller than or equal to the input value
BlockHandles	Output	An array of block handles – BlockHandleCount specifies the number of handles in this array. The calling function must allocate enough space to accommodate the number of application handles stated in the input value of BlockHandleCount.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_APPLICATION_DOES_NOT_EXIST	The Application handle specified is invalid.
PT_STATUS_INVALID_INDEX	StartIndex is invalid

4.6.3.2 ISVS_GetBlockAttributes

ISVS_GetBlockAttributes enables an application to get a listing of attributes for a single allocated block. Blocks allocated with the "Hidden" visibility attribute will not appear in this command unless the command is issued by the block owner.

Note: An owner may hide a block at any time, so a block that was previously listed as visible may no longer be visible when [ISVS_GetBlockAttributes](#) is called.

Function Header

```
PT_STATUS ISVS_GetBlockAttributes (
    SESSION_HANDLE           SessionHandle,
    ISVS_BLOCK_HANDLE       BlockHandle,
    PTSDK_BLOCK_ATTRIBUTES *BlockAttributes
);
```

Function Parameters		
Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
BlockHandle	Input	Identifies the block whose attributes are being requested
BlockAttributes	Output	A pointer to the attributes of the block specified in the corresponding request message – valid only on return status: PT_STATUS_SUCCESS

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PTSDK_STATUS_PARAM_BUFFER_TOO_SHORT	Specified container can not hold the requested string.
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden

4.6.3.3 ISVS_LockBlock

The ISVS_LockBlock request message is sent by a registered ISV application to lock a block of non-volatile memory, thereby preventing other applications from reading or writing the block's data. If the request succeeds, the requesting application will own the block's lock until it explicitly releases it with an ISVS_UnlockBlock request message, or until an inactivity timer expires, whichever occurs first. Also, if the application session terminates, the lock is also removed. The lock does not prevent the block from being de-allocated. If the owner of a block de-allocates it while another application holds the lock, the lock is destroyed with the block. Any subsequent requests that reference the de-allocated block will fail with the status code PT_STATUS_BLOCK_DOES_NOT_EXIST. The caller can query the timeout period by calling [ISVS_GetTimeoutValues](#). As a lock operation following a lock operation performed on the same block, executed by the same application would succeed, this operation can not be used to synchronize operation between multiple application threads.

Function Header

```
PT_STATUS ISVS_LockBlock (
    SESSION_HANDLE      SessionHandle,
    ISVS_BLOCK_HANDLE   BlockHandle
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BlockHandle	Input	Identifies the allocated block of ISV storage being locked

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully.
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified
PT_STATUS_BLOCK_LOCKED_BY_OTHER	Could not access the block for I/O operations; the block is currently locked by another application.
PT_STATUS_NOT_PERMITTED	Requesting application is not permitted to perform the specified I/O operation on the block; the application must have at least read permission to the block.

4.6.3.4 ISVS_UnlockBlock

ISVS_UnlockBlock enables an application to remove a lock from a block of non-volatile memory so that other applications may access the allocated block for read or write operations.

Function Header

```
PT_STATUS ISVS_UnlockBlock (
    SESSION_HANDLE      SessionHandle,
    ISVS_BLOCK_HANDLE   BlockHandle
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BlockHandle	Input	Identifies the allocated block of ISV storage being unlocked

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified

Status	Description
PT_STATUS_BLOCK_LOCKED_BY_OTHER	Could not access the block for I/O operations; the block is currently locked by another application.
PT_STATUS_NOT_PERMITTED	Requesting application is not permitted to perform the specified I/O operation on the block; the application must have read permission to the block.
PT_STATUS_BLOCK_NOT_LOCKED	Specified block is not locked

4.6.3.5 ISVS_ReadBlock

ISVS_ReadBlock is sent by a registered ISV application to retrieve data previously stored in a block of local nonvolatile memory. The data is transparent to Intel AMT and is treated as an unformatted sequence of bytes. The requesting application must have read permissions on the block.

Note: If the requested data size is larger than the underlying transport layer MTU, the Library will fragment the caller's Read request into multiple Read requests. In case one of the fragmented requests initiated by the library fails, the library will cease the Read operation, specify the number of bytes actually read (0 to ByteCount -1) and return the appropriate error status code.

Function Header

```
PT_STATUS ISVS_ReadBlock (
    SESSION_HANDLE      SessionHandle,
    ISVS_BLOCK_HANDLE  BlockHandle,
    uint32              ByteOffset,
    uint32              ByteCount,
    uint32              *BytesRead,
    uint8              Data[]
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BlockHandle	Input	Identifies the block from which data will be retrieved
ByteOffset	Input	The byte offset – relative to the first byte of the block – from which the first byte of Data is to be retrieved.
BytesCount	Input	The number of bytes to read. Data[] buffer should be sufficiently large to hold these number of bytes.
BytesRead	Output	The number of bytes actually read. ByteOffset, being the first, should always be smaller than or equal to the BytesCount value. Note: This value is valid on any return status; it may indicate that partial data was read if the return status is not PT_STATUS_SUCCESS.
Data	Output	A buffer containing read data.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request succeeded. The caller must check the BytesRead value to check how many bytes were actually written to the block.
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden.
PT_STATUS_INVALID_BYTE_OFFSET	Invalid block offset was specified; the value must be less than the block's size.
PT_STATUS_INVALID_BYTE_COUNT	Invalid byte count was specified
PT_STATUS_BLOCK_LOCKED_BY_OTHER	Could not access the block for I/O operations; the block is currently locked by another application.
PT_STATUS_NOT_PERMITTED	The requesting application is not permitted to perform the specified I/O operation on the block. The application must have at least read permission to the block.

4.6.3.6 ISVS_WriteBlock

ISVS_WriteBlock enables an application with proper permissions to write data to an allocated block. The format of the data is unknown to the API and is stored without modification.

Note: If the requested data size is larger than the underlying transport layer MTU, the Library will fragment the caller's Write request into multiple Write requests. In the case that one fragmented request is initiated by the library, the library will cease the Write operation, specify the number of bytes actually written (0 to ByteCount -1) and return the appropriate error status code.

Function Header

```
PT_STATUS ISVS_WriteBlock (
    SESSION_HANDLE      SessionHandle,
    ISVS_BLOCK_HANDLE  BlockHandle,
    uint32              ByteOffset,
    uint32              ByteCount,
    uint32              *BytesWritten,
    uint8               Data[]
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BlockHandle	Input	Identifies the block in which Data[] will be stored
ByteOffset	Input	The byte offset – relative to the first byte of the block - at which the first byte of Data[] is to be stored.
ByteCount	Input	The number of bytes in Data[]

Parameters	Input/output	Description
BytesWritten	Output	A pointer to the number of bytes, starting from Data[0] that were actually stored by this operation. Note: This value is valid on any return status, it may indicate that partial data was modified although return value is not PT_STATUS_SUCCESS.
Data	Input	The data to be stored.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request succeeded
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified
PT_STATUS_INVALID_BYTE_OFFSET	Invalid block offset was specified. The value must be less the block's size.
PT_STATUS_INVALID_BYTE_COUNT	Invalid byte count was specified.
PT_STATUS_BLOCK_LOCKED_BY_OTHER	Could not access the block for I/O operations. The block is currently locked by another application.
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanisms prevented a write to an NVRAM sector.
PT_STATUS_NOT_PERMITTED	Requesting application is not permitted to perform the specified I/O operation on the block. The application must have at least write permission to the block.

4.6.3.7 ISVS_GetBlockWriteEraseLimit

ISVS_GetBlockWriteEraseLimit returns the number of currently available write operations on the specified block. If the block is accessible to other applications the application should lock the block before querying this value. The limit is a temporal limit; the Intel AMT device maintains a counter per sector that is decremented on each erase operation and incremented every 40 minutes.

Function Header

```
PT_STATUS ISVS_GetBlockWriteEraseLimit (
    SESSION_HANDLE    SessionHandle,
    ISVS_BLOCK_HANDLE BlockHandle,
    uint32            *WriteEraseLimit,
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BlockHandle	Input	Identifies the block in which Data[] will be stored

Parameters	Input/output	Description
WriteEraseLimit	Output	On successful return, this value indicates the number of write operations, per 4KB sector, that are guaranteed to succeed before the Flash wear-out protection mechanism will disable subsequent write operations to the given block.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request succeeded
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified
PT_STATUS_NOT_PERMITTED	Requesting application is not permitted to perform the specified I/O operation on the block. The application must have at least read permission to the block.

4.6.4 Permissions

The permissions functions allow an ISV application to configure access rights of other applications to non-volatile storage allocated to the application. Several of these commands have already been addressed in the task-based descriptions above and will not be repeated in this section.

An application may use permission groups to allow read and/or write access to its blocks from other registered applications. Each permission group contains:

- A name
- A permission type (read or read/write), and
- A list of application handles and/or permissions group filters which are granted this permission type.

A block may have more than one group assigned to it, but a group is assigned to only one block.

The flow chart in Figure 4 shows the steps involved in setting the permissions for other ISV applications.

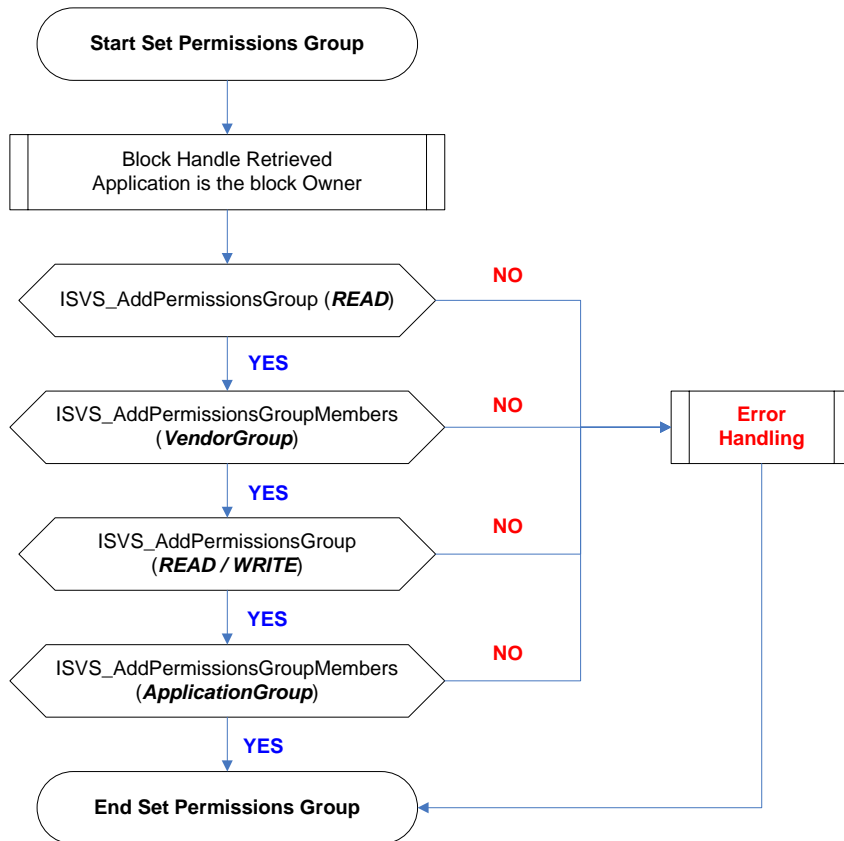


Figure 4: Sequence for Setting Permissions

A Block Permission Group consists of a list of registered application handles and the permission these applications have to access the block that has the group assigned to it. The following constants are special handles that attach a collection of applications to a group when the special handle is added to the group.

```
#define ISVS_APPLICATION_NAME_FILTER    0xFFFFFFFF0
```

A filter handle describing all registered applications which share the same Vendor Name, Application Name and Enterprise name as the application that owns the block.

```
#define ISVS_VENDOR_NAME_FILTER        0xFFFFFFFF1
```

A filter handle describing all registered applications which have the same Vendor Name and Enterprise name as the application that owns the block.

4.6.4.1 ISVS_GetPermissionsGroups

The `ISVS_GetPermissionsGroups` call is sent by a registered and authenticated ISV application to get a list of handles for the permissions groups currently defined for a block of ISV storage. Only the block owner may request this operation for a hidden block. Any registered application may request this operation for a visible block.

Function Header

```
PT_STATUS ISVS_GetPermissionsGroups (
    SESSION_HANDLE      SessionHandle,
    ISVS_BLOCK_HANDLE   BlockHandle,
```

```

uint32          StartIndex,
uint32          *TotalGroupCount,
uint32          *GroupHandleCount,
ISVS_GROUP_HANDLE GroupHandles[ ]
);

```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
BlockHandle	Input	Identifies the storage block being modified
StartIndex	Input	A handle used by the static library in subsequent calls to continue current enumeration – in case the buffer provided by the caller is not sufficiently large to fit all the block's permissions group handles. On first call this value should be set to 1.
TotalGroupCount	Output	A pointer to the number of permission groups set on the specified block; the value in this field is always greater than or equal to the output value in the GroupHandleCount field.
GroupHandleCount	Input/Output	Input: a pointer to the number of permission group handles allocated in the GroupHandles array. Output: a pointer to the number of permission group handles actually placed in the GroupHandles array. The output value is always smaller than or equal to the input value.
GroupHandles	Output	An array of group handles – GroupHandleCount specifies the number of handles in this array. The calling function must allocate enough space to accommodate the number of group handles stated in the input value of GroupHandleCount.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_INVALID_INDEX	StartIndex is invalid
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified, or block is hidden

4.6.4.2 ISVS_GetPermissionsGroupAttributes

ISVS_GetPermissionsGroupAttributes is sent by a registered ISV application to get the attributes for a single permissions group. Only the block owner may request this operation for a group that is associated with a hidden block. Any registered application may request this operation for a group that is associated with a visible block.

Function Header

```
PT_STATUS ISVS_GetPermissionsGroupAttributes (
```

```

SESSION_HANDLE          SessionHandle,
ISVS_BLOCK_HANDLE      BlockHandle,
ISVS_GROUP_HANDLE      GroupHandle,
PTSDK_PERMISSIONS_GROUP_ATTRIBUTES *GroupAttributes
);

```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
BlockHandle	Input	Identifies the storage block that owns the group specified by GroupHandle
GroupHandle	Input	Identifies the permissions group whose attributes are being requested
GroupAttributes	Output	The attributes of the group specified by GroupHandle

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden
PTSDK_STATUS_PARAM_BUFFER_TOO_SHORT	Specified container can not hold the requested string
PT_STATUS_GROUP_DOES_NOT_EXIST	Invalid permission group handle specified

4.6.4.3 ISVS_AddPermissionsGroup

ISVS_AddPermissionsGroup is sent by a registered ISV application to add a permissions group to a block of ISV storage, set its initial permissions value, and define its name. Each permissions group has its own set of permissions. Each member of a permission group is an application with a unique Application Handle. A permissions group can be used to establish different storage roles for different applications from the same ISV. Each permissions group is associated with exactly one block. The sending application must be the owner of the block.

Function Header

```

PT_STATUS ISVS_AddPermissionsGroup (
    SESSION_HANDLE          SessionHandle,
    ISVS_BLOCK_HANDLE      BlockHandle,
    ISVS_GROUP_PERMISSIONS GroupPermissions,
    PTSDK_UNICODE_STRING   *GroupName,
    ISVS_GROUP_HANDLE      *GroupHandle
);

```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message

Parameters	Input/output	Description
BlockHandle	Input	Identifies the storage block being modified
GroupPermissions	Input	The permission flags for this group, which only apply to this storage block
GroupName	Input	An optional name that the sender can assign to this group – a null string indicates no name is associated with this group
GroupHandle	Output	A group handle generated by Intel AMT – used to reference the group in other Intel AMT non-volatile storage operations. Valid only if return value is PT_STATUS_SUCCESS

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully.
PTSDK_STATUS_INVALID_PARAM	One or more of parameters are invalid (Null pointer, invalid string length, invalid session handle).
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified, or block is hidden.
PT_STATUS_NOT_OWNER	The operation is only permitted to the block's owner. The requesting application does not own the block.
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear out protection mechanism prevented a write to an NVRAM sector.
PT_STATUS_MAX_LIMIT_REACHED	The maximum number of permissions groups has already been created for the specified block.
PT_STATUS_INVALID_GROUP_PERMISSIONS	The specified Group Permissions value is invalid.

4.6.4.4 ISVS_RemovePermissionsGroup

ISVS_RemovePermissionsGroup is sent by a registered ISV application to remove a permissions group from a storage block. The sending application must be the owner of the block.

Function Header

```
PT_STATUS ISVS_RemovePermissionsGroup (
    SESSION_HANDLE      SessionHandle,
    ISVS_BLOCK_HANDLE   BlockHandle,
    ISVS_GROUP_HANDLE   GroupHandle,
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle, of the application that is sending the message.

Parameters	Input/output	Description
BlockHandle	Input	Identifies the storage block that owns the group specified by GroupHandle.
GroupHandle	Input	Identifies the permissions group being removed.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully.
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified, or block is hidden.
PT_STATUS_GROUP_DOES_NOT_EXIST	Invalid permission group handle specified
PT_STATUS_NOT_OWNER	The operation is only permitted to the block's owner. The requesting application does not own the block.
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.

4.6.4.5 ISVS_AddPermissionsGroupMembers

ISVS_AddPermissionsGroupMembers is sent by a registered and authenticated ISV application to add member applications or generic application filters to a permissions group associated with a single block of ISV storage. The sending application must be the owner of the block.

Note: The maximum number of members in a permissions group must not exceed eight.

Function Header

```
PT_STATUS ISVS_AddPermissionsGroupMembers (
    SESSION_HANDLE           SessionHandle,
    ISVS_BLOCK_HANDLE       BlockHandle,
    ISVS_GROUP_HANDLE       GroupHandle,
    uint32                   MemberHandleCount,
    ISVS_APPLICATION_HANDLE MemberHandles[]
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
BlockHandle	Input	Identifies the block of storage being modified
GroupHandle	Input	Identifies the permissions group being modified
MemberHandleCount	Input	A pointer to the number of application handles provided in the Handles array
MemberHandles	Input	An array of application handles or generic application filters. MemberHandleCount specifies the number of handles in this array. The calling function must allocate enough space to accommodate the number of application handles stated in the input value of MemberHandleCount.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden
PT_STATUS_GROUP_DOES_NOT_EXIST	Invalid permission group handle specified
PT_STATUS_INVALID_MEMBER_COUNT	The number of specified handles is invalid or the limit on the number of members in the group has been reached.
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.
PT_STATUS_APPLICATION_DOES_NOT_EXIST	The provided handle array includes one or more invalid handles. None of the applications in the input list will be added to the group.
PT_STATUS_NOT_OWNER	The operation is only permitted to the block's owner. The requesting application does not own the block.

4.6.4.6 ISVS_RemovePermissionsGroupMembers

ISVS_RemovePermissionsGroupMembers is sent by a registered ISV application to remove member applications or generic application filters from a permissions group associated with a single block of ISV storage. The sending application must be the owner of the block.

Function Header

```
PT_STATUS ISVS_RemovePermissionsGroupMembers (
    SESSION_HANDLE           SessionHandle;
    ISVS_BLOCK_HANDLE       BlockHandle;
    ISVS_GROUP_HANDLE       GroupHandle;
    uint32                   MemberHandleCount;
    ISVS_APPLICATION_HANDLE MemberHandles[];
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
BlockHandle	Input	Identifies the block of storage being modified
GroupHandle	Input	Identifies the permissions group being modified
MemberHandleCount	Input	The number of application handles in the MemberHandles array
MemberHandles	Input	An array of application handles or generic application filters. MemberHandleCount specifies the number of handles in this array. The calling function must allocate enough space to accommodate the number of application handles stated in the input value of MemberHandleCount

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request succeeded, the caller must check the HandleCount value to determine the number of members actually removed from the permission group.
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified, or block is hidden.
PT_STATUS_GROUP_DOES_NOT_EXIST	Invalid permission group handle specified.
PT_STATUS_INVALID_MEMBER_COUNT	The number of specified handles is invalid.
PT_STATUS_APPLICATION_DOES_NOT_EXIST	MemberHandles includes an application handle that has never been allocated by the Intel AMT device. To identify which members were actually deleted the caller should call: ISVS_GetPermissionsGroupMembers This status is also returned when MemberHandles includes an application handle that is not a member of the permissions group.
PT_STATUS_NOT_OWNER	The operation is only permitted to the block's owner. The requesting application does not own the block.
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.

4.6.4.7 ISVS_SetPermissionsGroupName

ISVS_SetPermissionsGroupName is sent by a registered and authenticated ISV application to set the name of a storage block permissions group. The sending application must be the owner of the block.

Function Header

```
PT_STATUS ISVS_SetPermissionsGroupName (
    SESSION_HANDLE      SessionHandle,
    ISVS_BLOCK_HANDLE   BlockHandle,
    ISVS_GROUP_HANDLE   GroupHandle,
    PTSDK_UNICODE_STRING *GroupName
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
BlockHandle	Input	Identifies the block of storage being modified
GroupHandle	Input	Identifies the permissions group whose name is being modified

Parameters	Input/output	Description
GroupName	Input	The name assigned to the permissions group

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden
PT_STATUS_GROUP_DOES_NOT_EXIST	Invalid permission group handle specified
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.
PT_STATUS_NOT_OWNER	The operation is only permitted to the block's owner. The requesting application does not own the block.

4.6.4.8 ISVS_GetPermissionsGroupMembers

ISVS_GetPermissionsGroupMembers is sent by any registered ISV application to get the attributes of members of a storage block permissions group. The members of a permissions group are registered ISV applications or generic application filters. Members of groups associated with hidden blocks will not be returned unless the originator of the call is the block owner.

Function Header

```
PT_STATUS ISVS_GetPermissionsGroupMembers (
    SESSION_HANDLE           SessionHandle,
    ISVS_BLOCK_HANDLE       BlockHandle,
    ISVS_GROUP_HANDLE       GroupHandle,
    uint32                   StartIndex
    uint32                   *TotalMemberCount;
    uint32                   *MemberHandleCount;
    ISVS_APPLICATION_HANDLE MemberHandles[];
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
BlockHandle	Input	Identifies the block of storage being modified
GroupHandle	Input	Identifies the permissions group being modified
StartIndex	Input	A handle used by the static library in subsequent calls to continue current enumeration in case the buffer provided by the caller is not sufficiently large to fit all the member handles. On first call this value should be set to 1.
TotalMemberCount	Output	A pointer to the total number of members in the specified permission group; the value in this field is always greater than or equal to the output value in the MemberHandleCount field. Valid only on return status: PT_STATUS_SUCCESS

Parameters	Input/output	Description
MemberHandleCount	Input/Output	Input: a pointer to the number of member handles allocated in the MemberHandles array Output: a pointer to the number of member handles actually placed in the MemberHandles array; the output value is always smaller than or equal to the input value. Valid only on return status: PT_STATUS_SUCCESS
MemberHandles	Output	An array of member handles – MemberHandleCount specifies the number of handles in this array; the calling function must allocate enough space to accommodate the number of application handles stated in the input value of MemberHandleCount.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden
PT_STATUS_INVALID_INDEX	StartIndex is invalid
PT_STATUS_GROUP_DOES_NOT_EXIST	Invalid permission group handle specified

4.6.4.9 ISVS_SetPermissionsGroupPermissions

ISVS_SetPermissionsGroupPermissions is sent by a registered ISV application to set the permission flags for a storage block permissions group. The sending application must be the owner of the block.

Function Header

```
PT_STATUS ISVS_SetPermissionsGroupPermissions (
    SESSION_HANDLE          SessionHandle,
    ISVS_BLOCK_HANDLE       BlockHandle,
    ISVS_GROUP_HANDLE       GroupHandle,
    ISVS_GROUP_PERMISSIONS Permissions
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
BlockHandle	Input	Identifies the block of storage being modified
GroupHandle	Input	Identifies the permissions group whose permissions are being modified
Permissions	Input	The permission flag settings to be applied

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully

Status	Description
PT_STATUS_BLOCK_DOES_NOT_EXIST	Invalid storage block handle specified or block is hidden
PT_STATUS_GROUP_DOES_NOT_EXIST	Invalid permission group handle specified
PT_STATUS_INVALID_GROUP_PERMISSIONS	The specified Group Permissions value is invalid
PT_STATUS_FLASH_WRITE_LIMIT_EXCEEDED	The operation failed because the Flash wear-out protection mechanism prevented a write to an NVRAM sector.
PT_STATUS_NOT_OWNER	The operation is only permitted to the block's owner. The requesting application does not own the block.

4.6.5 General Functions

The following functions are not tied to the previous function groups.

4.6.5.1 ISVS_GetTimeoutValues

The `ISVS_GetTimeoutValues` request message is sent by a registered ISV application to get the timeout values used by the Intel AMT device for idle registered applications and idle locked blocks.

Function Header

```
PT_STATUS ISVS_GetTimeoutValues (
    SESSION_HANDLE      SessionHandle,
    uint32              *RegistrationTimeout,
    uint32              *LockTimeout
);
```

Function Parameters

Parameters	Input/Output	Description
SessionHandle	Input	The session handle of the application that is sending the message.
RegistrationTimeout	Output	On successful return specifies time, in seconds, after which an idle registered application is automatically un-registered by the Intel AMT device
LockTimeout	Output	On successful return specifies time, in seconds, after which an idle locked block is automatically unlocked by the Intel AMT device to avoid deadlocks. The following operations reset the expiration timer of the block they operate on: ISVS_ReadBlock , ISVS_WriteBlock , ISVS_LockBlock .

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request succeeded

4.6.5.2 ISVS_GetLastNetworkError

ISVS_GetLastNetworkError should be called by an ISV application in the event that the previous ISV call returned a status of PTSDK_STATUS_NETWORK_ERROR in order to determine the nature of the network failure. Each ISV Session (identified by a session handle) maintains the last network error status it encountered. The library error codes returned are defined by gSOAP. For additional information see:

http://www.cs.fsu.edu/~engelen/soapdoc2.html#tth_sEc10.2

The error codes are defined in the gSOAP package in stdsoap2.h file.

This file is provided as part of the SDK in the <SDK root directory>Windows\Intel AMT SDK\ThirdParty\gSOAP directory for Windows, and in <SDK root directory>Linux\IntelAMTSDK\WSDLFiles\Samples\network_samples.tar.gz for Linux.

When a WinINet error occurs, only the error number is returned to ISVS_GetLastNetworkError. For descriptions of WinINet Error Codes see:

<http://support.microsoft.com/default.aspx?scid=kb;EN-US;193625>

Function Header

```
void ISVS_GetLastNetworkError(
    SESSION_HANDLE SessionHandle
    void *NetworkError
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle for which a previous ISVS call resulted in a PTSDK_NETWORK_ERROR. If the failed ISV call does not utilize a session handle (e.g. ISV_GetAPIVersion, ISVS_RegisterApplicationEx), the caller can query the network error by specifying 0. Caller must guarantee serialization in the later case.
NetworkError	Output	A pointer to implementation specific error status container provided by the caller.

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
N/A	N/A

4.6.5.3 ISVS_GetAPIVersion

*** This function has been deprecated. Use ISVS_GetAPIVersionEx in its place.**

The ISVS_GetAPIVersion call is sent by an application to get the ISVS API version supported by the Intel AMT device. The application does not have to register with the Intel AMT device before calling this API.

Function Header

```
PT_STATUS ISVS_GetAPIVersion (
wchar_t      *Username   OPTIONAL,
wchar_t      *Password   OPTIONAL,
char         *TargetURL  OPTIONAL,
ISVS_Version *Version
);
```

Function Parameters

Parameters	Input/output	Description
Username	Input	A pointer to a null terminated string used to authenticate with a remote Intel AMT device, using HTTP digest Authentication. This parameter is not used when communicating with a local Intel AMT device, in this case this parameter should be NULL. The username must meet the criteria defined in the <i>Network Interface Guide</i> .
Password	Input	A pointer to a null terminated string used to authenticate with a remote Intel AMT device, using Http digest Authentication. This parameter is not used when communicating with a local Intel AMT device, in this case this parameter should be NULL. The password must meet the criteria defined in the <i>Network Interface Guide</i> .
TargetURL	Input	A null-terminated string, pointing to the remote Intel AMT ISV Storage Service URL, if TargetURL is null, the local Intel AMT device is being inquired.
Version	Output	The ISV version supported by the Intel AMT device

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request succeeded; the caller must check the HandleCount value to determine the number of members actually removed from the permission group
PTSDK_STATUS_NETWORK_ERROR	A network error has occurred while processing the call. This might happen if Username, Password or TargetURL are incorrect. For other possible reasons see shared error code table.

4.6.5.4 ISVS_GetAPIVersionEx

ISVS_GetVersionEx is the extended version of ISVS_GetVersion designed to accommodate providing the client certificate used for Mutual Authentication.

The ISVS_GetAPIVersionEx call is sent by an application to get the ISVS API version supported by the Intel AMT device. The application does not have to register with the Intel AMT device before calling this API.

Function Header

```

PT_STATUS ISVS_GetAPIVersionEx(
    ISVS_VERSION          *Version,
    wchar_t               *Username // optional,
    wchar_t               *Password // optional,
    CHAR                  *TargetUrl ,
    SESSION_AUTHENTICATION_INFO *AuthInfo // optional
);

```

Function Parameters

Parameters	Input/output	Description
Version	Output	The ISV version supported by the Intel AMT device
Username	Input	A pointer to a null terminated string used to authenticate with a remote Intel AMT device using HTTP Digest Authentication. These credentials will be used to establish authentication in all subsequent calls during an established session. This parameter must be specified when communicating with a local Intel AMT Release 2.0 device or later release. This parameter is not used when communicating with a local Intel AMT Release 1.0 device. In this case this parameter should be an empty string. The username must meet the criteria defined in the <i>Network Interface Guide</i> .
Password	Input	A pointer to a null terminated string used to authenticate with a remote Intel AMT device, using Http digest Authentication. This parameter must be specified when communicating with a local Intel AMT Release 2.0 device or later release. This parameter is not used when communicating with a local Intel AMT Release 1.0 device, in this case this parameter should be an empty string. The password must meet the criteria defined in the <i>Network Interface Guide</i> .
TargetURL	Input	A pointer to a null terminated string that specifies the URL pointing to the Storage service on a remote host. Registration and all subsequent storage calls for the current session will be sent to the specified URL. This parameter must not be null. To specify access to the local host when Intel AMT is not configured for TLS , use the local host machine name or "localhost" or 127.0.0.1 as the machine address (e.g. http://localhost:16992/StorageService). When Intel AMT is configured for TLS , the host name in the URL must match the host name signed in the Intel AMT certificate.
AuthInfo	Input	Required if mutual authentication or Kerberos is enabled on the machine specified by TargetURL, otherwise it may be NULL. See ISVS_CreateAuthInfo

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request succeeded; the caller must check the HandleCount value to determine the number of members actually removed from the permission group

Status	Description
PTSDK_STATUS_NETWORK_ERROR	A network error has occurred while processing the call. This might happen if Username, Password or TargetURL are incorrect. For other possible reasons see shared error code table.

4.6.5.5 ISVS_GetBytesAvailable

ISVS_GetBytesAvailable is sent by a registered ISV application to determine the number of bytes currently available for it to allocate. This command can be used if the results of the [ISVS_AllocateBlock](#) return a status of PT_STATUS_NOT_ENOUGH_STORAGE.

Function Header

```
PT_STATUS ISVS_GetBytesAvailable (
    SESSION_HANDLE SessionHandle,
    uint32          *BytesAvailable
);
```

Function Parameters

Parameters	Input/output	Description
SessionHandle	Input	The session handle of the application that is sending the message
BytesAvailable	Output	A pointer to the number of bytes available to the application, identified by SessionHandle for allocation

Function Return Status

See [Shared Error Codes](#) table for additional values

Status	Description
PT_STATUS_SUCCESS	Request completed successfully

5 Appendix A: Application Migration Flow-Chart

The following flowchart shows the steps required to migrate an application to a new UUID. The process requires registering two instances of the application, one with the old UUID and one with the new UUID. The migration process then proceeds, block by block, taking the properties of a block associated with the old UUID and creating a block using the new UUID. The properties include the data in the block, the attributes and the permission groups linked to the block. It may be necessary to release the block under the old UUID so that there is enough non-volatile memory to allocate under the new UUID.

After migrating all blocks, delete the application instance with the old UUID from Intel AMT using `ISVS_Uninitialize`.

